DM Plug-in Design

Avid Technology

Revision 00.00.01

Author: Jeffrey Bedell

23-03-99

Contents

1. Overview

This document is the design specification for the handling of plug-ins within the Advanced Authoring Format (AAF) SDK. Included within this specification are:

- The mechanism for selecting a plug-in which is currently loaded
- An example of plug-ins by AAFEssenceAccess (AAFEssenceStream and AAFEssenceCodec)
- An example of plug-in interpolators
- An example of reaching information needed to handle a plug-in with an API other than those above

Code to call plug-ins with other API codec is NOT included in this design.

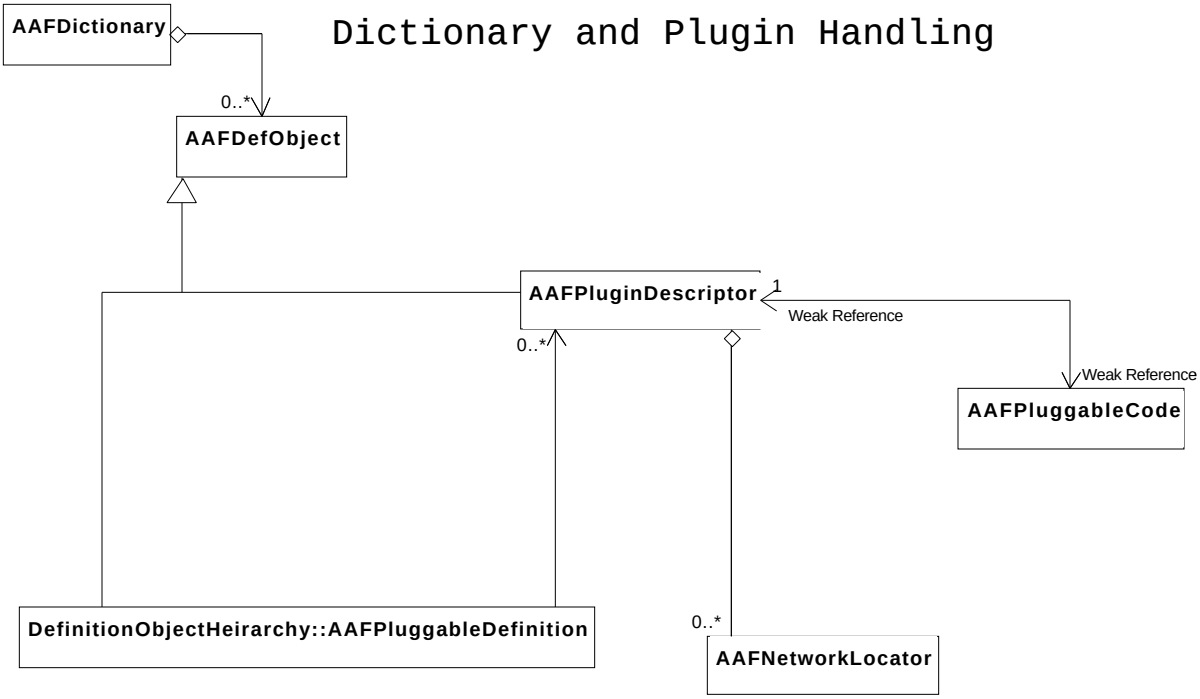
The specification is annotated with UML class diagrams and interaction diagrams. A quick reference for UML is available at:

`http://www.rational.com/uml/resources/quick/uml_poster.jtmpl`

2. Plug-in Design

2.1 Objects

The objects making up the plug-in subsystem come in both persistent and transient flavors. The AAFPluggableDefinition and the AAFPluginDescriptor both reside in the dictionary, and the AAFPluggableCode resides in the content storage.



