Project: Enhanced Audio Streaming Interface

Document: Specification

Status: Version 1.0 - 25 May 1999

Authors: Felix Bertram
Markus Fritze
Nikolaus Gerteis

EASI MOTIVATION

To interface PC-based hard-disk-recording applications with contemporary audio hardware, driver software is needed abstracting the hardware's capabilities to a common interface. There is a variety of different interfaces available today, most importantly, Microsoft's MME and DirectSound drivers for Windows, Apple's SoundManager drivers for MacOS, and Steinberg's proprietary ASIO architecture which aims to be cross-platform. None of the above interfaces meets all of the requirements of sophisticated hard-disk-recording solutions. These architectural constraints were the true motivation for Emagic's engineers to develop a new concept allowing better product solutions: EASI, the Enhanced Audio Streaming Interface. This document describes the EASI technology, providing a background and a starting point for your own EASI developments. Enjoy!

A DOZEN REASONS FOR DOING IT THE EASI WAY

In comparison with all of the existing interfaces, EASI provides numerous advantages for both hardware and software manufacturers. Following you will find those of most importance.

1. *EASI is a system solution.* Most contemporary interfaces have been created to ease software design. Consequently this leads to heavy constraints on the hardware design and continues to limit the feature sets. EASI elegantly combines the requirements of both worlds, hardware and software, allowing superior system solutions with increased feature sets and performance.
2. *EASI is designed for professional audio.* Professional audio has its own requirements on software and hardware. From multi-channel support via full-duplex operation, to sample-accurate synchronization, EASI is designed to truly meet those professional requirements.
3. *EASI performs better.* EASI decreases the CPU load significantly by making it possible to take advantage of hardware accelerator functions and by minimizing data transfers and format conversions. This means increased stability, more functionality and more DSP power.
4. *EASI makes your hardware more distinctive.* Unlike other APIs, EASI can fully support the distinctive features provided by your hardware, namely input monitoring, level metering and hardware mixing, thus giving hardware manufacturers much more options for creating competitive products.
5. *EASI makes your hardware more cost-effective.* EASI can emulate important hardware features by software, allowing very cost-effective system solutions. In contrast to other APIs, there is no additional software development effort required on your end. Instead, you can simply delegate the desired features to the EASI host.
6. *EASI simplifies your driver design.* EASI plug-ins are very thin layers, mapping your hardware features to a standardized API. There is no redundant code to be implemented again and again, therefore, EASI plug-ins can be created in a relatively short time.

7. *EASI is truly cross-platform.* EASI plug-ins are highly platform-independent. This enables you to ship your product simultaneously on Windows and Mac with very little additional development effort. The same applies for future operating systems, making EASI a sound investment in tomorrow's technology.
8. *EASI is state-of-the-art.* The EASI API is based on object-oriented design. This further eases integration with contemporary driver software, increases code reusability and protects your development investment.
9. *EASI is a public standard.* EASI technology comes to you at no cost right from the start. There is no licensing or non-disclosure agreement required to implement EASI. This is not limited to the creation of EASI plug-ins but also includes the development of EASI hosts, ensuring that EASI will become a widely accepted standard very quickly
10. *EASI is tried and tested.* EASI is not vaporware. It has already been proven to be a versatile yet easy-to-implement interface. EASI is the core technology of Logic Audio's 4.0 audio engine.
11. *EASI is standardized and documented.* EASI is clearly defined by documentation and sample code to ensure a complete and common understanding of its behavior. This further speeds up development and reduces compatibility issues for both hardware and software manufacturers.
12. *EASI is future-proof.* EASI is prepared to follow technology progression. Emagic is happy to receive feedback and input from hardware and software manufacturers to keep the EASI standard alive and up-to-date.

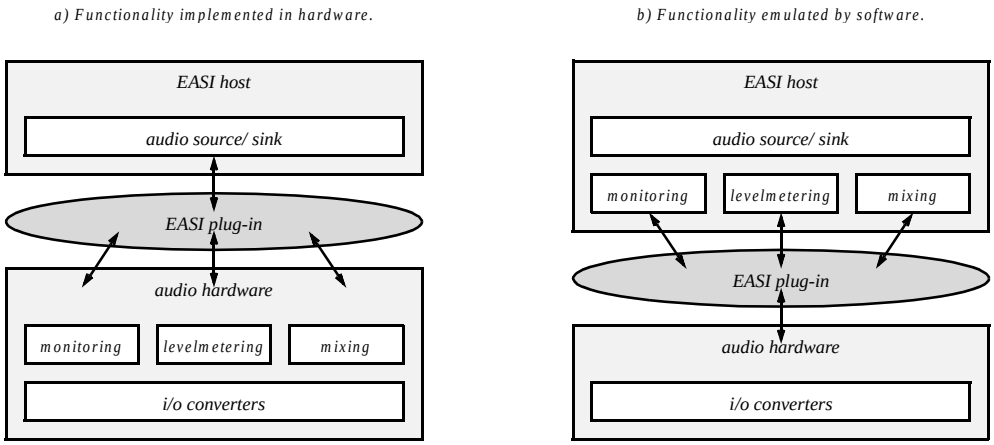
EASI KEY CONCEPTS

Like all plug-in architectures, EASI consists of two main software entities: The plug-ins and the host. The host is part of the application and contains all algorithms required to generate audio data and control EASI plug-ins. Every plug-in is a small piece of code provided by the hardware manufacturer, interfacing between the hardware and the host.

State-of-the-art professional audio systems are typically based on multi-channel full-duplex hardware, very likely with higher bit resolutions or sample rates than 16bit/ 44.1kHz and often targeting more than a single operating system. APIs defined for general-purpose operating systems are simply not powerful enough to meet professional requirements, hence the development of alternative concepts driven by the music industry.

Available hardware differs over a wide range characterized by the number of I/Os, the supported data formats, internal data processing, on-board routing and by DSP capabilities. Typically, one of two problems occur: Either the API is limiting the hardware design or the API is not powerful enough to exploit features provided by the hardware. Consequently the performance is decreased and the product becomes less distinctive.

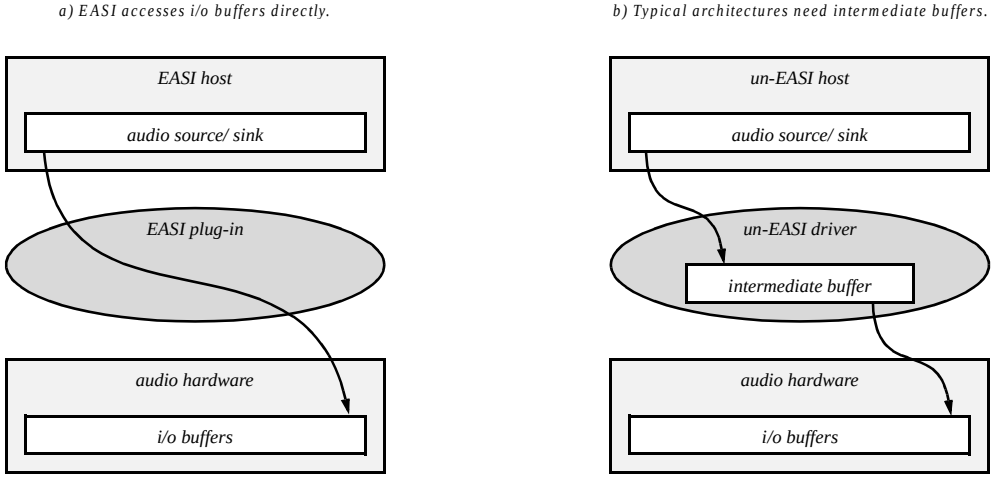
Figure 1: Hardware implementation vs. software emulation.



