APPSCIO™

Appscio(tm)
Media Processing Framework (MPF)
Installation Guide

Table of Contents

1. Document Information

This section contains information about this Appscio document.

1.1 Document History

Version	Date	Author	Changes
1.0	Feb 15, 2009	Appscio, Inc.	Add MPF documentation
1.1	May 9, 2009	Appscio, Inc.	Revised Installation Section
1.2	May 20, 2009	Appscio, Inc.	Added Source RPM install path
1.3	May 24, 2009	Appscio, Inc.	Added dependency diagrams
1.4	Jul 13, 2009	Appscio, Inc.	Added tar install instructions; other minor modifications

1.2 Document Conventions

The Appscio MPF Installation Guide will observe the following conventions:

- Informational text will be provided using Arial, 10 point
- Instructions for any user input, including commands will be given in ***bold, italicized*** form
- Output from any actions will be shown using `Courier,10 point`
- References to components and modules will be made using `Courier-New, 10 point`

1.3 Companion Documentation

This Appscio MPF Installation Guide documents the installation process. For help with getting started after installation, please reference the Appscio MPF Development Guide.

2. Overview

In this section, we explore the challenges associated with audio and video data (AV data) and the opportunities that exist to address the management of this data.

2.1 The Challenges of AV Data

Audio and video (AV) data is being generated and consumed at exponentially growing rates. Though the data is incredibly rich, it is also essentially unstructured and orders of magnitude larger than conventional structured data. It's unstructured nature combined with it's size makes multimedia data unwieldy to manage using conventional access control, search, policy enforcement, storage control and transmission software technologies.

Simple compression algorithms can eliminate redundancy and reduce the overall size of audio and video files. However, managing their **contents** requires unlocking the meaning and structure within these files (i.e. the **semantics**).

Semantics can be expressed as structured data describing "events" that occur at specific points in time within an AV file. Examples of events might include:

- motion beginning or ending in a video
- tracking when a person starts or stops speaking in an audio file
- recognizing the text of a speech in an audio track
- selective recognition of a person in a video or series of videos
- tracking suspicious behavior

Combinations of these semantic event descriptions define logical "**segments**" that correspond to meaningful activities (e.g. a meeting, transaction, Powerpoint slide being displayed, particular speaker talking, certain language being spoken, etc). These logical segments that can in turn be (indirectly) managed with conventional software. For example, access to the segment of surveillance video recorded in a conference room during a meeting can be restricted using existing identity management and access control software.

Temporal semantic data is available from two generic sources:

- Intelligent devices associated with the multimedia file (e.g. RFID tag readers in the vicinity of video conference equipment, POS systems within the field of view of security cameras, etc.)

- From the media files themselves (e.g. algorithms that detect, classify, track, or recognize objects within the AV channels.)

Well understood, standard mechanisms exist for capturing, integrating and consuming the structured data generated by intelligent devices.

2.2 Integrating Audio/Video Algorithms

The availability of professional and consumer digital AV devices (e.g. camcorders, audio recorders) several decades ago triggered research resulting in algorithms capable of harvesting structured data from the files these devices generate. The early efforts were focused on targeting weapons and supporting surveillance.

To that end, agencies of the U.S. government invested millions and millions of dollars in machine vision and acoustic research through programs, such as Video Analysis and Content Extraction (VACE) and Cognitive Assistant that Learns and Organizes (CALO).

These research projects produced many powerful, specific algorithms capable of recognizing objects, people, behaviors and words in digital media. Funding continues to advance the state-of-the-art (albeit more slowly) with new approaches and faster hardware.

However, these research efforts were typically funded with a narrow focus on either audio or video algorithms to solve one specific problem. Each was funded with different requirements for media formats, evaluation criteria, test frameworks, etc. To date, no technology has been developed to serve as a standard development and deployment environment for new algorithmic work.

The result is that each time researchers begin work on a new problem, they first must build a supporting environment for their algorithms. Worse yet, if and when newly developed algorithms are commercialized – many remain proofs-of-concept or published only in academic papers – they're typically embedded in vertical applications and/or custom hardware.

These "stove-pipe" systems are rigid, proprietary deliverables that are extremely expensive to extend and upgrade as new and better algorithms emerge.

In addition, the absence of a standard "media algorithms" software framework causes other problems:

- **Re-use**: Building upon algorithms developed by others is difficult because they're typically packaged within different, proprietary environments which encourages researchers to re-invent rather than re-use.
- **Multi-modal recognition**: Developing algorithms that utilize metadata from all audio and video channels within AV files requires an environment supporting the communication of metadata across and between audio and video algorithms; none exists today.
- **Evaluation**: Comparing algorithms and combinations of algorithms (e.g. combining signal conditioners with object detectors) becomes a significant engineering project when the algorithms were developed for different frameworks.
- **Distribution/deployment**: The lack of a standard framework inhibits the rapid deployment of new algorithms.
- **Integration**: Integrating "best-of-breed" algorithms from independent researchers requires porting, repackaging and re-factoring – a significant development effort on its own.