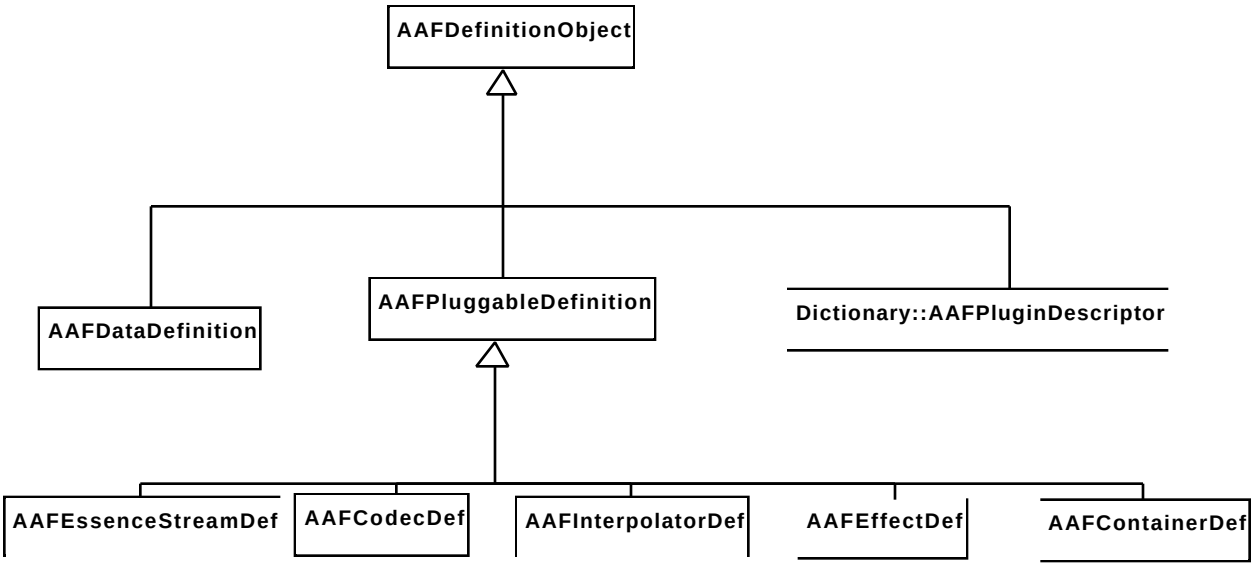
The AAFPluggableDefinition object describes the kind of plug-in, without referring to actual executable code. It has subclasses for AAFCodecDefinition, AAFInterpolatorDefinition, and so on. An AAFPluggableDefinition has weak references to one or more AAFPluginDescriptor, which tells how to get to executable code for one platform.

Having the AAFPluginDescriptors owned by the dictionary and using weak references from the AAFPluggableDefinition allows a single plug-in code module to contain more than one AAFPluggableDefinition (ex: A collection of AAFEffectDefinitions, all of which may be processed with a single piece of code).

The executable code may be present in the AAF file, encapsulated in an AAFPluggableCode object, which is attached to the content descriptor. The AAFPluginDescriptor may instead contain two arrays of locators, one of which is the actual download URL for the plug-in, and the other is the URL of the manufacturer of the plug-in. The array of download URLs is an ordered list, with the most available download sites first in the list. The list should be scanned once to see if one of the URLs is local to the receiving site, and then tried top to bottom.

Multiple AAFPluginDescriptors may exist for a given AAFPluggableDefinition, indicating other implementations of a given effect, AAFEssenceCodec, etc. The possible differences include platform, hardware accelerated, no hardware required, and cost (if any) of the plug-in.

The definition object class hierarchy now looks as follows:



2.2 Operations on plug-ins

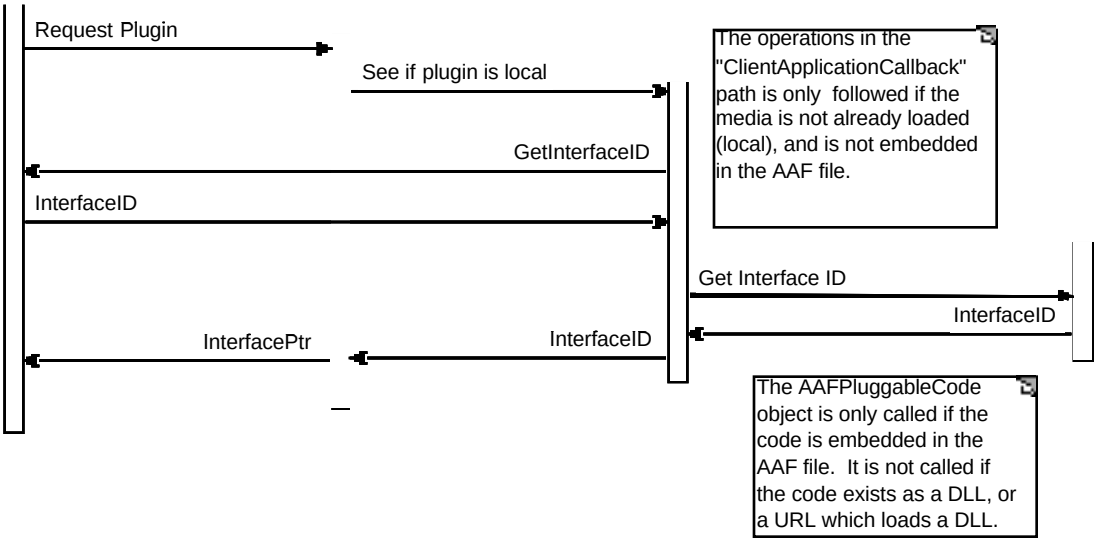
This section describes some common operations on plug-ins without reference to any particular type of plug-in. This is because only a few plug-in APIs have been defined, and these are covered in later sections).

