Integrated Computer Solutions

Dr. Roland Krause

Video: http://bit.ly/IVI-Media-Manager

Motivation

- At ICS we have designed and built many In-Vehicle-Infotainment systems (see e.g. this video)
- When asked to look into Media Management we found this to be a vexing and complex problem
- The challenge for automotive IVI implementations is that
  - People's media -- their music, videos, audiobooks, podcasts and television -- exist in a multitude of forms and originate from many disparate sources.
  - For example, some music files may reside at home in an iTunes library, others may have been purchased from Amazon Music or Google Play.
  - Media may have then been downloaded to a computer, a USB drive or a phone, or stored on a cloud server.
  - Management of digital rights adds yet another layer of complexity to the situation -- one that can't be ignored.

Requirements

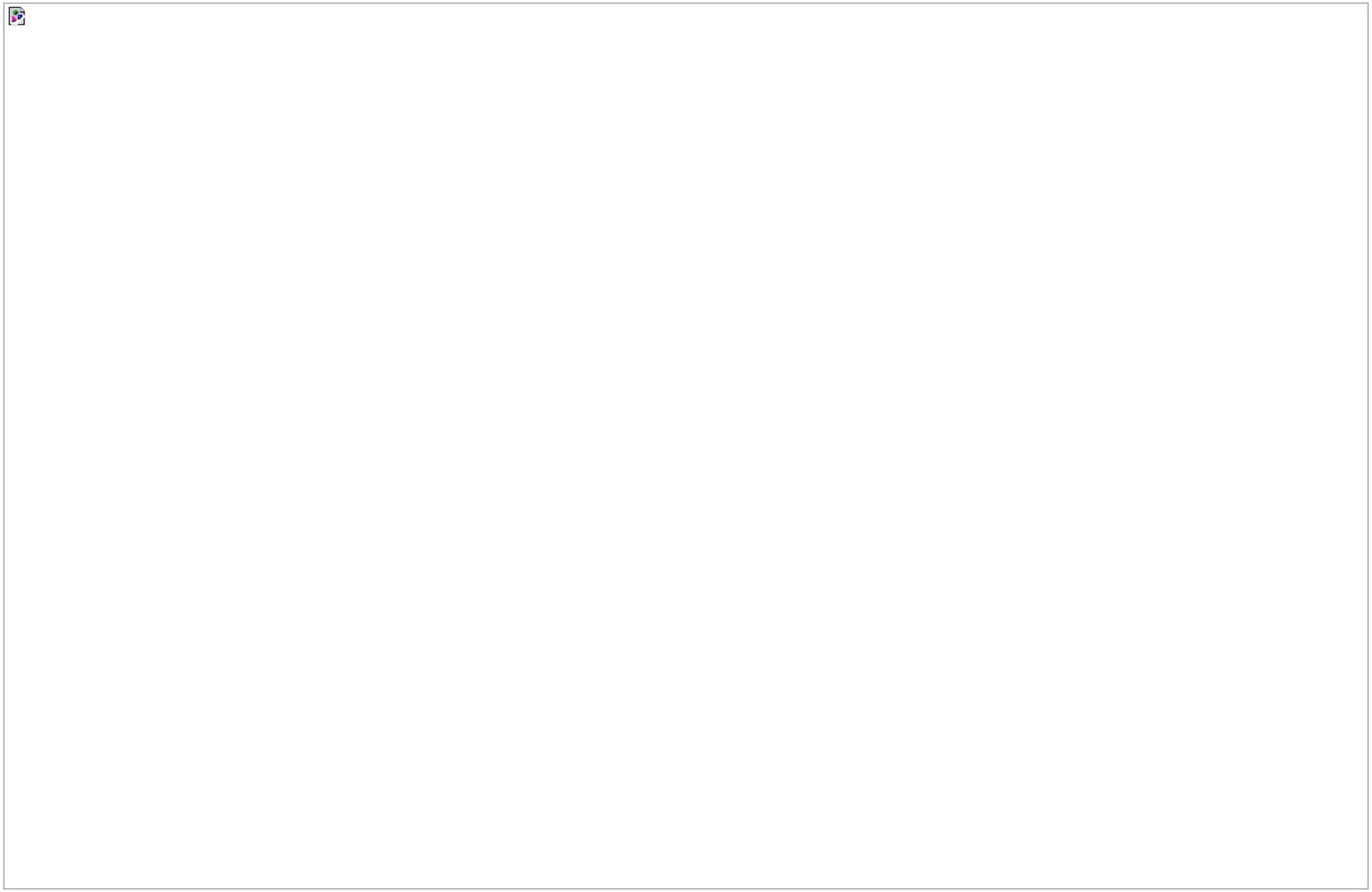
- The job of finding and making available media to the passengers of a car is that of the Media Manager
- First step: Recognizing a Device is brought into the car
- Next: Finding and Indexing Media on the Device
- Possibly: Enhancing Media Information to allow improved search, filtering, etc..
- Definitely: Playing of Media using the car's advanced audio systems
- Controlling the flow of Media to e.g. different Speaker Zones, Headphones, Videos to headrest screens etc..
    - When multiple occupants drive in the car each individual should be able to enjoy their own audio and video selections.
    - Hence a media manager should be able to direct media to specific passengers.