2.3 Why is Integration Important?

Progress in the development of AV algorithms has been widespread. Thousands of developers in hundreds of academic institutions, research institutes and commercial organizations worldwide continue to invent new algorithms. But, these "*centers of excellence*" typically focus on difficult problems with limited scope.

Exploiting the full power of these algorithms requires combining them in a variety of ways. Within an audio and/or video channel, obtaining optimal results requires integrating signal conditioners, signal transforms, primitive object detectors and trackers, object recognizers, metadata analysis and synthesis algorithms and, potentially, CODECs.

For example, virtually all speech-to-text applications combine a variety of algorithms to produce their results. First, a variety of decoders are deployed to decompress the data in the target audio channel. Typically, the decompressed audio is then filtered and enhanced to reduce noise and isolate the part of the signal most likely to correspond to human speech.

Increasingly standard algorithms then harvest numeric "features" from segments of the audio signal. Phoneme recognizers then generate candidates which are then analyzed and combined by word recognition algorithms. Word candidates are then processed through algorithms that apply language models to select appropriate words and apply punctuation. Finally, text analytics perform such functions as topic recognition.

Similar combinations are employed in the video domain to recognize text, faces, license plates and so on. Future improvements will increasingly combine cross-domain (multi-modal) algorithms to produce even higher quality results. For example, recognizing the identity of a particular speaker from the text superimposed on broadcast video should improve the quality of speech recognition.

2.4 Standard Metadata Vocabularies

Integrating elementary algorithms requires that they communicate. Effective communications between independently developed algorithms requires standards. Though there have been attempts to define such standards, to date no standard metadata formats and vocabularies have emerged. This presents another significant obstacle to exploiting the capabilities of the full spectrum of AV algorithms.

Specifically, many of the important algorithms are, at their core, pattern recognizers. These algorithms detect "events" in a channel within an AV stream and express confidence levels associated with the recognition of those events. For example, a "face detector" recognizes objects in a video channel that it believes (with a certain degree of confidence) are human faces.

Typically, such an algorithm will report its results as "regions of interest" within a frame containing faces and associated numeric confidence levels. If this face detector's output is to be used as input to a face recognizer from another source, they must share a common definition of a "region of interest" and how "confidence" is measured.

A standard format for representing metadata generated by these algorithms must emerge as well as standard vocabularies for describing "events" (e.g. a "region of interest") and agreement on how to express confidence in the results.

3. Introducing the Appscio Media Processing Framework (MPF)

Appscio is developing a new class of software to address the issues involved in capturing, integrating, synchronizing, evaluating and consuming semantic data generated by media algorithms.

Appscio approached the challenges associated with audio and video data (AV data) with the following set of key objectives:

- Promote rapid, widespread adoption
- Leverage existing technologies
- Simplify the development and deployment of media algorithms
- Allow portability across the spectrum from mobile devices to server farms
- Maximize the scalability of solutions
- Encourage the emergence of standards

The Appscio Media Processing Framework (MPF) is an open software framework that is independent of media formats and data types. To promote adoption it is distributed as open source software. To leverage existing work it is implemented as an extension to one of the most mature and respected open source projects – gstreamer (http://www.gstreamer.net) – and it employs the W3C (www.w3.org) standard for expressing metadata – Resource Description Framework (RDF). More information on RDF can be found at: (www.w3.org/RDF).

3.1 Understanding Basic MPF Terminology

At its heart, the Appscio MPF is a tool for configuring and executing "pipelines" composed of semantic media algorithms. A **Pipeline** is a sequence of **Algorithms** (wrapped with Appscio MPF code) configured to perform an application-specific **Service**. The services implemented in Pipelines can be as simple as transcoding a media file from one format to another or as complex as generating a transcript from the audio track, or recognizing people in the video stream.

Algorithms perform a wide variety of elementary functions and fall into distinct categories. **CODECs** compress (encode) and decompress (decode) audio and video (AV) signals. **Filters** modify frames in AV signals. **Detectors** identify objects (e.g. words, faces, text) and behaviors (e.g. motion) in AV channels. **Classifiers** and **Recognizers** associate structured fragments of metadata with objects and behaviors. **Analytics** distill, improve, combine and/or summarize fragments of metadata.

Appscio MPF provides a mechanism for communicating semantic metadata between Algorithms. Metadata is "published" and consumed through **Pads** implemented in the Appscio MPF software **wrappers** around elementary media algorithms. Algorithms are assembled in Pipelines and metadata flows "downstream" through Pads forming a "metadata channel."

3.2 About Appscio MPF and gstreamer

Appscio MPF extends gstreamer – an open source media handling toolkit. Since its inception, the gstreamer community has created large libraries of elements – algorithms typically developed for media playback applications. Most of the elements in the gstreamer libraries are CODECs and Filters.

Appscio MPF wrappers are designed to hide the complexity of the gstreamer environment and to add support for transporting metadata. Semantic algorithms can be turned into gstreamer elements by wrapping them with Appscio MPF code. Pipelines are defined by specifying combinations of native gstreamer and wrapped Appscio MPF elements in a particular order. Which elements are chosen and the order in which they are executed depends on the application.

