
HELIX

April 2004

SO WHAT IS HELIX?

The core technology came from Real Networks and was open sourced in 2002. The Helix DNA platform provides “*the first open multi-format platform for digital media creation, delivery and playback*”, being:

- Producer
- Server
- Client

Producer:

Media encoding engine that creates:

- Streaming broadcasts
- On demand streaming content
- Downloadable Audio / Video files

Server:

Media delivery agent:

- Real time media delivery / congestion response

Client:

Media playback engine:

- Handles presentation of media streams (audio / video)
- Interacts with user to control presentations

LICENSING

RPSL: Real Networks Public Source License

- OSI certified
- Copy-left
- Open Source
- Developed application must be open sourced

RCSL: Real Networks Community Source License

- Commercial source licence
- Provisions for inclusion in proprietary applications

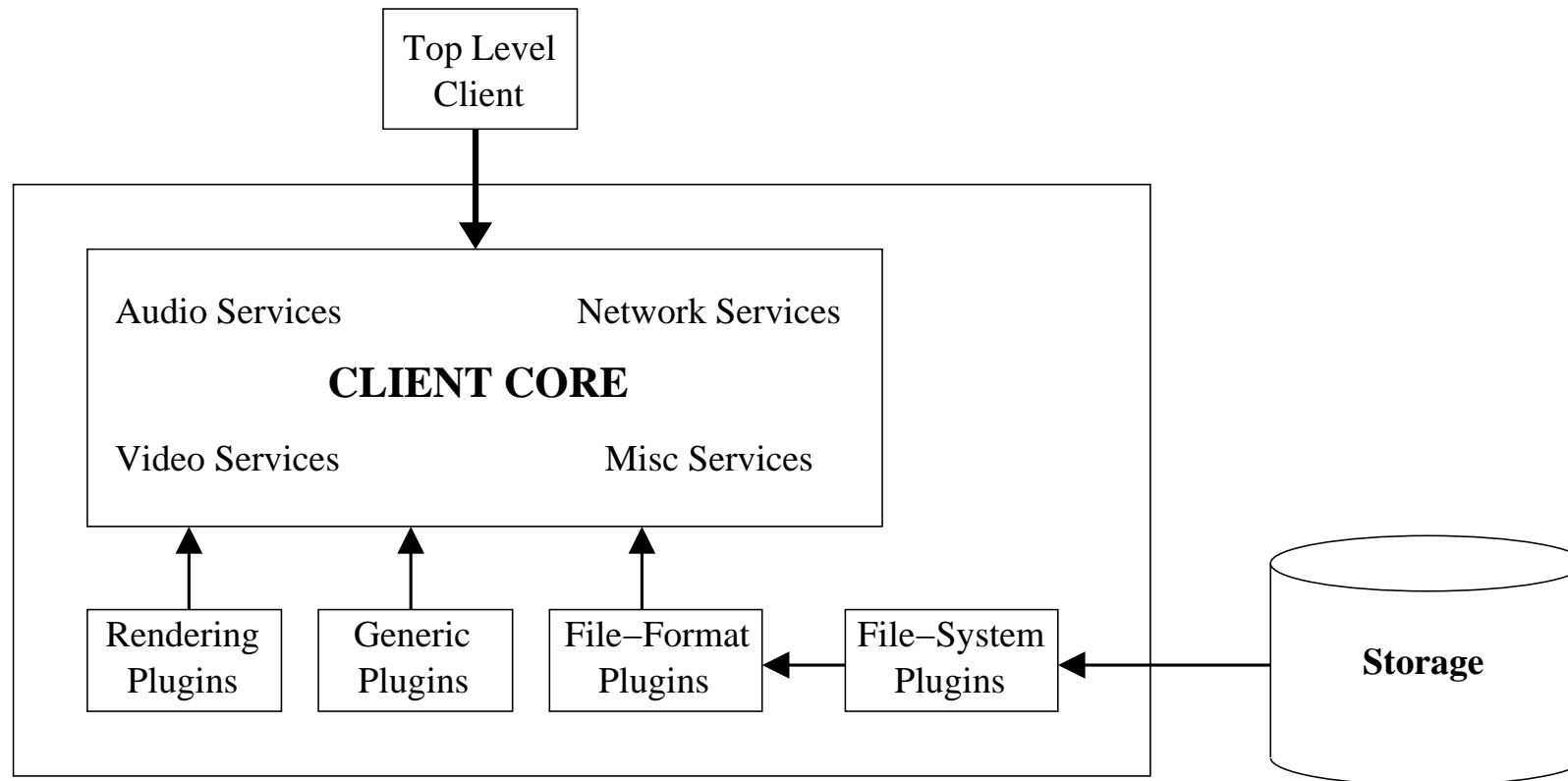
PRICING - CLIENT

R&D: No Fee (both RPSL & RCSL)

Commercial Use:

- Without RealAudio / RealVideo
 - First million free
 - 10c per unit over one million
- With RealAudio / RealVideo