2.2.1 Finding the executable code for a plug-in

One common operation is to locate the interface pointer for a plug-in, given some form of AAFPluggableDefinition (such as AAFInterpolatorDefinition). Given the assumption that the client application already has a pointer to an AAFPluggableDefinition subclass, there possibilities are:

1. The code is already loaded as a registered DLL
2. The code exists in a DLL on some remote system
3. The code exists inside of an AAFPluggableCode in the AAF file.

I am assuming that the binary data inside of an AAFPluggableCode is a complete DLL, which may be registered by the AAF SDK, and so there is not a fourth case of an AAFPluggableCode which is loaded.

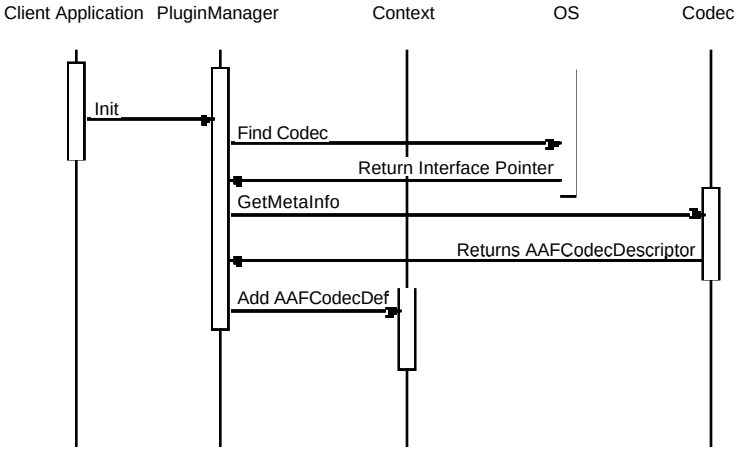


When searching for the best AAFPluginDescriptor, other criteria such as hardware accelerated or software-only can be specified.

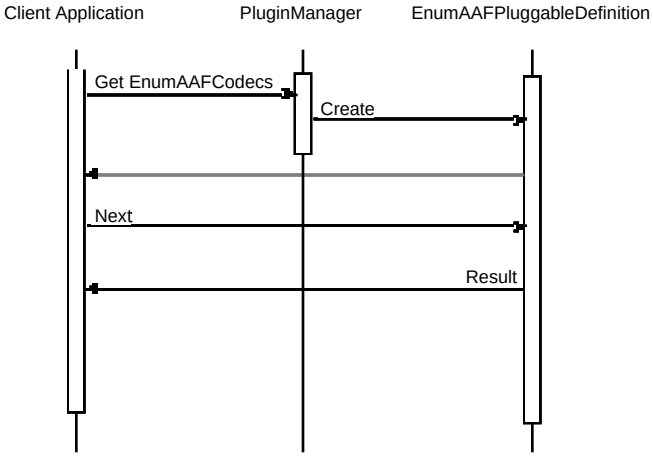
This design assumes the ability to call back across the interface between the application and the SDK, as only the application has the ability to manage the user interface which may be required in order to download the plug-in.

2.2.2 Loading the base plug-in set

The AAF SDK loads a certain set of basic plug-in definitions on startup. How the definitions are stored is <TBD>, but the actual code will exist in a single DLL for each type of plug-in.