3.3 Appscio MPF Architecture Overview

The core of the Appscio MPF architecture is the Analytics Pipeline, which is illustrated in *Figure 1*. The Analytics Pipeline supports moving media streams (e.g. audio, video) through a network of elements and components.

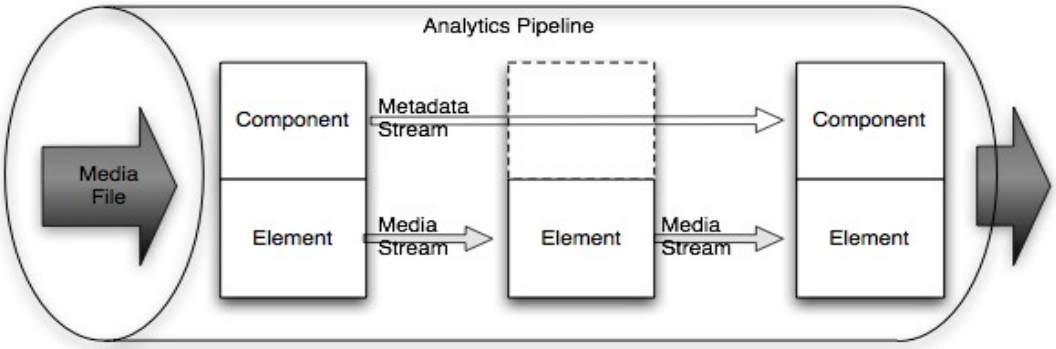


Figure 1 The Analytics Pipelines Uses Elements and Components

The Analytics Pipeline is built on top of gstreamer's pipeline and its associated packages of elements. The elements from gstreamer are as follows:

- **stream managers** (e.g. sources and sinks for HTTP or RTP, queues, multiplexers, or demultiplexers)
- **stream handlers** (e.g. codecs for getting streams in and out of various formats, or transformers to adjust color spaces, audio levels, video brightness, etc.)

Existing algorithms or media-processing software elements are wrapped using Appscio tools to become a component – a manageable, reusable code block with efficient connections into the metadata communication and management layer. The component wrapper layer depends on the software environment or on proprietary libraries to provide the transformative or analytic code.

Elements are low-level, while the components in the Analytics Pipeline recognize or react to higher-level objects (e.g. faces, actions, words). The components generate metadata and pass that metadata into the metadata stream while staying synchronized with the media stream. The metadata stream can either be exported to applications, or used by downstream components in recognizing yet-higher level events.

The low-level elements can be quickly intermixed with the components to meet specific data and format needs of the various components. The Pipeline manages the configuration negotiation to ensure each component gets input data of the type it specifies.

The primary architectural goals of Appscio MPF are:

1. pre-package repetitive code needed for creating media-centric applications
2. provide a pluggable framework that allows multiple vendor analytics modules to be joined together as application building blocks
3. gain cost effectiveness and security/quality improvements by adopting and participating in open source and open development practices
4. provide for future evolution in AV metadata availability and languages

Appscio MPF takes media input from files and produces both transformed media files and descriptive metadata.

4. Appscio MPF Installation

The instructions in this section should enable you to quickly begin working with Appscio MPF.

4.1 Hardware Requirements

Appscio MPF leverages commodity hardware that is readily available from a variety of sources. You will need an Intel-based machine of recent manufacture. While Appscio MPF can be installed on a variety of hardware configurations, the following is the *minimum* configuration recommended for development:

- One processor
- 1GB memory
- 100MB disk space for core Appscio MPF
- Disk space to accommodate any audio and video files you may need during development

Note: If your development will include significant transcoding activity, you will want to install Appscio MPF in a more robust hardware configuration (e.g. additional processors, memory).

Tip: Care needs to be taken to address the bandwidth required to process audio and video data, particularly when utilizing external storage.

4.2 Software Requirements

This section discusses the software requirements necessary to support Appscio MPF.

4.2.1 Operating System

The base Operating System for Appscio MPF is the Fedora 10 – Intel, 64-bit architecture. For reference, the following links should be reviewed:

- Download the Fedora 10 – 64 bit image: http://fedoraproject.org/en/get-fedora-all
- Read the Fedora 10 Release Notes: http://docs.fedoraproject.org/release-notes/f10/en_US/
- Reference the Fedora 10 Install Guide: http://docs.fedoraproject.org/install-guide/f10/en_US/

4.3 Appscio MPF Installation

This section discusses the installation of the Operating System, companion software, and the installation of Appscio MPF.

Note: To install Appscio MPF, please make sure you are logged in as the 'root' superuser.

4.3.1 Install the Operating System

Tip: The recommended Fedora installation path is to de-select all packages during the installation while selecting the “Fedora” and “Update” repositories for future updates. Please note that selecting these repositories during installation requires that you have an Internet connection present.

If you haven't done so already, download the Fedora 10, x86_64 distribution (http://fedoraproject.org/en/get-fedora-all) and create a DVD (http://docs.fedoraproject.org/readme-burning-isos/en_US) Use this DVD to install Fedora. During Fedora installation you will be asked to select packages for installation. Be sure to reference the Fedora 10 Install Guide (http://docs.fedoraproject.org/install-guide/f10/en_US/) for a trouble-free installation.