 - 25c per unit (on windows PC's)
 - 25c per unit, up to maximum annual payment of \$1 million (all other platforms)

HELIX ARCHITECTURE - CLIENT



CORE SERVICES

All services aim to provide a platform independent interface for plug-ins to core functionality such as audio playback.

Audio:

- Playback synchronisation
- Mixing & re-sampling
- Cross-fading
- Volume control

Video:

- Window (site) management
- Alpha blending
- Access to key / mouse events

Networking:

- TCP / UDP functionality
- DNS lookups

Miscellaneous:

- Plug-in handler
- Cross platform thread abstraction
- Client registry

PLUG-INS

Plug-ins are used by the Producer, Server and Client.

File System:

Provides access to different data streams, such as HTTP, Local File, RTSP...

File Format:

Converts a physical bitstream of data into packet data for streaming. Stream management (eg: congestion notification, language). This step usually generates packets that are individual frames of whatever media is being streamed.

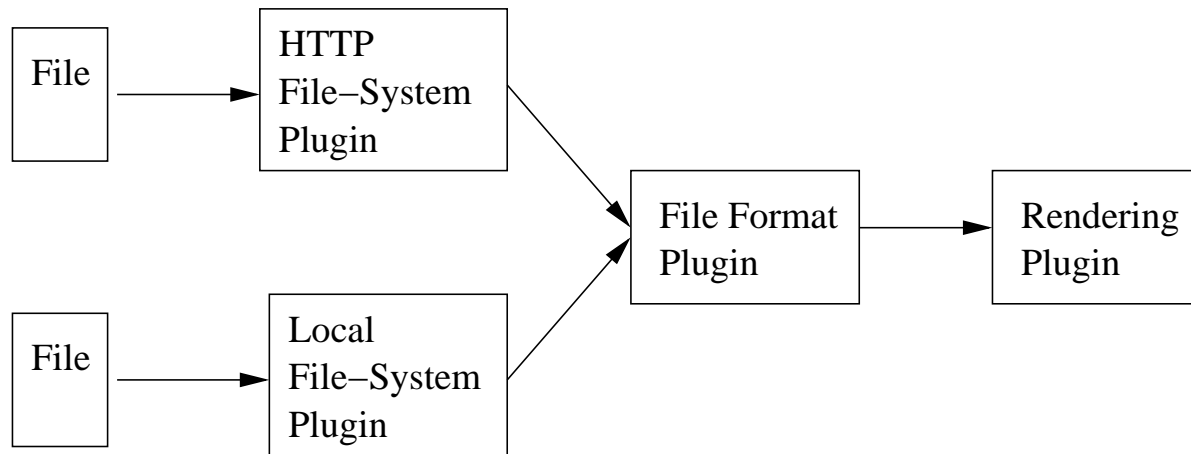
Packets are generally demuxed into logical streams from a single physical stream.