EASI addresses these issues by defining a powerful data streaming protocol, which is scalable to benefit from hardware accelerated features like mixing, input-monitoring and level metering. Features provided by the hardware are resulting in an immediate increase in performance and usability, while being emulated by the host software, should they not be incorporated in the hardware design.

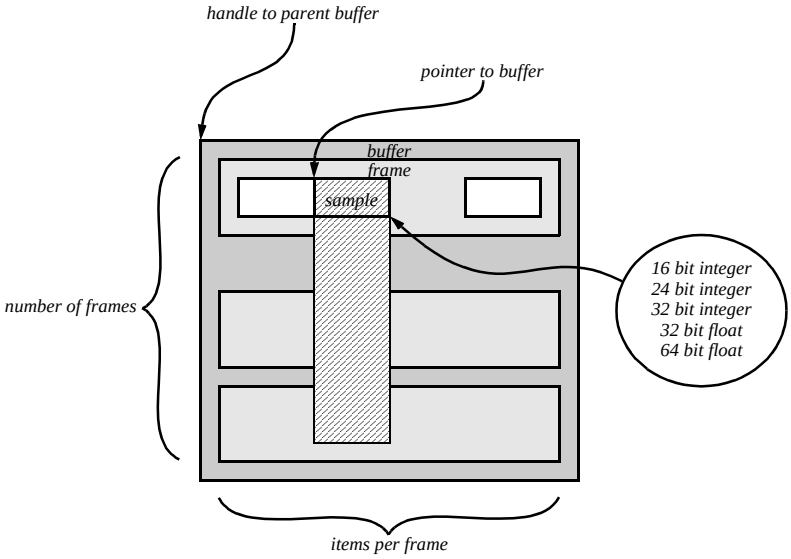
To achieve maximum audio streaming performance, the number of data transfers and conversions must be minimized. If host and hardware are incompatible in buffer format, size or toggling scheme, intermediate buffers are required. This means that additional transfers or conversions become unavoidable for the data streaming between the mentioned intermediate buffers and the hardware's I/O buffers.

Figure 2: EASI minimizes data transfers.



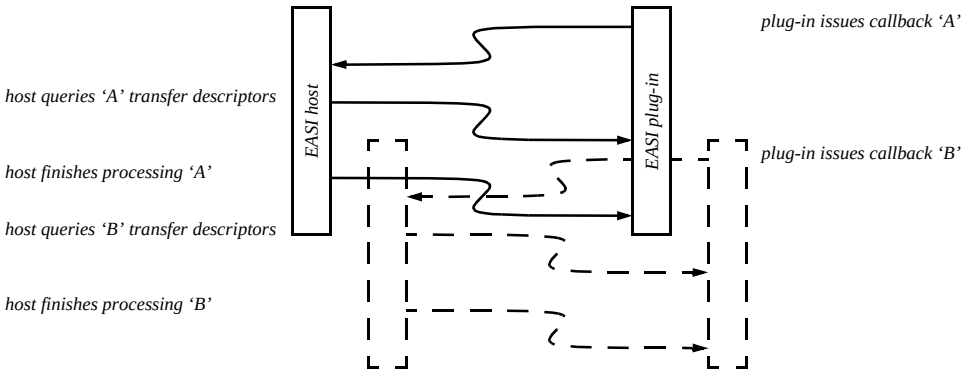
EASI supports I/O buffers of any interleaved structure. Interleaved structures are often handy in hardware design, since typical I/O converters are stereo. EASI describes buffer layouts using transfer descriptors. Besides the information on buffer base address, length and sample format, every transfer descriptor contains information about its parent buffer and its interleaving scheme. Integer and floating-point samples are supported with different resolutions and varying unit value.

Figure 3: EASI transfer descriptor.



Instead of fixing buffer size and toggling scheme during system start-up, EASI plug-ins hand out their transfer descriptors for every I/O channel and every callback. This increased flexibility is especially useful in cases where the hardware does not work with standard double-buffered I/O. To allow additional asynchronous data processing, the host notifies the plug-in on completion.

Figure 4: EASI callback protocol.



EASI DETAILS

The EASI API consists of several function groups, each of them being responsible for a subset of the plug-in's functionality. The following section gives you a brief overview about these function groups.

EASI plug-in instantiation

EASI defines a plug-in architecture of dynamically loaded software modules. As module enumeration and loading is platform dependent, the EASI plug-in instantiation is defined differently on the supported operating systems.

Win32 COM implementation

The Win32 implementation is based on COM. COM is a proven mechanism to instantiate object-oriented software components.

- The host compiles a list of all available EASI devices. This is done by scanning *HKEY_LOCAL_MACHINE\SOFTWARE\EASI*. Every EASI plug-in places a key here, the name of the key is used as the name of the plug-in. The key contains a mandatory *CLSID* value, which is the COM class id used to instantiate the EASI plug-in. Further values may be stored in the registry, containing additional private information required by the plug-in.
- The host instantiates EASI plug-ins by calling *CoCreateInstance*. It passes the *CLSID* of the plug-in, a pointer to its own *IID_IEASIhost_1* and the requested interface which is *IID_IUnknown*:

```
extern REFCLSID  EASIpluginID;    // plug-in's CLSID
extern LPUNKNOWN EASIpluginUnk;   // plug-in's IUnknown
extern LPUNKNOWN EASIhostUnk;     // host's IUnknown

HRESULT hr= ::CoCreateInstance
    ( EASIpluginID,               // rclsid
      EASIhostUnk,                // pUnkOuter
      CLSCTX_INPROC_SERVER,       // dwClsContext
      IID_IUnknown,               // riid
      (LPVOID*)&EASIpluginUnk);   // ppv
```

- The host queries the plug-in's *IUnknown* for *IID_EASIplugin_1*:

```
extern LPUNKNOWN EASIpluginUnk;   // EASI plug-in's IUnknown
extern LPUNKNOWN EASIplugin;      // EASI plug-in
extern IID IID_IEASIplugin_1;     // IID of EASI plug-ins

HRESULT hr= EASIpluginUnk->QueryInterface
    ( IID_IEASIplugin_1,          // riid
      (LPVOID*)&EASIplugin);      // ppvObject
```

- The plug-in queries the host for *IID_EASIhost_1*:

```
extern LPUNKNOWN pUnkOuter;       // host's IUnknown
extern LPUNKNOWN pHost;           // EASI host

HRESULT hr= pUnkOuter->QueryInterface
    ( IID_IEASIhost_1,            // riid
      (LPVOID*)&pHost);           // ppvObject
```

Note: This implies that the EASI host is derived from IUnknown!

- The IIDs of host and plug-in APIs are updated with every new API version.

MacOS implementation

The MacOS implementation is based on the Code Fragment Manager.

- The host compiles a list of all available EASI devices. Device drivers are placed either in a "EASI devices" folder beside the current EASI host application or in a "EASI devices" folder inside the system folder. With MacOS 8.5 the folder is located in the "Application Support" folder inside the system folder.
- The filetype of the device driver is always 'EASI', the creator may be chosen by the developer of the driver.
- We provide complete sample code for the host side and for the client side, so a developer don't have to care about the MacOS specific features of EASI.