4.3.2 Install the Companion Software

Before installing Appscio MPF, let's add some gstreamer packages that will work in concert with Appscio MPF. To do so, launch a Terminal window and execute the following:

yum clean all
yum update -yv

The previous two commands make certain that your system is up to date with the latest package versions. Now, let's install the companion software. Still in the Terminal window, execute the following:

cd /tmp

wget http://download1.rpmfusion.org/free/fedora/rpmfusion-free-release-stable.noarch.rpm

rpm -iv rpmfusion-free-release-stable.noarch.rpm

wget http://download1.rpmfusion.org/nonfree/fedora/rpmfusion-nonfree-release-stable.noarch.rpm

rpm -iv rpmfusion-nonfree-release-stable.noarch.rpm

After installation of the *rpmfusion-free* and *rpmfusion-nonfree* packages, you should list out the gstreamer packages, which should now be installed. Do so by executing the following command:

yum list gst*

The output from the command above should look similar to this:

```
gstreamer-0.10.21-2.fc10
gstreamer-plugins-base-0.10.21-2.fc10
gstreamer-plugins-flumpegdemux-0.10.15-4.fc10
gstreamer-plugins-good-0.10.13-1.fc10
gstreamer-tools-0.10.21-2.fc10
```

Beyond the gstreamer packages installed to this point, a few additional packages are also useful. To install the additional gstreamer packages, execute the following in the Terminal window:

yum install gstreamer-ffmpeg gstreamer-plugins-bad gstreamer-plugins-ugly gst-inspector gst-launch gstreamer-devel

With these packages installed, the next step is to install the Appscio MPF software.

4.3.3 Appscio MPF Installation

This next section discusses the various paths you can take to install Appscio MPF. For most people, installing Appscio MPF from available binary RPMs is the easiest path to follow. So, let's examine this installation path first and then discuss the other options for installation in the sections that follow.

4.3.3.1 Appscio MPF Binary RPM Installation

To install Appscio MPF via the binary RPM install path, execute the following in a Terminal window:

wget http://www.appscio.com/repository/packages/release/fedora/10/RPMS/appscio.repo -O /etc/yum.repos.d/appscio.repo

yum install mpf-core

yum install mpf-iplimage

yum install mpf-opencv-opticalflow

yum install mpf-rdf

yum install mpf-template

Following the installation of Appscio MPF, the following packages should be installed at a minimum:

Appscio MPF Packages
mpf-core
mpf-iplimage
mpf-opencv-opticalflow
mpf-rdf
mpf-template

Note: If you plan on developing components, you will need the development packages for MPF (e.g. mpf-core-devel). You can view the available development packages using the *yum list mpf** command.

To verify your MPF installation, type the following in the Terminal window:

yum list mpf*

Your output from the above command should display the Appscio MPF packages as shown above.

Appscio MPF is now installed and ready to use. This completes the binary RPM installation path.

Note: It is not necessary to execute the installation methods shown below if you chose the binary RPM installation option as documented in the previous section. With Appscio MPF installed, you can go to section five of this document as a next step.

4.3.3.2 Appscio MPF Source RPM Installation

As an alternative, you may wish to install Appscio MPF from “Source RPMs” and Appscio makes this installation path available as an option. This section assumes that you have installed the Operating System and the Companion Software as previously described. If you are installing using the “Source RPM” approach, it is not necessary to execute the tasks in section 4.3.3.1 (previous section).

To begin the “Source RPM” install path, open a Terminal window and execute the following:

yum install rpm-build

This command installs the RPM Build package, which we will use later in this section to create the RPMs to complete the installation.

Now, let's obtain the "Source RPMs" from Appscio. Execute the following in the Terminal:

cd /tmp

Note: When you install from "Source RPMs" your package version numbers may differ slightly from what is shown here depending on when you install. You may wish to first visit the "Source RPMs" site http://www.appscio.com/repository/packages/release/fedora/10/SRPMS/ to verify the current version numbers. Then obtain and install the packages as shown below.

Here's an example of obtaining the "Source RPMs". You can follow the packaging naming convention to receive the appropriate packages for installation, but note that the version numbers may differ slightly.

```
wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/grdf-0.0.1.1-snapshot.r682.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-ams-librarysink-0.0.1.0-snapshot.r586.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-buffers-0.0.1.0-snapshot.r692.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-core-0.0.3.1-snapshot.r693.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-gst-modified-0.0.1.0-snapshot.r680.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-iplimage-0.0.3.1-snapshot.r641.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-opencv-opticalflow-0.0.3.0-snapshot.r633.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-rdf-0.0.1.0-snapshot.r687.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-rdftest-0.0.3.0-snapshot.r676.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/mpf-template-0.0.1.0-snapshot.r698.src.rpm

wget http://www.appscio.com/repository/packages/release/fedora/10/
SRPMS/voidraw-0.0.1.0-snapshot.r579.src.rpm
```

