AAF Codecs

Null codec implementation and (re)use.

AAF Engineering Committee Meeting,
Atlanta, Georgia,
October 7, 2002

Jim Trainor, AAF Association

Extending AAF - Overview

- From the “AAF SDK Developers’ Guide V1.0”, extensions include:
  - Add new properties to built-in classes.
  - Define new classes.
  - Define new kinds of effects operations.
  - Define new kinds of audio or video compression.
  - Define new kinds of essence.
  - Define new container mechanisms to store essence data.

Extending AAF - Overview

- Add new properties to built-in classes:
  - Objects are simply a set of properties.
  - A property is a value with a defined name and type. Example:

Property Name	Property Type	Value
LastModified	TimeStamp	Fri Nov 2 14:47:35 2001

  - Easy to create, read, write, modify. Only real issue is documentation so other users know about your new property.

Extending AAF - Overview

- Define new classes:
  - Add a new ClassDefinition to the dictionary.
  - A ClassDefinition object defines the class' property set, and parent class.
  - Any application that discovers this new class should be able to read/write/modify the values and understand inheritance relationships.
  - Only real issue is documentation.

Extending AAF - Overview

- Define new kinds of effects operations:
  - Add an OperationDefinition object to the dictionary.
  - This describes the operations.
    - A category (a unique identifier, e.g. kAAFEffectMonoAudioDissolve).
    - Number of inputs, outputs, etc.
  - Plugins can be described, or identified, to perform all or some of the effects processing.
    - e.g. Interpolations plugins can be created to compute parameter values given key samples.

Extending AAF - Overview

- Define new kinds of effects operations, continued:
  - Plugins may or may not be required to process effects.
  - The SDK does not locate, load, or execute effect plugins.
  - Enough information is provided by the OperationDefinition for an application to locate, load, and execute any code that may be required to process the effect.

Extending AAF - Overview

- Define new kinds audio or video compression, and new kinds of essence:
  - A new audio or video compression format.
  - A new data type, e.g. motion capture data used to drive 3D animation.
  - GPS data (Global Positioning System), to accompany a video stream.
  - Auxiliary data feeds extracted from a video signal.
  - Next hot thing, etc, etc, etc

Extending AAF - Overview

- Define new kinds audio or video compression, and new kinds of essence, continued:
  - The AAF SKD (i.e. the COM library) does not directly support any form of essence IO.
  - **Essence codecs** are required for all essence IO operations.
  - The AAF SDK **does** automate the process of locating, loading, and executing, essence codecs.
  - **The null codec is an essence codec.**

Extending AAF - Overview

- Define new container mechanisms to store essence data.
  - Essence data processing is distinct from storage device IO.
  - Storage IO is implemented by separate interfaces that must also be provided by plugins.
  - These are simple read/write/seek interfaces.

Application View of Codecs

- An AAF application must select a codec when creating new essence material.
- New essence in an AAF file is normally created using the IAAFMasterMob::CreateEssence() function.
- Codecs are identified by a unique identifier.

Application View of Codecs

- A code fragment might look like this:

```
IAAFEssenceAccess* essenceAccess;
masterMob->CreateEssence(1,                    // Slot ID
                         soundDef,             // MediaKind
This ID selects the codec ➔ kAAFCodecWAVE,     // codecID
                         editRate,
                         sampleRate,
                         kAAFCompressionDisable,
                         NULL,                 // Essence locator
                         ContainerAAF,
                         &essenceAccess );

// Now use the essenceAccess interface to write essence data.
```

Application View of Codecs

- The AAF SDK has four sample codecs:
  - Wave audio
  - AIFC audio
  - JPEG compressed video
  - CDCI uncompressed video
- A sample MPEG codec is also available (thanks to the BBC), but not currently part of the SDK.

Application View of Codecs

- Example programs that read/write essence using codecs:
  - AAF/example/com-api/ImportAudioExample.cpp
  - AAF/example/com-api/ExportAudioExample.cpp
  - AAF/examples/axExample/axEssenceCreate.cpp
- See the Washington Tutorial “Essence Create” presentation and associated sample code.

Application View of Codecs

- CreateEssence() creates quite a bit of “scaffolding” in the AAF file.
- Among the objects created and added to the file, CreateEssence() adds a CodecDefinition object to the Dictionary.
- When the essence is opened for reading, this CodecDefinition is consulted to determine the codec id.

Application View of Codecs

- The application's responsibilities include:
  - Loading the codec using IAAFPluginManager.
  - Determining the codec's uuid for use in the CreateEssence() call.