2.2.3 Finding out what plug-ins are loaded

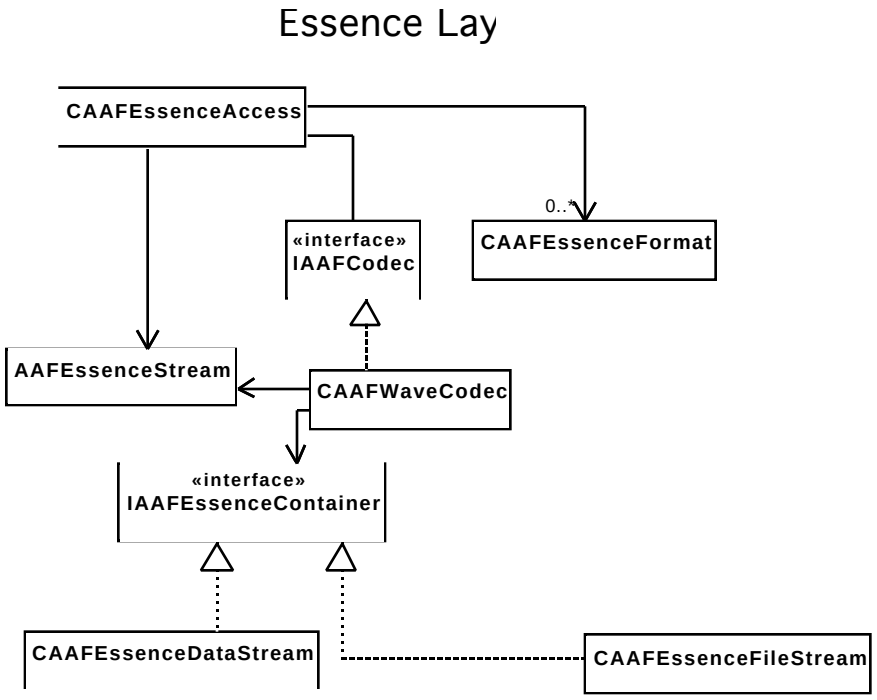


3. Essence Plug-in Design

In order to show an example of an AAF Plug-in type, the AAF SDK essence access class and support classes will be described, showing how plug-ins integrate. There are two plug-ins in this subsystem, codecs and essence containers.

3.1 Objects

The essence access layer of the AAF SDK is diagramed below. The class AAFEssenceFormat is not related to plug-ins, and will not be discussed further in this design document.



The classes covered here are:

- AAFEssenceAccess
  A helper class, which allows you to access essence as a stream, without having to consider formatting, decompression, and reformatting.

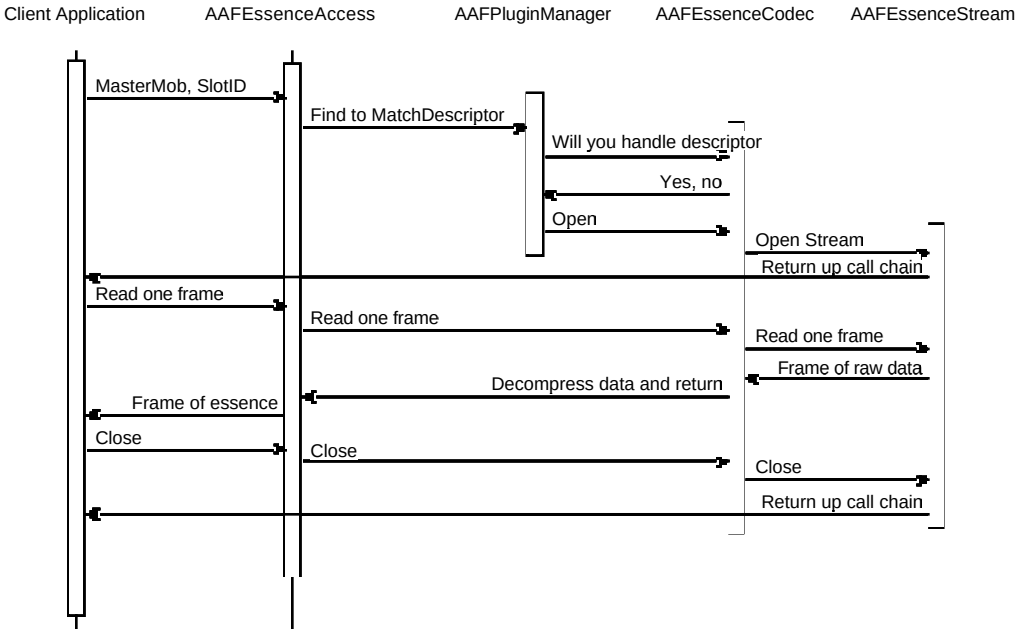
- Implementers of IAAFEssenceCodec (ex: CAAFWAVECodec)
  Classes which encapsulate the format of the bits within a file (ex: TIFF, WAVE, RGBA) and any compression used.