After executing the ***wget*** commands above, you should now have the "Source RPMs" in your */tmp* directory and they should look similar to the following:

```
grdf-0.0.1.1-snapshot.r682.src.rpm
mpf-ams-librarysink-0.0.1.0-snapshot.r586.src.rpm
mpf-buffers-0.0.1.0-snapshot.r692.src.rpm
mpf-core-0.0.3.1-snapshot.r693.src.rpm
mpf-gst-modified-0.0.1.0-snapshot.r680.src.rpm
mpf-iplimage-0.0.3.1-snapshot.r641.src.rpm
```

```
mpf-opencv-opticalflow-0.0.3.0-snapshot.r633.src.rpm
mpf-rdf-0.0.1.0-snapshot.r687.src.rpm
mpf-rdftest-0.0.3.0-snapshot.r676.src.rpm
mpf-template-0.0.1.0-snapshot.r698.src.rpm
voidraw-0.0.1.0-snapshot.r579.src.rpm
```

The next step is to build the RPMs for installation on your system and to account for package interdependencies along the way.

Using the Terminal, execute the following:

cd /tmp

Before building the RPMs, please verify that you have installed the following packages:

```
check
check-devel
check-static
```

You can do this verification by executing:

yum list check*

If you are missing any of the three packages, they can be installed by executing the following:

yum install <package name>

With these packages installed, you're now ready to build and install the RPMs.

Note: We install the development (and debug) packages as an optional step. You will want to install them should you plan to develop MPF components. Use the *yum list mpf** command to view available development (and debug) packages.

Using the Terminal, execute the following:

rpmbuild --rebuild grdf-0.0.1.1-snapshot.r682.src.rpm

yum install gstreamer-devel gstreamer-plugins-base-devel -yv

rpmbuild --rebuild mpf-core-0.0.3.1-snapshot.r693.src.rpm

cd /<home>/rpmbuild/RPMS/x86_64

rpm -i mpf-core-0.0.3.1-snapshot.r693.src.rpm

rpm -i mpf-core-debuginfo-0.0.3.1-snapshot.r693.x86_64.rpm

rpm -i mpf-core-devel-0.0.3.1-snapshot.r693.x86_64.rpm

The above commands will install the core Appscio MPF package. Now, let's build and install some additional packages for Appscio MPF.

cd /tmp

rpmbuild --rebuild mpf-buffers-0.0.1.0-snapshot.r692.src.rpm

rpmbuild --rebuild mpf-gst-modified-0.0.1.0-snapshot.r680.src.rpm

yum install opencv opencv-devel -yv

rpmbuild --rebuild mpf-iplimage-0.0.3.1-snapshot.r641.src.rpm

cd /<home>/rpmbuild/RPMS/x86_64

rpm -i grdf-0.0.1.1-snapshot.r682.x86_64.rpm

rpm -i grdf-devel-0.0.1.1-snapshot.r682.x86_64.rpm

rpm -i grdf-debuginfo-0.0.1.1-snapshot.r682.x86_64.rpm

cd /tmp

rpmbuild --rebuild mpf-rdf-0.0.1.0-snapshot.r687.src.rpm

cd /<home>/rpmbuild/RPMS/x86_64

rpm -i mpf-rdf-0.0.1.0-snapshot.r687.x86_64.rpm

rpm -i mpf-rdf-devel-0.0.1.0-snapshot.r687.x86_64.rpm

rpm -i mpf-rdf-debuginfo-0.0.1.0-snapshot.r687.x86_64.rpm

cd /tmp

rpmbuild --rebuild mpf-ams-librarysink-0.0.1.0-snapshot.r586.src.rpm

cd /<home>/rpmbuild/RPMS/x86_64

rpm -i mpf-iplimage-0.0.3.1-snapshot.r641.x86_64.rpm

rpm -i mpf-iplimage-devel-0.0.3.1-snapshot.r641.x86_64.rpm

rpm -i mpf-iplimage-debuginfo-0.0.3.1-snapshot.r641.x86_64.rpm

cd /tmp

rpmbuild --rebuild mpf-opencv-opticalflow-0.0.3.0-snapshot.r633.src.rpm

rpmbuild --rebuild mpf-rdftest-0.0.3.0-snapshot.r676.src.rpm

rpmbuild --rebuild mpf-template-0.0.1.0-snapshot.r698.src.rpm

rpmbuild --rebuild voidraw-0.0.1.0-snapshot.r579.src.rpm

Now, let's install the remaining packages, as follows:

cd /<home>/rpmbuild/RPMS/x86_64

rpm -i mpf-ams-librarysink-0.0.1.0-snapshot.r586.x86_64.rpm

rpm -i mpf-ams-librarysink-debuginfo-0.0.1.0-snapshot.r586.x86_64.rpm

rpm -i mpf-buffers-0.0.1.0-snapshot.r692.x86_64.rpm

rpm -i mpf-buffers-debuginfo-0.0.1.0-snapshot.r692.x86_64.rpm

rpm -i mpf-gst-modified-0.0.1.0-snapshot.r680.x86_64.rpm

rpm -i mpf-gst-modified-debuginfo-0.0.1.0-snapshot.r680.x86_64.rpm

rpm -i mpf-opencv-opticalflow-0.0.3.0-snapshot.r633.x86_64.rpm

rpm -i mpf-opencv-opticalflow-debuginfo-0.0.3.0-snapshot.r633.x86_64.rpm

rpm -i mpf-rdftest-0.0.3.0-snapshot.r676.x86_64.rpm

rpm -i mpf-template-0.0.1.0-snapshot.r698.x86_64.rpm

rpm -i mpf-template-debuginfo-0.0.1.0-snapshot.r698.x86_64.rpm

rpm -i voidraw-0.0.1.0-snapshot.r579.x86_64.rpm

rpm -i voidraw-debuginfo-0.0.1.0-snapshot.r579.x86_64.rpm

This completes the installation of Appscio MPF using the “Source RPMs” install path. You should now reference section five of this document for next steps.

4.3.3.3 Appscio MPF Source Tar Installation

Appscio MPF can also be installed from *tar* files containing source code, which can be found at:

http://www.appscio.com/repository/packages/release/SOURCES/

To install from the *tar* files, create a directory under /tmp. Download the archive files for each MPF component and decompress them into your temporary directory (e.g. /tmp/mpf-source-install). Once de-compressed, you will see a sub-directory for each MPF component. Change to the sub-directory for each component and issue the ***./configure***, ***make***, ***make install*** command sequence to complete the installation and then reference section five of this document.

5. Getting Started With Appscio MPF

The Appscio MPF Installation Guide is focused on helping you get Appscio MPF up and running. To start working with Appscio MPF, please reference the companion document, Appscio MPF Development Guide, for the steps needed to start working with Appscio MPF.

Note: If you have experienced any issues during the installation of Appscio MPF, please visit the Appscio Community Web site (www.appscio.org) and post a message in the Forums area of the site.

6. Appendices

Appendix A: Glossary

In this section we provide definitions for common terms you may uncover while working with Appscio MPF:

Term	Definition
Component	Appscio MPF basic unit of metadata-handling function (wraps Element)
Element	gstreamer's basic unit of media-handling function
IplImage	An OpenCV image definition
Metadata	Content description - or - data about data
MPF	Media Processing Framework
Ontology	A formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain.
OpenCV	The Open Source Computer Vision Library
Pad	Interconnection points for passing data between components in a pipeline
Pipeline	A sequence of Algorithms (wrapped with Appscio (tm) MPF code)
ROI	Region Of Interest
RDF	Resource Description Framework
SDK	Software Development Kit

Appendix B: Naming Conventions

This section discusses the naming conventions currently used in Appscio MPF.

- For RPM binaries: package_name-major.minor.build<.nano>-release_tag.i386.rpm
- For Source RPMs: *package_name-major.minor.build<.nano>-release_tag.src.rpm*
- For tarballs: *package_name-major.minor.build<.nano>-release_tag.tar.gz*

Naming convention examples:

- Pre-release RPM binary: *mpf-core-0.0.3.2-pre-0.0.4.fc9.i386.rpm*
- Pre-release Source RPM: *mpf-core-0.0.3.2-pre-0.0.4.src.rpm*
- Pre-release tarball: *mpf-core-0.0.3.2-pre-0.0.4.tar.gz*
-
- Release RPM binary: *mpf-core-0.0.4-fc9.i386.rpm*
- Release Source RPM: *mpf-core-0.0.4-src.rpm*
- Release tarball: *mpf-core-0.0.4-tar.gz*

- Maintenance RPM binary: *mpf-core-0.0.4-1.fc9.i386.rpm*

Appendix C: Recommended Background Material

The following background material in recommended to gain greater insight into how Appscio is approaching the management of AV data:

- gstreamer Application Development Manual
  http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html

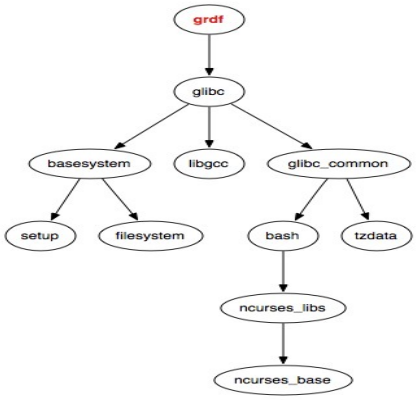
- RDF Primer
  http://www.w3.org/TR/rdf-primer

In addition to the above, visit the Appscio Community Web site (www.appscio.org) to learn more about Appscio MPF and to see what others in the Community are doing with Appscio MPF.