Rendering:

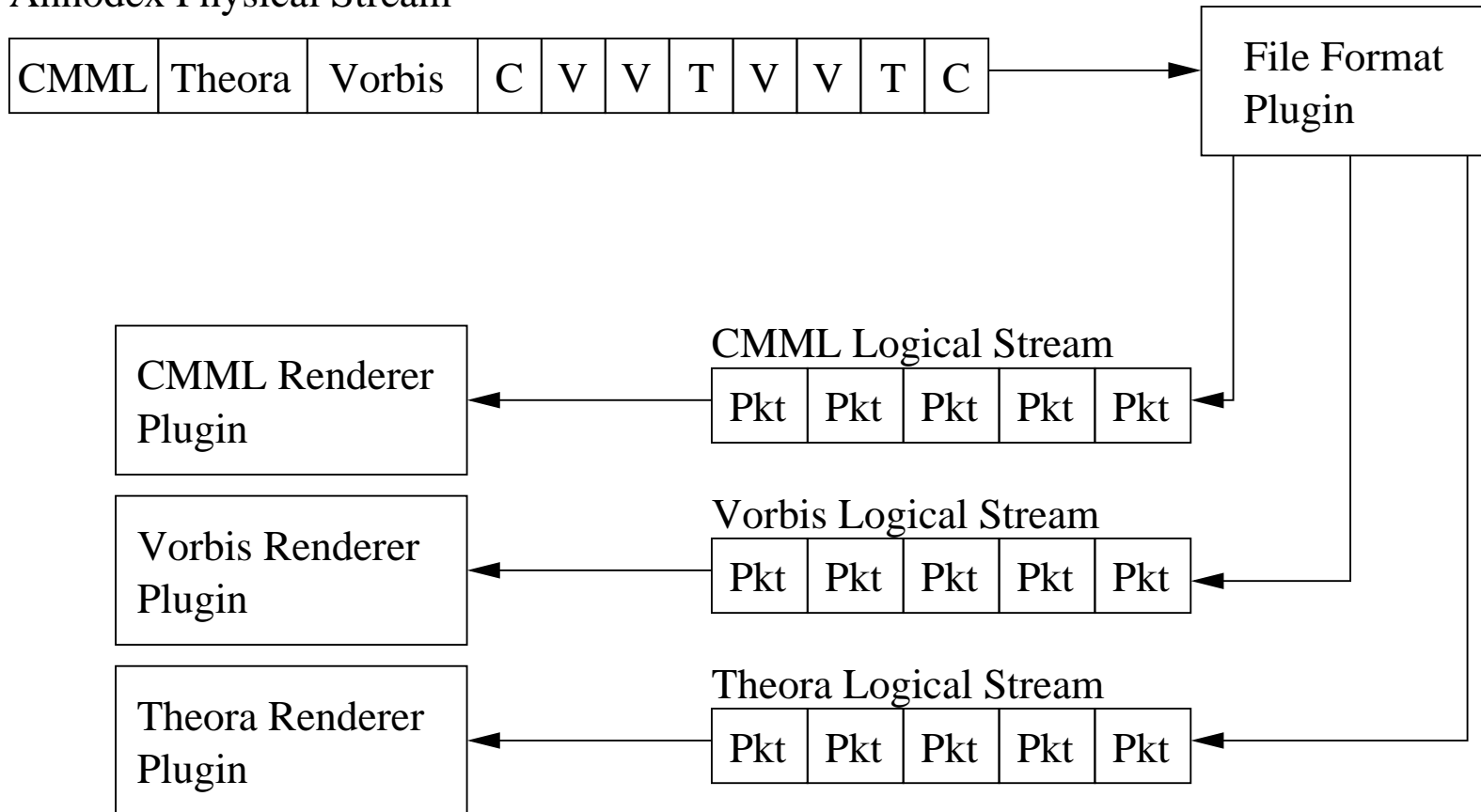
Prepares streamed packets for playback using the audio or video services.

Generic:

Any plug-in which doesn't fit into the above categories.
Examples given include Source Viewing & Audio Visualisation.



Annodex Physical Stream



TOP LEVEL CLIENT

The application that brings together all the functionality of the helix engine and allows user interaction. The TLC interacts with the helix engine using interfaces. The 3 major interfaces of interest when writing a TLC are:

IHXClientEngine: provides hooks into the helix engine enabling access to Create / Destroy *Player* objects as well as notifying the engine of an event.

IHXPlayer: controls presentation playback by opening / playing / pausing / stopping a media stream.

IHXSiteSupplier: is implemented by any TLC that is capable of displaying video presentations. Receives notifications from rendering plug-ins when a site is required to render to.