The Idea - Coding for the Unknown

Today's media consumers behavior changes rapidly:

- Remember the "Walkman" - Enjoyed it for Decades
- CDs - Lasted maybe 10 years
- MP3s on CDs, USB Pendrives, Less than 5 years
- Cloud based music sharing, Amazon tbd.
- Streaming:
- Pandora, By all means not saying it's dead but:
- Spotify, is the current Darling (< 2 years)
- What is next?
  - The cycles become shorter and shorter
  - Consumers change phones 2-3 years on average
- We must keep in mind that what we create might be partially outdated by the time is is released - Ouch!

Architecture

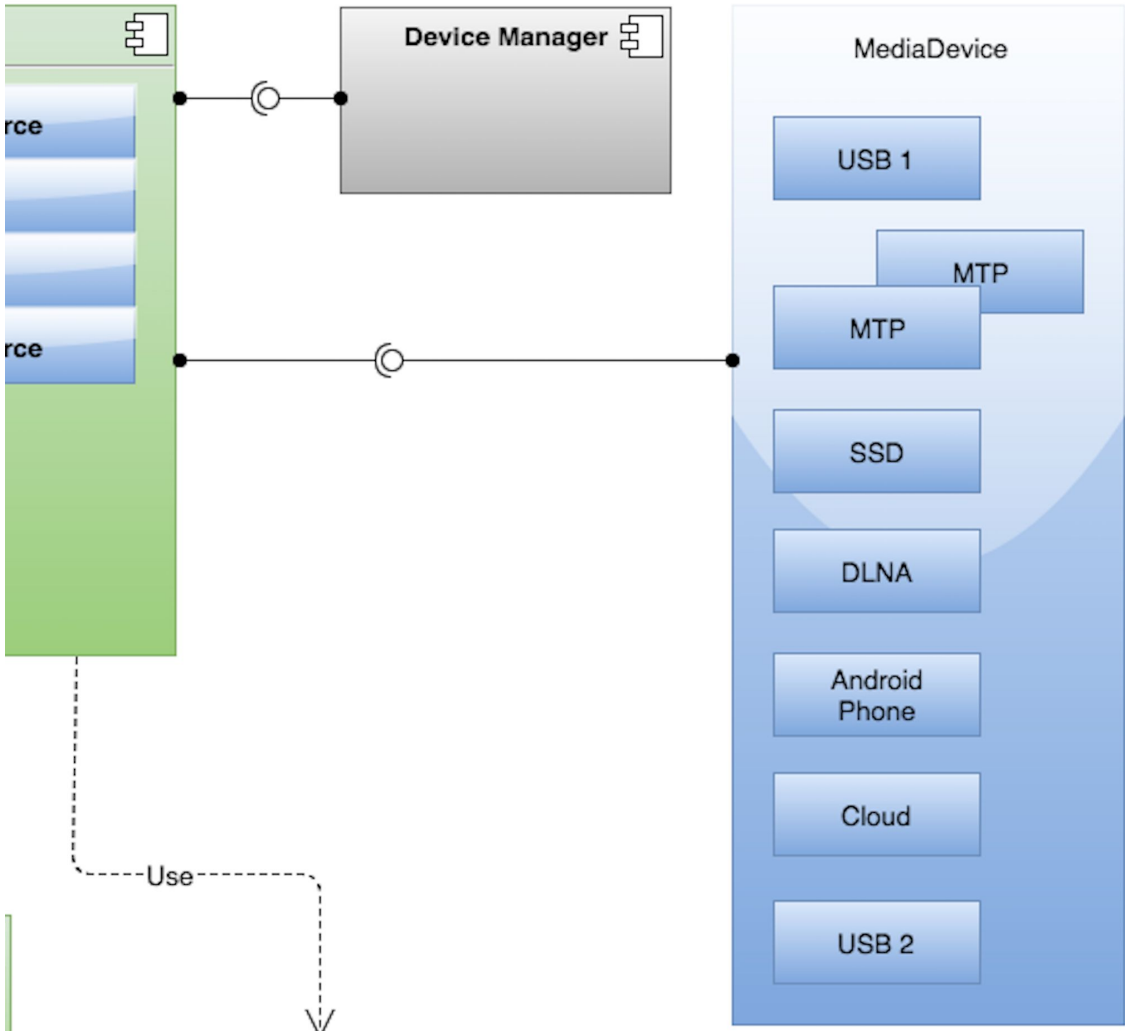


Plugin Architecture

Media Manager Core Functionality

- Load Plugins:
  - Device Manager
  - Media Devices
  - Media Players
  - Services (Audio Manager, Media Enrichment)
  - Controllers (UI, RC, RVI)