- AAFEssenceStream
  Class which restructures the data from byte streams into sample streams. Created by a collaboration of IAAFEssenceContainer and IAAFEssenceCodec.

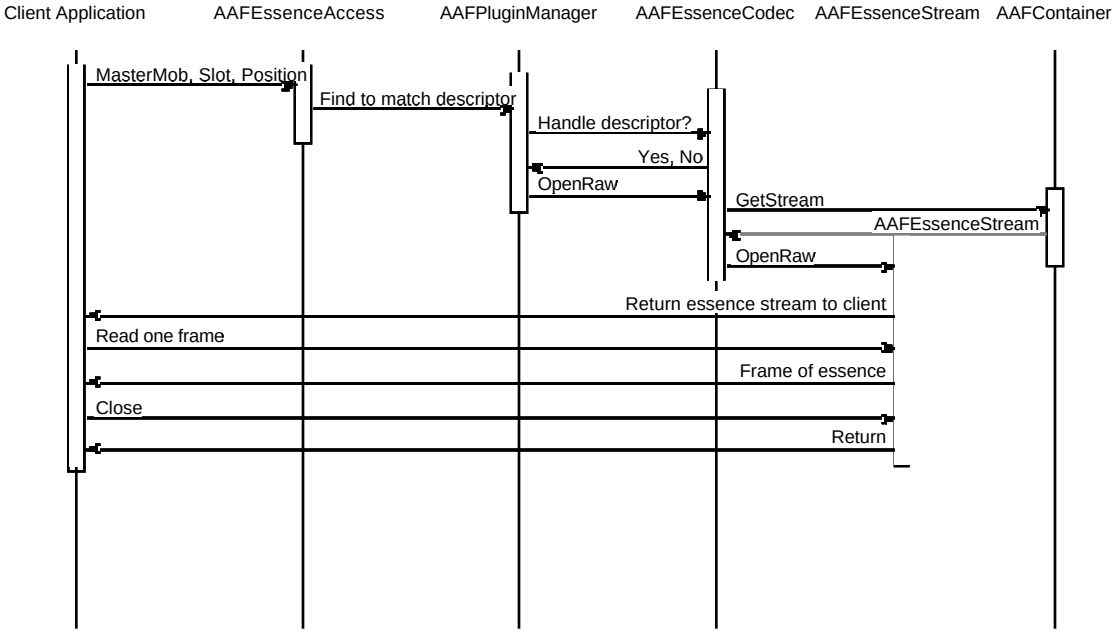
- Implementers of IAAFEssenceContainer
  Classes which encapsulate the form in which the bits produced by the codec are stored. Examples of streams include CAAFEssenceDataStream, in which the bits exist as a property within an AAF file, and CAAFEssenceFileStream, in which the bits exist in a file by themselves.

3.2 Typical Media Actions

3.2.1 Opening and reading a media stream



3.2.2 Opening and reading a raw media stream



3.2.3 Creating a media stream

4. Interpolator Plug-in Design

The design of interpolators is much simpler, as the plug-in code consists of a single class with no interaction outside of itself. The class AAFInterpolatorDef will have methods to locate the plug-in code, which implement IAAFInterpolatorPlugin. The classes which implement IAAFInterpolatorPlugin have a single method which interpolates between to given AAFControlPoint to a new location, given the algorithm used by the plugins. Some interpolators (ex: linear and step) should be able to handle a range of types on the control points, whereas others may require specific types (ex: pairs of coordinate points).

5. Related Documents

File Name	Location	Owner	Description
SMPTE Specification			My version is SMPTE P18.277.990122 (Dynamic)
OMFI 2.1 Spec		Avid	

6. Revision History

Name	Date	Version	Description
Jeffrey Bedell	2/16/1999	0.0.1	Initial Revision