RMB PROJECT

We needed to work with:

- Video Service (evas support)
- File Format Plugins (ogg→anx, saving)
- Rendering Plugins (cmml, theora)
- Top Level Client

OBTAINING HELIX

Helix components are available in both binary and source formats. For both formats it is necessary to:

- Create an account on helixcommunity.org
- Agree to a licence prior to downloading

Binary Downloads:

Once a user account has been created and a licence agreed to, binaries from the [helixcommunity](http://helixcommunity.org) webpage are available to download.

Source Downloads:

- Create public/private key pair 'ssh-keygen'.
- Upload the public key to [helixcommunity](http://helixcommunity.org) user account.
- Download the build system...

THE BUILD SYSTEM

Components:

- Python Scripts (the build engine)
- Build Information Files (BIFs)
- .buildrc
- Profiles
- System Profiles
- Umakefile(s)

Python Scripts:

The Helix build system is a set of python scripts that parses all the other components of the build system (ie: profiles / umakefiles), to generate a series of platform specific Makefiles. The makefiles are then used to build a helix project.

Build Information Files:

An XML file defining intercomponent dependancies and libraries required to build components of the helix tree as well as third party programs using the helix tree.

```
<module id="datatype_image_jpg_common"
  name="datatype/image/jpg/common"
  group="core">

  <cvx root="helix"/>

  <source_dependlist>
    common_include
    common_util
    common_container
    datatype_common_util
    datatype_image_common
    datatype_image_jpg_import_jpeg-6b
  </source_dependlist>
</module>
```

.buildrc:

Defines the location of CVS repositories, BIF files and libraries required by components of the helix projects.

```
AddCVS("helix", "/home/fst/CVSRoot")
AddBIFPath("common", "[helix]/helix/common/build/BIF")
AddBIFPath("client", "[helix]/helix/client/build/BIF")
AddBIFPath("annodexclient", "[helix]/annodex/src/BIF")
SetSDKPath("oggvorbissdk", "/usr/local");
SetSDKPath("SYMBIAN70SSDK", "/usr/local/symbian");
```

Profiles:

A python script that specifies boolean #DEFINES that are used both during pre-build script parsing and compilation time.

Profiles exist within the helix build directory 'build/umakepf'.

...

```
project.AddDefines( 'HELIX_FEATURE_AUDIO' )
```

```
project.AddDefines( 'HELIX_FEATURE_CORECOMM' )
```

...

NOTE: There is minimal documentation of these helix defines located within the helix module in CVS at `ribosome/build/doc/features.html`, however no reference to dependancies of components to defines (when all else fails, turn them all on).

System Profiles:

Specifies a per-platform set of characteristics used by the build system to build helix, such as location of compiler / linker, platform specific libraries, endian-ness.

The system profile the build system will use is determined by the environment variable `$SYSTEM_ID`.

eg:

- linux-2.2-libc6-i586
- symbian-70s-armi
- win32-i386-vc7

Umakefile(s):

A platform-independent Makefile. Specifies everything about how to build the current subtree including source files to compile and libraries to link against. The Umakefile is a python script and thus can use conditional tests to (in|ex)clude build details depending on any previously defined data such as platform, helix defines... etc.

BUILD SYSTEM - ISSUES

- Rebuilding does NOT detect changes to header files.
- Takes a while to get used to.
- End up reverting to make files for speed.

BUILDING HELIX

- `$ export CVS_RSH=ssh`
- `$ cvs -d
:ext:username@cvs.helixcommunity.org:/cvsroot/ribosome
co build`
- `$ export BUILD_ROOT= ... /build`
- `$ export PATH=$PATH:$BUILD_ROOT/bin`
- `$ export SYSTEM_ID=linux-2.2-libc6-i386`
- create `~/.buildrc`
- `$ build`

What 'build' does:

Allows the selection of projects and profiles for building (as specified in the BIFs & profile directory).

Can be run as either an interactive menu or a command line utility.

- Checks out all required components from CVS
- Generates Makefiles from the Umakefiles
- `$ make all copy`

... SO IN CONCLUSION

Helix is the name applied to three piece media framework which while large and (sometimes) clumsy attempts to be as portable and extensible as possible.