Further operating systems

The EASI architecture is not limited to specific operating systems, however the instantiation mechanisms of further operating systems are to be defined. It is Emagic's intention to ensure a consistent implementation across all software platforms. Therefore Emagic is happy to receive your proposal on further or future operating system's instantiation mechanisms and add them to the EASI standard definition.

EASI version control

As EASI is a plug-in architecture, there is a need to verify that host and plug-ins are compatible. To do so, both host and plug-in contain a version id, identifying the EASI version used to create the host/ plug-in. This information is queried using the *GetVersion* API. During start-up the following version control is performed:

- The host verifies that the plug-in is newer than a minimum supported version. In case the verification fails, the host does not load the plug-in.
- The plug-in verifies that the host is newer than a minimum supported version. In case the verification fails, the plug-in returns failure on all subsequent calls.

EASI transport interface

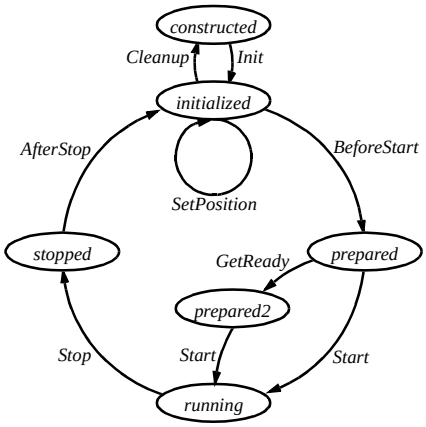
Typical hard-disk-recording applications follow the tape-recorder metaphor. The EASI transport interface controls data streaming and positioning in a similar way. The hardware is controlled by a finite-state-machine with the following states:

- *Constructed.* The plug-in is properly constructed, no additional resources are allocated, the hardware is not acquired.
- *Initialized.* The plug-in is initialized, all additional resources are allocated, the hardware is acquired but idle.
- *Prepared.* The plug-in is positioned and prepared to start streaming with low latency, all resources are allocated, the hardware is either idle, level metering or monitoring.
- *Prepared2.* The plug-in is positioned and prepared to start streaming with zero latency, all resources are allocated, the hardware is ready to start. For most devices this state is equal to the *Prepared* state.

- *Running.* The plug-in is streaming audio, all resources are allocated, the hardware is running.
- *Stopped.* The plug-in finished audio streaming, all resources are allocated, the hardware is stopped. Some recorded buffers may be pending.

The EASI host is responsible for validating all state transitions, there is no need inside the EASI plug-in to do so.

Figure 5: EASI transport state-machine.



The following commands belong to the transport interface.

- *Init.* Allocate all resources and acquire audio hardware.
- *SetPosition.* Set logical audio position to new value. The current logical audio position can be queried using *GetPosition* later on.
- *BeforeStart.* Preload initial playback data, start input-monitoring and level metering as required.
- *GetReady.* Prepare hardware for zero latency start-in. This command may be ignored by most devices.
- *Start.* Begin audio streaming. This command should have zero latency, if preceded by a call to *GetReady.*
- *Stop.* End audio streaming. This command should have zero latency.
- *AfterStop.* Synchronize audio hardware, flush all pending data.
- *Cleanup.* Free all resources including audio hardware.
- *GetInternalError.* Query and clear the current internal error status.

EASI capabilities interface

Every device has its own features, capabilities and properties which are constant at application run-time. The EASI host gathers all this device-specific information during start-up and configures its internal data processing accordingly. The following APIs belong to the capabilities interface:

- *GetNumChannels*. Query the number of hardware inputs and outputs.
- *GetChannelName*. Query name of hardware inputs or outputs.
- *GetNumTracks*. Query the maximum number of virtual tracks supported.
- *GetNumSampleRates, GetSampleRate, GetPitchRange*. Query the number of sample rates supported, their nominal values and the current pitch range.
- *IsConst, GetStdXfer.* Query common parameters of transfer descriptors to support host processing optimizations.

EASI streaming interface

To stream data EASI plug-ins issue *Streaming* callbacks to the host. EASI callbacks use a single parameter, the so-called callback handle, which is a ‘magic cookie’ by which the host queries additional information from the EASI plug-in. The following APIs belong to the streaming interface:

- *IsSynchronous*. EASI is designed to support multi-threaded applications. *Streaming* processing may be performed either synchronous or asynchronous at the choice of the EASI host, except when *IsSynchronous* returns true. In these cases the host must synchronize the *Streaming* callback with the plug-in.
- *GetPosition*. In addition to the current playback or recording position, EASI plug-ins maintain a streaming position for all callbacks issued, which is used to control the EASI host’s data source/ sink.
- *NotifyDone*. As the EASI host may defer processing asynchronously, the plug-in must be notified on processing completion via *NotifyDone*.
- *GetXfer.* EASI describes buffer layouts using transfer descriptors, every channel has its own set of transfer descriptors. The EASI host queries transfer descriptors from the EASI plug-ins using *GetXfer.*
- *GetXferLength*. EASI requires all transfer descriptors belonging to the same callback handle to be of equal length. This common length is queried using *GetXferLength*.

EASI control interface

All functionality required to control hardware features like input-monitoring, level metering or track mixing is accessed using the EASI control interface. The following APIs belong to the control interface:

- *SetSampleRate*. The host chooses a nominal sample rate via *SetSampleRate*.
- *SetPitch*. The host specifies a deviation from the nominal sample rate via *SetPitch*.
- *GetLevelmeter*. In case an EASI plug-in provides hardware accelerated level meters, it passes metering information to the host via *GetLevelmeter.*
- *SetRouting.* In case an EASI plug-in provides hardware accelerated input monitoring or track mixing, it receives its routing information via *SetRouting*.
- *SetGain*. In case an EASI plug-in provides hardware accelerated input monitoring or track mixing, it receives its gains via *SetGain*.
- *SetupDialog*. The host asks the plug-in to display its control panel, if available, via *SetupDialog*.

EASI host interface

Most of the EASI functionality is provided by the host. EASI plug-ins access the host using a set of callbacks, these callbacks are referred to as the EASI host interface. The host interface provides the following APIs:

- *Streaming*. EASI plug-ins call *Streaming* to stream a buffer of audio and delegate its audio mixing to the EASI host.
- *Monitoring*. EASI plug-ins call *Monitoring* to delegate input monitoring and level metering to the EASI host.

EASI REFERENCE

The following section gives detailed information about syntax and semantic of all EASI APIs in alphabetical order.

EASIplugin::AfterStop

Declaration

```
EASIapi(EASIerror)              // - error code
AfterStop                       // Cleanup after audio streaming.
( void)                         // -
```

Description

Cleanup after audio streaming. Typical EASI plug-ins save their pending recording buffers to disk and synchronize with their hardware during *AfterStop*. After returning from *AfterStop*, a plug-in must not call the host's streaming APIs any longer.

Example

```
void EndAudio(EASIplugin* plugin)
{ plugin->Stop();
  plugin->AfterStop();
  plugin->Cleanup();
}
```

See Also

```
Stop, Cleanup, BeforeStart
```

EASIplugin::BeforeStart

Declaration

```
EASIapi(EASIerror)              // - error code
BeforeStart                     // Prepeare for audio streaming.
( EASIiomask,                   // - channel mask playback
  EASIiomask)                   // - channel mask recording
```

Description

Prepare for audio streaming. Typical EASI plug-ins load their initial playback data from disk, start monitoring and level metering during *BeforeStart*.