Application View of Codecs

- IAAFPluginManager interface:
  - RegisterSharedPlugins() currently is hard coded to load the sample plugin libraries:
    - AAFPGAPI.dll (or .so)
    - AAFINTP.dll (or .so)
  - RegisterPluginFile() to load a single library.
  - RegisterPluginDirectory() to load all libraries in a directory.

Plugin Library Structure

- Plugins are dynamically loadable libraries.
- These libraries have:
  - Platform dependent entry points that are used to initialize, and otherwise query or control the library.
    - Windows: DllMain, etc
    - MacOS: DllInitializationRoutine, etc
    - Unix: ???, (static globals are constructed)

Plugin Library Structure

– Entry points defined by the AAF SDK:
  - AAFGetClassCount
  - AAFGetClassObjectId
  - DllGetClassObject
  - DllCanUnloadNow

– Look in ImplAAFPluginFile.cpp to see the code that checks for the presence of these symbols.

Plugin Library Structure

- Note, the following two functions are also standard Windows library entry points:
  - DllGetClassObject
  - DllCanUnloadNow
- They are also used by the SDK hence are required on all platforms.

Plugin Library Structure

- The following functions are used by the SDK to iterate over the list of class ids supported by the plugin:

ULONG AAFGetClassCount()

- Returns the number of COM objects implemented in the library.

ULONG AAFGetClassObjectID(ULONG index, CLSID *pClassID)

- Returns the i’th class id.

Plugin Library Structure

HRESULT DllGetClassObject(CLSID& clsid, IID& riid, void** ppv)

- Acts as a factory interface for the library.
- It creates an instance of the class identified by “clsid”, and uses “ppv” to return a pointer to the COM interface identified by “riid”.

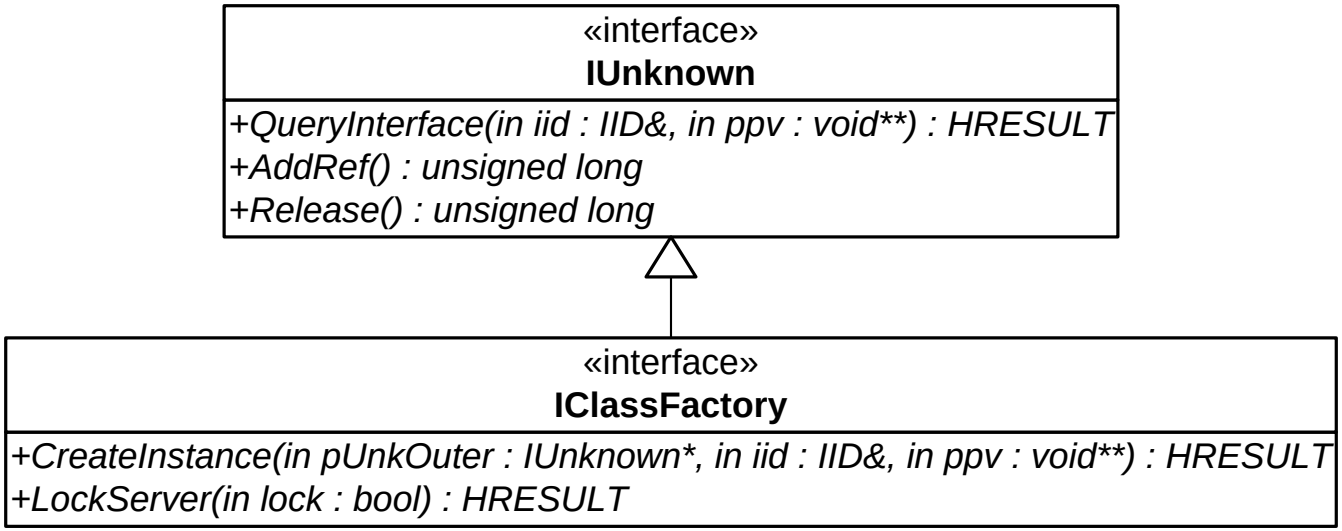
Plugin Library Structure

ULONG DllCanUnloadNow()

- Returns true if it is safe to unload the library.
- How does it know?
  - If any COM objects created by DllGetClassObject() have non zero reference counts, then it is not safe to unload the library - there are pointers floating around to the library's code.
  - It is even easier: increment an instance count in all your library's COM object constructors, decrement the instance count in the destructor. If the count is zero, it is safe to unload.