- Organize Flow of Media Info Data from Device to Player and Device to Controller

Media Devices



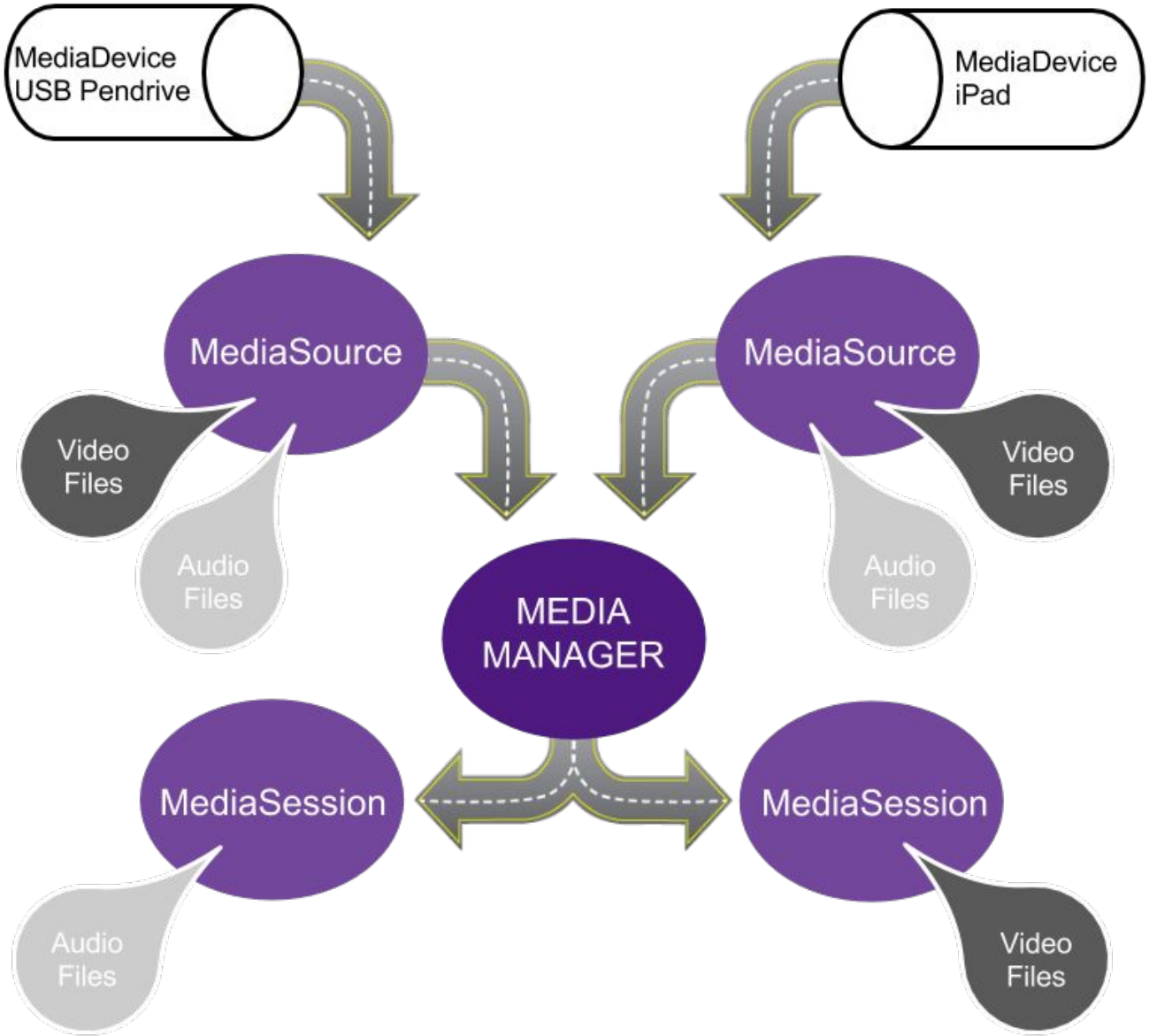
Device Manager Plugin Interface

```
/** DeviceManagerInterface is a Plugin Interface for DeviceManagers
*   that detect MediaDevices which contain Media that can be indexed
*   by a suitable MediaDevice.
**/
class DeviceManagerInterface : public QObject
{
   Q_OBJECT
public:
   explicit DeviceManagerInterface(QObject * parent=0) : QObject(parent) {}
   virtual ~DeviceManagerInterface() {}
signals:
   void deviceCreated(const QString mediaDeviceType, const QUrl mediaDevicePath) const;
   void deviceRemoved(const QString mediaDeviceType, const QUrl mediaDevicePath) const;
};

#define DeviceManagerInterface_iid "com.ics.media-manager.DeviceManagerInterface"
Q_DECLARE_INTERFACE(DeviceManagerInterface, DeviceManagerInterface_iid)
```