A separate bit mask is passed to specify the activity status of playback and recording channels, with each channel corresponding to the bit at the position with the channel's index. Most implementations will not create or verify these masks but use the predefined values *EASIiomask_all* and *EASIiomask_none* instead.

Example

```
void StartAudio(EASIplugin* plugin, bool fPlayback, bool fRecord)
{ EASIiomask mp= fPlayback? EASIiomask_all: EASIiomask_none;
  EASIiomask mr= fRecord?   EASIiomask_all: EASIiomask_none;
  plugin->Init();
  plugin->BeforeStart( mp, mr);
  plugin->Start();
}
```

See Also

```
Init, Start, AfterStop
```

EASIplugin::Cleanup

Declaration

```
EASIapi(EASIerror)              // - error code
Cleanup                         // Close plug-in.
( void)                         // -
```

Description

Close plug-in. Typical EASI plug-ins free all resources and release their hardware during *Cleanup*.

Example

```
void PlayAudio(EASIplugin* plugin)
{ plugin->Init();
  plugin->SetPosition(0);
  plugin->BeforeStart(EASIiomask_all, EASIiomask_none);
  plugin->Start();

  do
  { // playback in background...
  } while(!_kbhit());

  plugin->Stop();
  plugin->AfterStop();
  plugin->Cleanup();
}
```

See Also

```
Stop, AfterStop, Init
```

EASIplugin::Constructor

Declaration

```
EASIplugin                      // Constructor.
( EASIhost* p)                  // - EASI host
```

Description

Create EASI plug-in object. Typical EASI init their internal data structures here, while allocation of any further resources or acquiry of hardware is done during *Init*.

Note that instantiation of EASI plug-ins is platform dependent and therefore the EASI host does not call the constructor of any EASI plug-in directly.

Example

```
-
```

See Also

```
Init
```

EASIplugin::GetChannelName

Declaration

```
EASIapi(const char*)            // - name
GetChannelName                  // Query name of physical i/o.
( EASIiomode,                   // - i/o mode
  EASIindex)                    // - physical i/o
```

Description

Query name of physical input or output.

Example

```
void PrintOutputs(EASIplugin* plugin)
{ printf("Outputs:\n");
  EASIindex n= plugin->GetNumChannels(EASIiomode_play);
  for(EASIindex out= 0; out< n; out++)
  { const char* name= plugin->GetChannelName(EASIiomode_play, out);
    printf("%s\n", name);
  }
}
```

See Also

GetNumChannels

EASIplugin::GetInternalError

Declaration

```
EASIapi(EASIerror)              // - error code
GetInternalError                // Query and reset streaming error.
( void)                         // -
```

Description

Most EASI APIs return error codes to indicate success or failure immediately. To detect failure of internal background processing, the EASI host calls *GetInternalError*, which returns the first error that occurred since the last query.

Example

```
void WatchPlayback(EASIplugin* plugin)
{ while(1)
  { EASIerror err= plugin->GetInternalError();

    if (err)
    { // recover from error
    }

    { // do something useful
    }
  }
}
```

See Also

-

EASIplugin::GetLevelmeter

Declaration

```
EASIapi(EASIvolume)             // - level
GetLevelmeter                   // Query current input level.
( EASIindex,                    // - i/o mode
  EASIindex)                    // - physical input
```

Description

Query current level meter value. Levels are measured in floating point, with a nominal value of *EASIvolume_0dB*. To delegate metering to the EASI host, EASI plug-ins return *EASIvolume_illegal* here. In this case level meters are computed during *Monitoring*.

Example

```
void DoMetering(EASIplugin* plugin)
{ plugin->SetPosition(0);
  plugin->BeforeStart(EASIiomask_none, EASIiomask_all);
  plugin->Start();

  do
  { float level= plugin->GetLevelmeter(EASIiomode_rec, 0);
    int   level2= static_cast<int>(40*level);

    for (int i= 0; i< level2; i++) printf("*");
    printf(">>\n");
  } while (plugin->GetPosition()< 5*44100);

  plugin->Stop();
  plugin->AfterStop();
}
```

See Also

-

EASIplugin::GetNumChannels

Declaration

```
EASIapi(EASIindex)              // - number
GetNumChannels                  // Query number of physical i/os.
( EASIiomode)                   // - i/o mode
```

Description

Query number of physical inputs and outputs provided by this hardware.

Example

```
void PrintNumChannels(EASIplugin* plugin)
{ printf("# of inputs: %ld, # of outputs %ld\n",
         plugin->GetNumChannels(EASIiomode_rec),
         plugin->GetNumChannels(EASIiomode_play));
}
```

See Also

GetChannelName, GetXfer, GetNumTracks

EASIplugin::GetNumSampleRates

Declaration

```
EASIapi(EASIindex)        // - number
GetNumSampleRates         // Query number of supported samplerates.
( void)                   // -
```

Description

Query the number of sample rates supported by this device. The EASI host queries the nominal value of an indexed sample rate using *GetSampleRate*.

Example

```
void PrintSamplerates(EASIplugin* plugin)
{ EASIindex n= plugin->GetNumSampleRates();
  for (EASIindex i= 0; i< n; i++)
  { EASIsrate srate= plugin->GetSampleRate(i);
    printf("%ld\n", srate);
  }
}
```

See Also

GetSampleRate, GetPitchRange

EASIplugin::GetNumTracks

Declaration

```
EASIapi(EASIindex)              // - number
GetNumTracks                    // Query number of virtual tracks.
( void)                         // -
```

Description

Query maximum number of virtual tracks supported by this hardware.

Devices providing their own hardware accelerated mixing return their maximum mixing capability here. The EASI host will setup an audio stream for every virtual track in this case.

Devices that rely on the EASI host for mixing return zero and let the EASI host chose an arbitrary number of virtual tracks, depending on the CPU's computing power. The EASI host will setup an audio stream for every physical I/O in this case.

Example

```
EASIindex GetNumStreams(EASIplugin* plugin)
{ EASIindex tracks=   plugin->GetNumTracks();
  EASIindex channels= plugin->GetNumChannels(EASIiomode_play);
  EASIindex streams=  tracks? tracks: channels;
  return streams;
}
```

See Also

GetNumChannels

EASIplugin::GetPitchRange

Declaration

```
EASIapi(bool)                   // - true, if pitchable
GetPitchRange                   // Query pitch range from indexed samplerate.
( EASIindex,                    // - samplerate [index]
  EASIsrate*,                   // - min deviation [Hz]
  EASIsrate*)                   // - max deviation [Hz]
```

Description

Query pitch range currently supported by this hardware. Pitch is expressed as a frequency deviation from the nominal sample rate measured in Hz.

Example

```
void PrintPitchRanges(EASIplugin* plugin)
{ EASIindex n= plugin->GetNumSampleRates();
  for (int i= 0; i< n; i++)
  { EASIsrate srate= plugin->GetSampleRate(i);
    EASIsrate min;
    EASIsrate max;
    plugin->GetPitchRange(i, &min, &max);
    printf("%ld (%ld..%ld)\n", srate, min, max);
  }
}
```

See Also

GetSampleRate, SetPitch

EASIplugin::GetPosition (1)

Declaration

```
EASIapi(EASIframes)             // - position [frames]
GetPosition                     // Query current device position.
( void)                         // -
```

Description

Query current device position. The device position is the position that is currently converted by the device’s CODECs.