Plugin COM Interfaces

- IClassFactory
  - DllGetClassObject does not create the COM plugin interface directly.
  - It creates an IClassFactory object that in turn is responsible for creating the underlying COM object.



Plugin COM Interfaces

- The IClassFactory implementation creates an object that must implement:
  - IAAFPlugin
- And at least one of the other COM interfaces defined in AAFPlugin.h:

IAAFClassExtension
IAAFEssenceStream
IAAFEssenceDataStream

IAAFEssenceCodec
IAAFMultiEssenceCodec
IAAFEssenceContainer
IAAFInterpolator

Plugin COM Interfaces

- The null codec currently implements:
  - IAAFPlugin
  - IAAFEssenceCodec

COM Object Creation Sequence

1. The SDK loads the library and calls AAFGetClassCount()
2. For index = 0 to count, the SDK calls AAFGetClassObjectID()
3. For each class id the SDK calls: DllGetClassObject(), which returns an object that implements IClassFactory.
4. The SDK calls IClassFactory::CreateInstance() to create the underlying COM object.
5. The SDK makes repeated calls to QueryInterface() to determine what plugin interfaces the object supports.

COM Object Aggregation

- An important detail to be aware is the requirement to support COM object aggregation.
- Note the first argument of the IClassFactory CreateInstance method:

CreateInstance(IUnknown* pUnkOuter, IID& iid, void** ppv)

COM Object Aggregation

- If pUnkOuter is not null, then the COM object (e.g. the IAAFEssenceCodec implementation) is being aggregated by the object calling the CreateInstance() method.
- This is an important detail!
- The plugin's IUnknown implementation **must** be "aggregation aware".

COM Object Aggregation

- The null codec implementation supports aggregation using the technique described in:

  Inside COM, Dale Rogerson, Microsoft Press

- Note, this is different than the technique use by the original sample codecs supplied with the SDK.

Null Codec Basic Requirements

- Code that is easy to understand.
- Stand alone. No dependencies, other than on header files, on existing SDK code.
- Support multiple codecs in a single library.
- Easy to reuse:
  - Few, if any, modifications required to “bring up” a new plugin library.
  - Add code, don’t modify code.
  - Reusable IClassFactory and IUnkown implementations with support for aggregation.