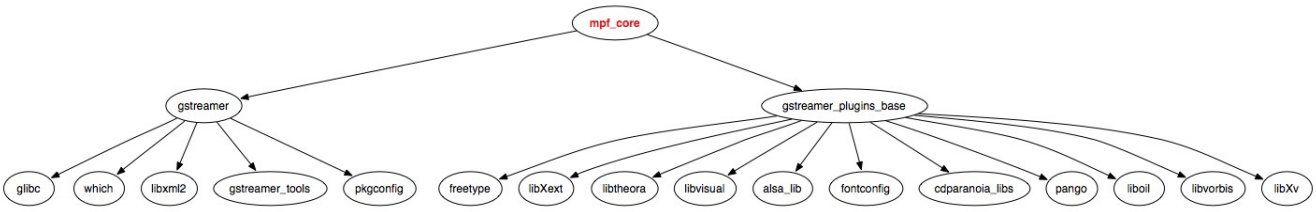
Appendix D: Appscio MPF Dependency Diagrams

This section outlines the high-level dependencies for each of the Appscio MPF packages for your reference.

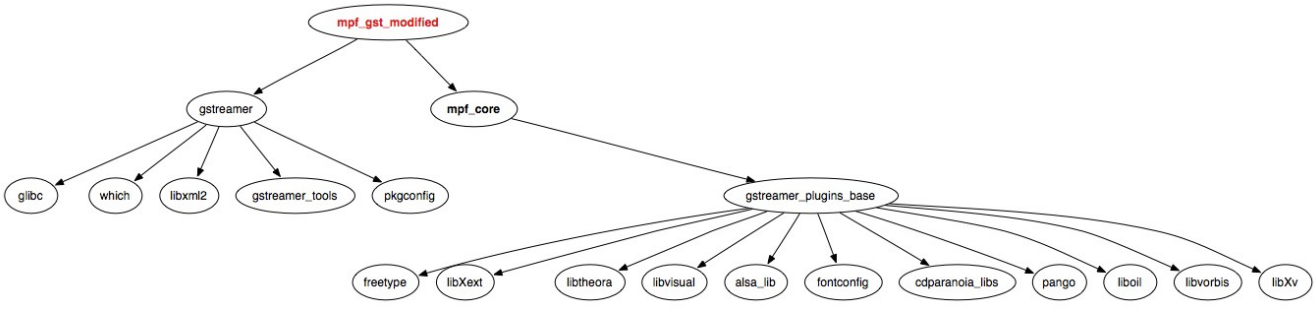
The first diagram is for the *grdf* package:



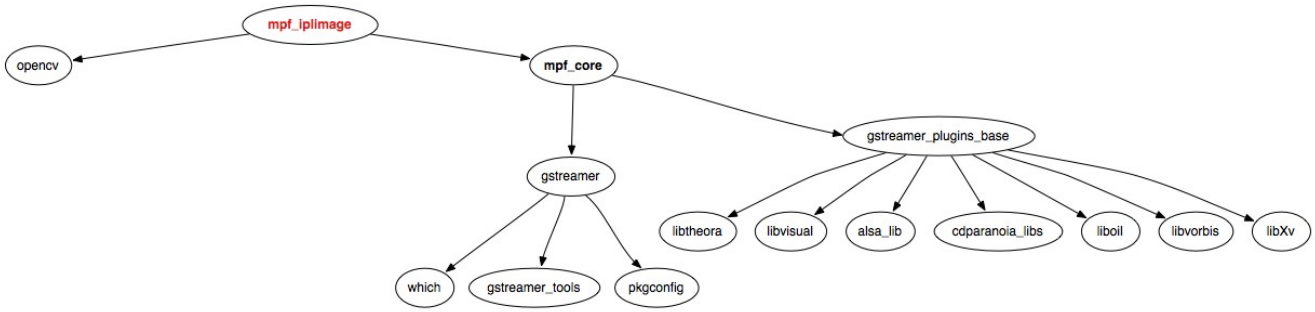
The next diagram shows the dependencies for *mpf-core*:



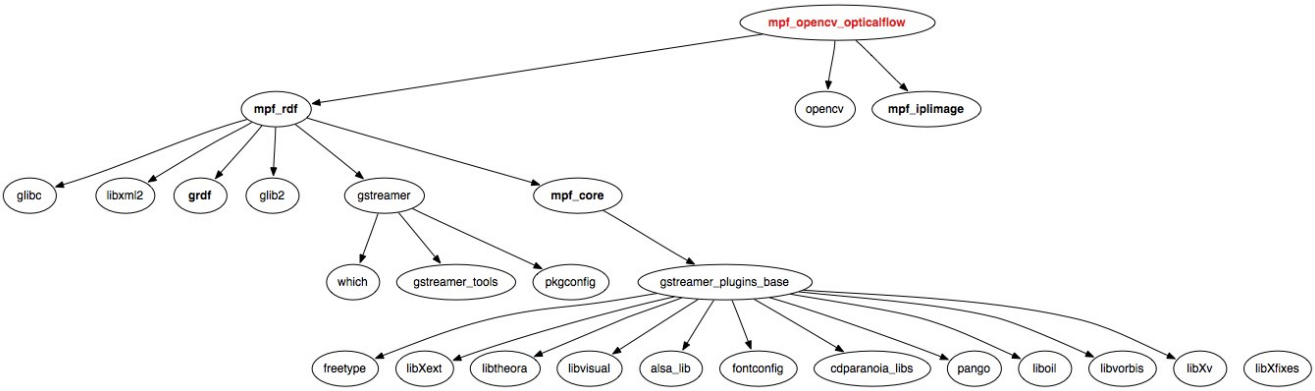
Here are the dependencies for *mpf-gst-modified*:



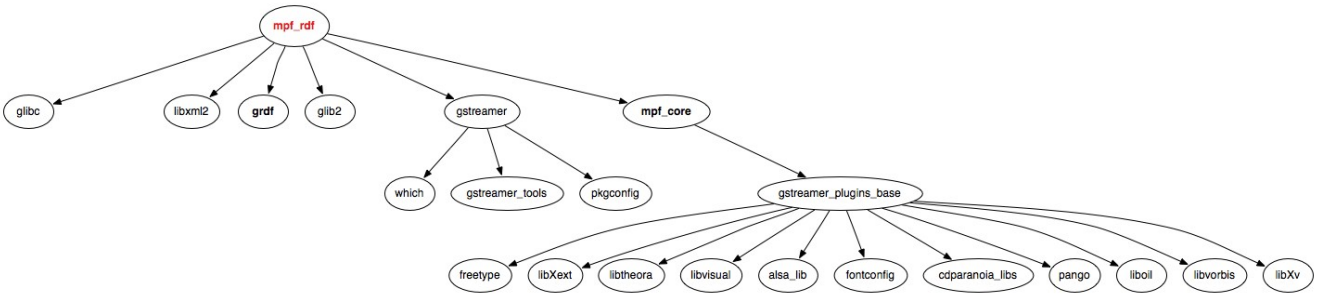
The next diagram shows the dependencies for the *mpf-iplimage* package:



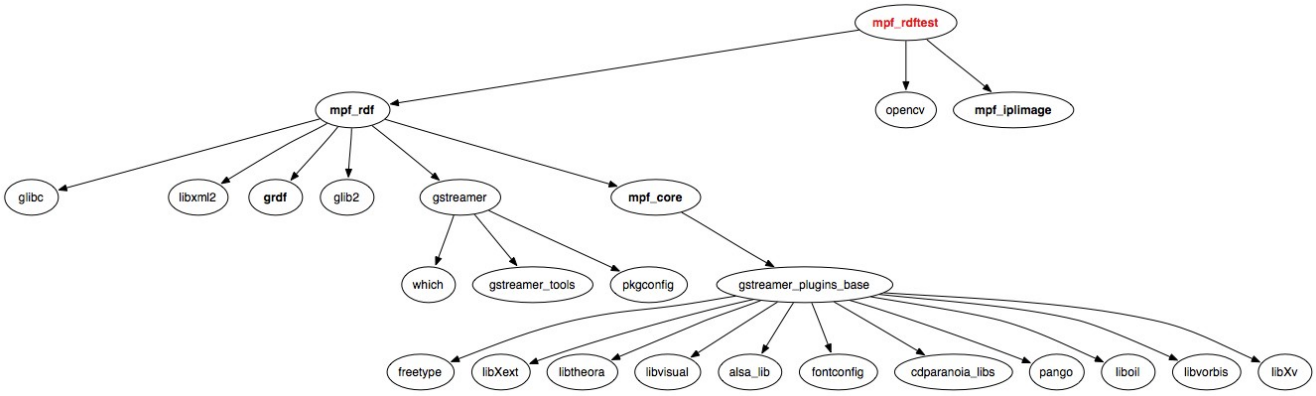
Here are the dependencies for the *mpf-opencv-opticalflow* package:



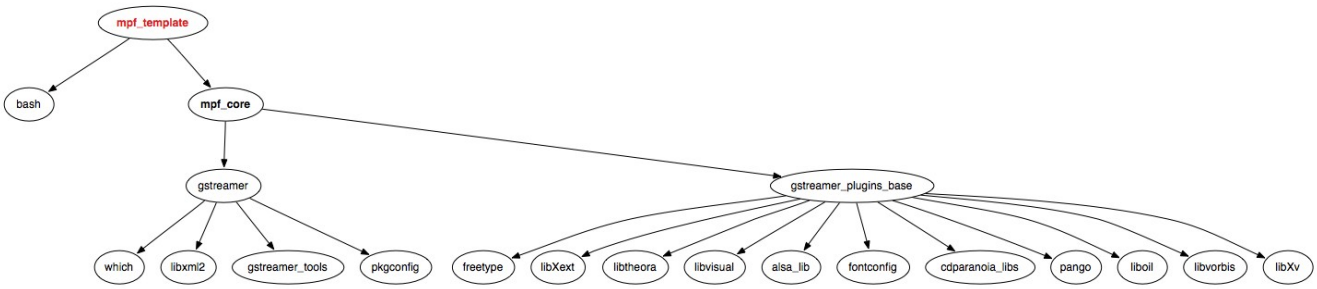
Next, here are the dependencies for the *mpf-rdf* package:



This next diagram shows the dependencies for the *mpf-rdftest* package:



And, lastly, this diagram depicts the dependencies for the *mpf-template* package:



Alphabetical Index

#########