Example

```
void PlayFiveSeconds(EASIplugin* plugin)
{ EASIframes pos1= plugin->GetPosition();
  EASIframes pos2= pos1 + 5*44100;

  plugin->Start();

  while(plugin->GetPosition()< pos2)
  { // do nothing
  }

  plugin->Stop();
}
```

See Also

SetPosition

EASIplugin::GetPosition (2)

Declaration

```
EASIapi(EASIframes)             // - position [frames]
GetPosition                     // Query streaming position.
( EASIhandle,                   // - callback handle
  EASIiomode)                   // - i/o mode
```

Description

Query streaming position of given callback. The streaming position is the position of the first frame contained in the buffers described by *GetXfer* calls with the given handle.

Example

```
void PrintLatency(EASIplugin* plugin, EASIhandle h)
{ EASIframes pos=     plugin->GetPosition();
  EASIframes oStream= plugin->GetPosition(h, EASIiomode_play);
  EASIframes iStream= plugin->GetPosition(h, EASIiomode_rec);

  printf("output latency= %ld\n", oStream-pos);
  printf("input  latency= %ld\n", pos-iStream);
}
```

See Also

GetXfer, Streaming

EASIplugin::GetReady

Declaration

```
EASIapi(EASIerror)              // - error code
GetReady                        // Prepare for zero-latency start-in.
( void)                         //  -
```

Description

Prepare for zero-latency start-in. In case the EASI host calls *GetReady* it expects the following *Start* command to behave as a real-time trigger with zero latency. Hosts that do not require zero-latency start-in may omit the *GetReady* command. Typical EASI plug-ins shut down all of their time-consuming processes like monitoring and level metering here.

Example

```
void StartExact(EASIplugin* plugin)
{ plugin->BeforeStart(EASIiomask_all, EASIiomask_none);
  plugin->GetReady();

  extern void WaitForTrigger(void);
  WaitForTrigger();

  plugin->Start();
}
```

See Also

Start, Stop

EASIplugin::GetSampleRate

Declaration

```
EASIapi(EASIsrate)              // - samplerate [Hz]
GetSampleRate                   // Query nominal samplerate from index.
( EASIindex)                    // - samplerate [index]
```

Description

Query nominal sample rate from given index.

Example

```
bool Can44k1(EASIplugin* plugin)
{ EASIindex n= plugin->GetNumSampleRates();
  for(EASIindex i= 0; i< n; i++)
  { if (plugin->GetSampleRate(i)== 44100) return true;
  }
  return false;
}
```

See Also

GetNumSampleRates

EASIplugin::GetStdXfer

Declaration

```
EASIapi(const EASIxfer*)        // - xfer
GetStdXfer                      // Query constant xfer descriptor fields.
( EASIiomode)                   // - i/o mode
```

Description

EASI plug-ins generally create new transfer descriptors for every channel and every callback issued. To avoid conditional code, EASI plug-ins may declare certain parameters of future transfer descriptors to remain constant. If parameters are declared constant using *IsConst* and *GetStdXfer*, the according values must not change during future *GetXfer* calls.

Example

```
bool CheckStdXfer(EASIplugin* plugin, EASIhandle h, EASIiomode m)
{ const EASIxfer* x1= plugin->GetStdXfer(m);

  EASIindex n= plugin->GetNumChannels(m);
  for(EASIindex i= 0; i< n; i++)
  { const EASIxfer* x2= plugin->GetXfer(h, m, i);

    if (plugin->IsConst(m, offsetof(EASIxfer, lNumFrames))
    &&  (x1->lNumFrames     != x2->lNumFrames))     return false;

    if (plugin->IsConst(m, offsetof(EASIxfer, lItemsPerFrame))
    &&  (x1->lItemsPerFrame != x2->lItemsPerFrame)) return false;

    if (plugin->IsConst(m, offsetof(EASIxfer, eFmt))
    &&  (x1->eFmt           != x2->eFmt))           return false;

    if (plugin->IsConst(m, offsetof(EASIxfer, xFmt))
    &&  (x1->xFmt           != x2->xFmt))           return false;
  }

  return true;
}
```

See Also

IsConst, GetXfer

EASIplugin::GetVersion

Declaration

```
EASIapi(EASIindex)              // - EASI version
GetVersion                      // Query version number.
( void)                         // -
```

Description

EASI requires every host and every plug-in to provide information about the EASI version it complies with. Hosts and plug-ins may chose not to cooperate with each other in case their version numbers mismatch.

Example

-

See Also

-

EASIplugin::GetXfer

Declaration

```
EASIapi(const EASIxfer*)        // - xfer
GetXfer                         // Query xfer descriptor.
( EASIhandle,                   // - callback handle
  EASIiomode,                   // - i/o mode
  EASIindex)                    // - physical i/o or virtual track
```

Description

Query transfer descriptor for given callback handle, I/O mode and channel. Transfer descriptors are used to pass buffer information from EASI plug-ins to the EASI host. The host must not store transfer descriptors any longer than the call to *NotifyDone*, except when declared constant using *GetStdXfer* and *IsConst*.

In case *GetNumTracks* returned zero, the index of a physical I/O is passed to *GetXfer*.

In case *GetNumTracks* returned a non-zero value, the index of a virtual track is passed to *GetXfer*.

Example

-

See Also

`GetXferLength, GetPosition, GetStdXfer, IsConst, Streaming`

EASIplugin::GetXferLength

Declaration

```
EASIapi(EASIframes)             // - length [frames]
GetXferLength                   // Query common xfer length.
( EASIhandle)                   // - callback handle
```

Description

For every callback being issued by an EASI plug-in, the length returned by *GetXferLength* equals the length encoded in all transfer descriptors returned by calls to *GetXfer* with the same handle.

Example

```
bool CheckXferLength(EASIplugin* plugin, EASIhandle h, EASIiomode m)
{ EASIframes l1= plugin->GetXferLength(h);

  EASIindex n= plugin->GetNumChannels(m);
  for(EASIindex i= 0; i< n; i++)
  { EASIframes l2= plugin->GetXfer(h, m, i)->lNumFrames;

    if (l1 != l2) return false;
  }

  return true;
}
```

See Also

GetXfer, GetPosition, GetStdXfer, IsConst, Streaming

EASIplugin::Init

Declaration

```
EASIapi(EASIerror)              // - error code
Init                            // Open plug-in.
( void)                         // -
```

Description

Open plug-in. Typical EASI plug-ins allocate all necessary resources, acquire their hardware and reset during *Init*.

Example

```
void PlayAudio(EASIplugin* plugin)
{ plugin->Init();
  plugin->SetPosition(0);
  plugin->BeforeStart(EASIiomask_all, EASIiomask_none);
  plugin->Start();

  do
  { // playback in background...
  } while(!_kbhit());

  plugin->Stop();
  plugin->AfterStop();
  plugin->Cleanup();
}
```

See Also

BeforeStart, Start, Cleanup

EASIplugin::IsConst

Declaration

```
EASIapi(bool)                   // - true, if const
IsConst                         // Check for constant xfer field.
( EASIiomode,                   // - i/o mode
  EASIindex)                    // - offsetof xfer field
```

Description

EASI plug-ins generally create new transfer descriptors for every channel and every callback issued. To avoid conditional code, EASI plug-ins may declare certain parameters of future transfer descriptors to remain constant. If parameters are declared constant using *IsConst* and *GetStdXfer*, the according values must not change during future *GetXfer* calls.