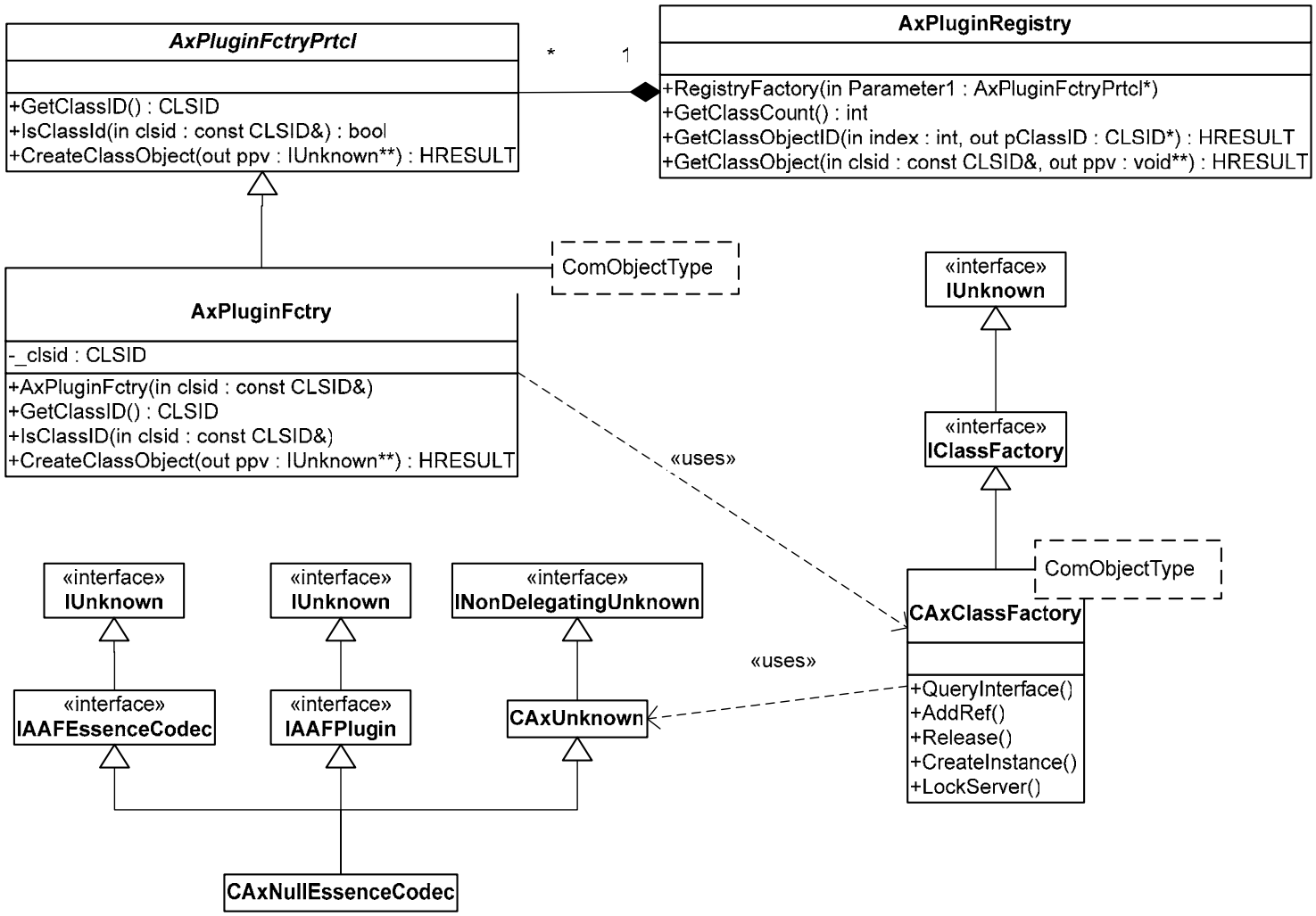
Null Codec Design

- A registry object to store a set of factory objects that will create the IClassFactory COM objects.
  - This registry is initialized at library init time.
  - Only one registry per library is required – use a singleton.
- IClassFactory implementation based on that found in the “Inside COM” book. This implementation supports aggregation.
  - The sample code in “Inside COM” varies only by the type of the underlying COM object – this can be cleanly implemented using a template.

Null Codec Design

- Shared IUnknown implementation based on that found in “Inside COM” (support aggregation.)
- DllCanUnloaded() now be implemented using class instance counters implemented using shared base class static counters.
  - Easy to support, all the COM objects will already share an IUnknown implementation.

Null Codec UML



Null Codec Reuse

- Declare a new class to implement your COM object:

```
#include "CAxUnknown.h"

class CMyCodec
: public IAAFEssenceCodec,
  public IAAFPlugin,
  public CAxUnknown
{
public:
        CAXUNKNOWN_DECLARE_IUNKNOWN_METHODS

        // Override CAxUnknown::NondelegatingQueryInterface() in order to added
        // support for the interfaces supported by this class.
        STDMETHOD( NondelegatingQueryInterface(const IID& iid, void** ppv) );
```

Null Codec Reuse

- Initialize CAxUnknown in your constructor:

```
CMyCodec::CMyCodec( IUnknown* pUnkOuter )
        :  CAxUnknown( pUnkOuter )
{
}
```

Null Codec Reuse

- Add your “NondelegatingQueryInterface” implementation:

```
HRESULT CMyCodec::NondelegatingQueryInterface(const IID& iid, void** ppv)
{
        if ( IID_IAAFPlugin == iid ) {
                *ppv = static_cast<IAAFPlugin*>(this);
                AddRef();
                return S_OK;
        }
        // Add tests for other IIDs you support.
        else {
                // If you don’t support it, delegate to CAxUnkown
                return CAxUnknown::NondelegatingQueryInterface( iid, ppv );
        }
}
```

Null Codec Reuse

- Register you new class:

```
// Simple class with constructor that creates a new AxPluginFctry<> and registers it under
// you class id.
class MyRegistration {
   MyRegistration
   {
     std::auto_ptr<AxPluginFctryPrtcl>
          myCodecFctry( new AxPluginFctry<CMyCodec>(
                         CLSID_CAxNullEssenceCodec ) );

     AxPluginRegistry::GetInstance().RegisterFactory( myCodecFctry );
   }
}

// Declare a static global – its constructor will register you COM class when the library is
// loaded.
MyRegistration myRegistration;
```

Null Codec Reuse

- It is not necessary to modify any of the null codec code to reuse it.
- Just follow the recipe to create a new .h and .cpp file for your new COM object, and compile.
- But wait…. This is just the beginning, the actual implementation of the COM object still must be added!


Advanced
Authoring
Format
AAF Association
Developer Support