Example

```
bool CheckConst(EASIplugin* plugin, EASIiomode m)
{ if (plugin->IsConst(m, offsetof(EASIxfer, pBuf)))           return true;
  if (plugin->IsConst(m, offsetof(EASIxfer, hBuf)))           return true;
  if (plugin->IsConst(m, offsetof(EASIxfer, lItemsPerFrame))) return true;
  if (plugin->IsConst(m, offsetof(EASIxfer, lNumFrames)))     return true;
  if (plugin->IsConst(m, offsetof(EASIxfer, eFmt)))           return true;
  if (plugin->IsConst(m, offsetof(EASIxfer, xFmt)))           return true;
  return false;
}
```

See Also

```
GetStdXfer
```

EASIplugin::IsSynchronous

Declaration

```
EASIapi(bool)                   // - true, if sync'ed
IsSynchronous                   // Check for synchronous processing.
( EASIhandle)                   // - callback handle
```

Description

EASI distincts between synchronous and asynchronous callback processing. Typical EASI plug-ins require synchronous operation during *BeforeStart* and *AfterStop*, while they prefer asynchronous operation during background data streaming. The host queries the current processing mode using the *IsSynchronous* function.

Example

```
extern EASIplugin* plugin;

EASIerror EASIhost::Streaming(EASIhandle h)
{ if (plugin->IsSynchronous(h))
  {
    { //... synchronous processing...
    }

    plugin->NotifyDone(h, EASIiomode_play);
    plugin->NotifyDone(h, EASIiomode_rec);
  }
  else
  {
    { //... asynchronous processing...
    }

    plugin->NotifyDone(h, EASIiomode_play);
    plugin->NotifyDone(h, EASIiomode_rec);
  }

  return EASIerror_success;
}
```

See Also

```
-
```

EASIplugin::NotifyDone

Declaration

```
EASIapi(void)                   // -
NotifyDone                      // Async completion notification.
( EASIhandle,                   // - callback handle
  EASIiomode)                   // - i/o mode
```

Description

Whenever the EASI host finished processing a *Streaming* callback, it notifies the plug-in by calling *NotifyDone*. Typical EASI plug-ins execute any additional postprocessing of streamed data here. It is mandatory that the EASi host generates two calls to *NotifyDone* for each call to *Streaming*, one for playback and one for recording.

Example

```
extern EASIplugin* plugin;

EASIerror EASIhost::Streaming(EASIhandle h)
{ { //... data processing...
  }

  plugin->NotifyDone(h, EASIiomode_play);
  plugin->NotifyDone(h, EASIiomode_rec);

  return EASIerror_success;
}
```

See Also

Streaming

EASIplugin::SetGain

Declaration

```
EASIapi(EASIerror)              // - error code
SetGain                         // Set mixer gains.
( EASIiomode,                   // - i/o mode
  EASIindex,                    // - physical input or virtual track
  EASIvolume,                   // - mixer gain: left
  EASIvolume)                   // - mixer gain: right
```

Description

The EASI mixer model routes every mono source to two mono destinations with separate gains. To support EASI plug-ins with own mixer capabilities, the EASI host calls *SetRouting* and *SetGain* with the current mixer settings. Devices not providing built-in mixers may ignore this command.

Example

```
void SetupStereoMonitoring(EASIplugin* plugin)
{ plugin->SetRouting(EASIiomode_rec, 0, 0, 1);
  plugin->SetRouting(EASIiomode_rec, 1, 0, 1);

  plugin->SetGain(EASIiomode_rec, 0, EASIvolume_0dB, EASIvolume_silence);
  plugin->SetGain(EASIiomode_rec, 1, EASIvolume_silence, EASIvolume_0dB);
}
```

See Also

SetRouting

EASIplugin::SetPitch

Declaration

```
EASIapi(EASIerror)              // - error code
SetPitch                        // Assign new frequency deviation.
( EASIsrate)                    // - deviation [Hz]
```

Description

Set deviation from nominal sample rate. Frequency deviation is measured in Hz.

Example

```
void SetExactSrate(EASIplugin* plugin, EASIsrate nom, EASIsrate exact)
{ EASIindex n= plugin->GetNumSampleRates();
  for(EASIindex i= 0; i< n; i++)
    if (plugin->GetSampleRate(i)== nom)
    { plugin->SetSampleRate(i);
      break;
    }

  EASIsrate pitch= exact-nom;
  plugin->SetPitch(pitch);
}
```

See Also

GetPitchRange

EASIplugin::SetPosition

Declaration

```
EASIapi(EASIerror)              // - error code
SetPosition                     // Set device position.
( EASIframes)                   // - position [frames]
```

Description

Set the device position to a new value measured in frames. The device position is the position that is currently converted by the device's CODECs. The *SetPosition* is not accepted in any other state than *initialized*.

Example

```
void PlayFromPos(EASIplugin* plugin, EASIframes pos)
{ plugin->SetPosition(pos);
  plugin->BeforeStart(EASIiomask_all, EASIiomask_none);
  plugin->Start();
}
```

See Also

GetPosition

EASIplugin::SetRouting

Declaration

```
EASIapi(EASIerror)              // - error code
SetRouting                      // Set output routing.
( EASIiomode,                   // - i/o mode
  EASIindex,                    // - physical input or virtual track
  EASIindex,                    // - physical output: left
  EASIindex)                    // - physical output: right
```

Description

The EASI mixer model routes every mono source to two mono destinations with separate gains. To support EASI plug-ins with own mixer capabilities, the EASI host calls *SetRouting* and *SetGain* with the current mixer settings. Devices not providing built-in mixers may ignore this command.

Example

```
void RouteAllTracksTo1stPair(EASIplugin* plugin)
{ for (EASIindex out= 0; out< plugin->GetNumTracks(); out++)
  { // route all virtual tracks to first output pair
    plugin->SetRouting(EASIiomode_play, out, 0, 1);
  }
}
```

See Also

SetGain

EASIplugin::SetSampleRate

Declaration

```
EASIapi(EASIerror)              // - error code
SetSampleRate                   // Assign new samplerate.
( EASIindex)                    // - samplerate [index]
```

Description

The EASI host uses the *SetSampleRate* command to assign a new nominal sample rate to the EASI plug-in.

Example

-

See Also

GetNumSampleRates, GetSampleRate

EASIplugin::SetupDialog

Declaration

```
EASIapi(bool)             // - true, to request re-init
SetupDialog               // Open setup dialog box
( const EASIdlgInfo*)     // - additional dialog box info
                                  =0;
```

Description

To control certain features that are not part of the EASI specification, EASI plug-ins may provide a control panel. The control panel is opened via *SetupDialog*. By returning *true*, an EASI plug-in requests that the EASI host re-initializes the plug-in.

Example

-

See Also

-

EASIplugin::Start

Declaration

```
EASIapi(EASIerror)              // - error code
Start                           // Start audio streaming.
( void)                         // -
```

Description

Start audio streaming. This call should have a very low latency, if preceded by a call to *GetReady*, latency should be zero. All time-consuming tasks must be executed during *BeforeStart*. This call may be executed in interrupt context on some target operating systems.

Example

-

See Also

BeforeStart, GetReady, Stop

EASIplugin::Stop

Declaration

```
EASIapi(EASIerror)              // - error code
Stop                            // Stop audio streaming.
( void)                         // -
```

Description

Stop audio streaming. This call should have a very low latency, all time-consuming tasks should be executed during *AfterStop*. This call may be executed in interrupt context on some target operating systems.

Example

-

See Also

AfterStop, Cleanup, Start

EASIhost::Constructor

Declaration

```
EASIhost                        // Constructor.
( const EASIctorInfo*,          // - [in]  construction info
  EASIplugin**)                 // - [out] EASI plugin
```

Description

Create EASI host and plug-in objects according to the *EASIctorInfo* passed in. EASI assumes that for every plug-in instance exactly one host exists. Typically host and plug-in init their internal data structures here, while allocation of any further resources or acquiry of hardware is done during *Init*.

Example

-

See Also

-

EASIhost::GetVersion

Declaration

```
EASIapi(EASIindex)              // - EASI version
GetVersion                      // Query version number.
( void)                         // -
```

Description

EASI requires every host and every plug-in to provide information about the EASI version it complies with. Hosts and plug-ins may chose not to cooperate with each other in case their version numbers mismatch.

Example

-

See Also

-

EASIhost::Monitoring

Declaration

```
EASIapi(EASIerror)              // - error code
Monitoring                      // Monitoring/ metering callback.
( EASIhandle)                   // - callback handle
```

Description

EASI plug-ins call *Monitoring*, whenever host assistance for level metering and input monitoring is required. The host will query buffer information via *GetXfer* and *GetXferLength*.

Example

-

See Also

-

EASIhost::Streaming

Declaration

```
EASIapi(EASIerror)              // - error code
Streaming                       // Data streaming callback.
( EASIhandle)                   // - callback handle
```

Description

EASI plug-ins call *Streaming*, whenever host assistance for data streaming is required. The host will query buffer information via *GetXfer* and *GetXferLength*, process the data, and notify the plug-in on completion via *NotifyDone*.

Example

```
extern EASIplugin* plugin;

template<class T> void _xferCopy(const double* sig, const EASIxfer* xfer, T)
{ const long  len=    xfer->lNumFrames;
  const float unit=   xfer->xFmt;
  const long  intrlv= xfer->lItemsPerFrame;

  T* p= reinterpret_cast<T*>(xfer->pBuf);
  for(int i= 0; i< len; i++)
  { *p= static_cast<T>(sig[i] * unit);
    p+= intrlv;
  }
}

void xferCopy(const double* sig, const EASIxfer* xfer)
{ switch(xfer->eFmt)
  { case EASIfmt_i8:  _xferCopy(sig, xfer, char  (0));  break;
    case EASIfmt_i16: _xferCopy(sig, xfer, short (0));  break;
    case EASIfmt_i32: _xferCopy(sig, xfer, long  (0));  break;
    case EASIfmt_f32: _xferCopy(sig, xfer, float (0));  break;
    case EASIfmt_f64: _xferCopy(sig, xfer, double(0));  break;
  }
}

EASIerror EASIhost::Streaming(EASIhandle h)
{ EASIframes len= plugin->GetXferLength(h);
  EASIframes pos= plugin->GetPosition(h, EASIiomode_play);

  EASIindex trk= plugin->GetNumTracks();
  EASIindex cha= plugin->GetNumChannels(EASIiomode_play);
  EASIindex str= trk? trk: cha;

  double* sig= new double[len];
  for (int i= 0; i< len; i++) sig[i]= sin((pos+i)/10.0);

  for(EASIindex idx= 0; idx< str; idx++)
  { const EASIxfer* xfer= plugin->GetXfer(h, EASIiomode_play, idx);
    xferCopy(sig, xfer);
  }

  delete[] sig;

  plugin->NotifyDone(h, EASIiomode_play);
  plugin->NotifyDone(h, EASIiomode_rec);

  return EASIerror_success;
}
```

See Also

GetNumChannels, GetXfer, GetPosition, NotifyDone

EASI.HPP

```
////////1////////2////////3////////4////////5////////6////////7////////8
//                  (C) 1999 Emagic Soft- und Hardware GmbH
// All rights are reserved.  Reproduction in whole or in part is prohibited
// without the written consent of the copyright owner.
//
// Emagic reserves the right to make changes without notice at any time.
// Emagic makes no warranty, expressed, implied or statutory, including but
// not limited to any implied warranty of merchantibility of fitness for any
// particular purpose, or that the use will not infringe any third party
// patent, copyright or trademark.  Emagic must not be liable for any loss
// or damage arising from its use.
// ---------------------------------------------------------------------------
// File:        EASI.hpp
// Description: EASI - enhanced audio streaming interface.
// History:     Felix Bertram, 23-Feb-99, Created.
//              Felix Bertram, 19-May-99, Version 1.0
////////1////////2////////3////////4////////5////////6////////7////////8

#ifndef __EASI_HPP__
#define __EASI_HPP__

//============================================================================
// * * *   b a s i c   t y p e s   &   f o r w a r d s   * * * * * * * * * * *
//============================================================================

#define EASIversion          1               // EASI API version

class EASIplugin;                            // generic EASI plugin class
class EASIhost;                              // generic EASI host class
class EASIctorInfo;                          // generic EASI construction info
class EASIdlgInfo;                           // EASI dialog box info

typedef void*                EASIhandle;     // callback handle
typedef long                 EASIframes;     // # of samples/ frames
typedef long                 EASIindex;      // # of channels, samplerates etc.
typedef unsigned long        EASIiomask;     // mask w/ activity bits

typedef float                EASIvolume;     // volumes
typedef long                 EASIsrate;      // sample rate

typedef enum _EASIerror                      // error codes
{ EASIerror_success=         0,              // successful operation
  EASIerror_notsupported=    1,              // feature not supported
  EASIerror_hardwarenotfound= 2,             // hardware not found
  EASIerror_hardwarefailure= 3,              // hardware failure
  EASIerror_hardwarebusy=    4,              // hardware is busy
  EASIerror_cpuoverload=     5,              // cpu overload
  EASIerror_outofrange=      6,              // parameter out of range
  EASIerror_unknown=         -1              // unknown error
} EASIerror;

typedef enum _EASIiomode                     // i/o modes
{ EASIiomode_play,                           // playback
  EASIiomode_rec                             // recording (full-duplex)
} EASIiomode;

typedef enum
{ EASIfmt_raw=               0,              // raw bytes
  EASIfmt_i8=                1,              // 8 bit integer
  EASIfmt_i16=               2,              // 16bit integer
  EASIfmt_i24=               3,              // 24bit integer
  EASIfmt_i32=               4,              // 32bit integer
  EASIfmt_f32=               5,              // 32bit float
  EASIfmt_f64=               6,              // 64bit double
} EASIfmt;

typedef struct _EASIxfer                     // data xfer descriptor
{ void*       pBuf;                          // pointer to frames/ items
  long        hBuf;                          // handle of data buffer
  long        lItemsPerFrame;                // # of eFmt items per frame
  long        lNumFrames;                    // # of frames in this xfer
  EASIfmt     eFmt;                          // dataformat id
  float       xFmt;                          // dataformat nominal value
} EASIxfer;

extern const EASIvolume EASIvolume_0dB;      // nominal level/ 0dB
extern const EASIvolume EASIvolume_silence;  // no signal/ silence
extern const EASIvolume EASIvolume_illegal;  // illegal volume

extern const EASIiomask EASIiomask_all;      // mask w/ all channels active
extern const EASIiomask EASIiomask_none;     // mask w/ no channels active
```

```
//============================================================================
// * * *   t a r g e t   o s   s t u f f   * * * * * * * * * * * * * * * * * *
//============================================================================

#if TARGET_OS_WIN32
  // target is Win32
  #include "EASIwin32.hpp"
#elif TARGET_OS_MAC
  // target is MacOS
  #include "EASImacos.hpp"
#else
  // undefined target os
  #include "EASInoos.hpp"
#endif

//============================================================================
// * * *   E A S I p l u g i n   * * * * * * * * * * * * * * * * * * * * * *
//============================================================================

class EASIplugin: public EASIpluginBase
{ //=== transport ============================
  public:
    EASIapi(EASIerror)        // - error code
    Init                      // Open plug-in.
    ( void)                   // -
                                        =0;
    EASIapi(EASIerror)        // - error code
    SetPosition               // Set device position.
    ( EASIframes)             // - position [frames]
                                        =0;
    EASIapi(EASIerror)        // - error code
    BeforeStart               // Prepeare for audio streaming.
    ( EASIiomask,             // - channel mask playback
      EASIiomask)             // - channel mask recording
                                        =0;
    EASIapi(EASIerror)        // - error code
    GetReady                  // Prepare for zero-latency start-in.
    ( void)                   //  -
                                        =0;
    EASIapi(EASIerror)        // - error code
    Start                     // Start audio streaming.
    ( void)                   // -
                                        =0;
    EASIapi(EASIframes)       // - position [frames]
    GetPosition               // Query current device position.
    ( void)                   // -
                                        =0;
    EASIapi(EASIerror)        // - error code
    GetInternalError          // Query and reset streaming error.
    ( void)                   // -
                                        =0;
    EASIapi(EASIerror)        // - error code
    Stop                      // Stop audio streaming.
    ( void)                   // -
                                        =0;
    EASIapi(EASIerror)        // - error code
    AfterStop                 // Cleanup after audio streaming.
    ( void)                   // -
                                        =0;
    EASIapi(EASIerror)        // - error code
    Cleanup                   // Close plug-in.
    ( void)                   // -
                                        =0;
  //=== capabilities ===========================
  public:
    EASIapi(EASIindex)        // - number
    GetNumChannels            // Query number of physical i/os.
    ( EASIiomode)             // - i/o mode
                                        =0;
    EASIapi(const char*)      // - name
    GetChannelName            // Query name of physical i/o.
    ( EASIiomode,             // - i/o mode
      EASIindex)              // - physical i/o
                                        =0;
    EASIapi(EASIindex)        // - number
    GetNumTracks              // Query number of virtual tracks.
    ( void)                   // -
                                        =0;
    EASIapi(EASIindex)        // - number
    GetNumSampleRates         // Query number of supported samplerates.
    ( void)                   // -
                                        =0;
    EASIapi(EASIsrate)        // - samplerate [Hz]
    GetSampleRate             // Query nominal samplerate from index.
    ( EASIindex)              // - samplerate [index]
                                        =0;
```

```
    EASIapi(bool)             // - true, if pitchable
    GetPitchRange             // Query pitch range from indexed samplerate.
    ( EASIindex,              // - samplerate [index]
      EASIsrate*,             // - min deviation [Hz]
      EASIsrate*)             // - max deviation [Hz]
                                          =0;
    EASIapi(bool)             // - true, if const
    IsConst                   // Check for constant xfer field.
    ( EASIiomode,             // - i/o mode
      EASIindex)              // - offsetof xfer field
                                          =0;
    EASIapi(const EASIxfer*)  // - xfer
    GetStdXfer                // Query constant xfer descriptor fields.
    ( EASIiomode)             // - i/o mode
                                          =0;
    EASIapi(EASIindex)        // - EASI version
    GetVersion                // Query version number.
    ( void)                   // -
                                          ;
  //=== streaming ============================
  public:
    EASIapi(bool)             // - true, if sync'ed
    IsSynchronous             // Check for synchronous processing.
    ( EASIhandle)             // - callback handle
                                          =0;
    EASIapi(EASIframes)       // - position [frames]
    GetPosition               // Query streaming position.
    ( EASIhandle,             // - callback handle
      EASIiomode)             // - i/o mode
                                          =0;
    EASIapi(void)             // -
    NotifyDone                // Async completion notification.
    ( EASIhandle,             // - callback handle
      EASIiomode)             // - i/o mode
                                          =0;
    EASIapi(const EASIxfer*)  // - xfer
    GetXfer                   // Query xfer descriptor.
    ( EASIhandle,             // - callback handle
      EASIiomode,             // - i/o mode
      EASIindex)              // - physical i/o or virtual track
                                          =0;
    EASIapi(EASIframes)       // - length [frames]
    GetXferLength             // Query common xfer length.
    ( EASIhandle)             // - callback handle
                                          =0;
  //=== control ==============================
  public:
    EASIapi(EASIerror)        // - error code
    SetSampleRate             // Assign new samplerate.
    ( EASIindex)              // - samplerate [index]
                                          =0;
    EASIapi(EASIerror)        // - error code
    SetPitch                  // Assign new frequency deviation.
    ( EASIsrate)              // - deviation [Hz]
                                          =0;
    EASIapi(EASIvolume)       // - level
    GetLevelmeter             // Query current input level.
    ( EASIiomode,             // - i/o mode
      EASIindex)              // - physical input or virtual track
                                          =0;
    EASIapi(EASIerror)        // - error code
    SetRouting                // Set output routing.
    ( EASIiomode,             // - i/o mode
      EASIindex,              // - physical input or virtual track
      EASIindex,              // - physical output: left
      EASIindex)              // - physical output: right
                                          =0;
    EASIapi(EASIerror)        // - error code
    SetGain                   // Set mixer gains.
    ( EASIiomode,             // - i/o mode
      EASIindex,              // - physical input or virtual track
      EASIvolume,             // - mixer gain: left
      EASIvolume)             // - mixer gain: right
                                          =0;
    EASIapi(bool)             // - true, to request re-init
    SetupDialog               // Open setup dialog box
    ( const EASIdlgInfo*)     // - additional dialog box info
                                          =0;
  //=== plugin creation ======================
  protected:
    EASIplugin                // Constructor.
    ( EASIhost*)              // - EASI host
                                          ;
};
```

```
//============================================================================
// * * *   E A S I h o s t   * * * * * * * * * * * * * * * * * * * * * * * *
//============================================================================

class EASIhost: public EASIhostBase
{ public:
    EASIapi(EASIerror)        // - error code
    Streaming                 // Data streaming callback.
    ( EASIhandle)             // - callback handle
                                        =0;
    EASIapi(EASIerror)        // - error code
    Monitoring                // Monitoring/ metering callback.
    ( EASIhandle)             // - callback handle
                                        =0;
    EASIapi(EASIindex)        // - EASI version
    GetVersion                // Query version number.
    ( void)                   // -
                                        ;
  //=== host creation ==========================
  protected:
    EASIhost                  // Constructor.
    ( const EASIctorInfo*,    // - [in]  construction info
      EASIplugin**)           // - [out] EASI plugin
                                        ;
};

#endif

////////1////////2////////3////////4////////5////////6////////7////////8
// end of file
```