Device Manager Plugin Implementation

```
/** SimpleDeviceManager is a Plugin Interface for a Device Manager
*  that watches system mounted devices such as USB Pendrives and SD Cards.
**/
class SimpleDeviceManager : public DeviceManagerInterface
{
   Q_OBJECT
   Q_PLUGIN_METADATA(IID "com.ics.media-manager.DeviceManagerInterface" FILE "SimpleDeviceManager.
json")
   Q_INTERFACES(DeviceManagerInterface)
public:
   SimpleDeviceManager(QObject *parent = 0);

protected slots:
   void deviceChanged();

private:
   QFileSystemWatcher * m_usbWatcher;
   QString m_usbWatchPath;
   QStringList m_devices;
};
```

Data Flow



Core Components

- MediaSource
  - Provide interfaces to devices.
  - Devices are physical media such as Phones, iPads, USB thumb drives, Microsoft Media Players, DLNA, Bluetooth, cloud or any source that can be indexed.
- MediaSource Playlists
  - Each source presents to the media manager one or more source playlists.
  - The media manager takes these lists and add them to corresponding MediaSessions.
  - For example, video playlists are offered to the session that interfaces to a video player, whereas Bluetooth playlists are offered to a Bluetooth Player which in turn controls a Bluetooth device through the AVRCP protocol.

MediaSession and MediaSource

- MediaSession
  - Each MediaSession holds a playlist of tracks specific to a media type e.g. mp3 files, video files or Bluetooth streams.
  - MediaSession interfaces a single instance of a media player for the specific media type.
  - Contains a JSON Object consisting of multiple JSON Arrays,
  - One per MediaType present on the device.
- MediaPlaylist is a JSON Array
  - Each JSON Array contains indexing data
  - Indexing data are JSON Objects,
    - one for each media item
    - containing attributes of a single media item
    - e.g., file names, artists, cover art
      and many other things of interest to the end user.

Class MediaSource

```
class MediaSource : public QObject
{
   Q_OBJECT
   Q_PROPERTY(QJsonObject mediaSourcePlaylist READ mediaSourcePlaylist  WRITE setMediaSourcePlaylist
              NOTIFY mediaSourcePlaylistChanged)
   Q_PROPERTY(QUrl deviceUrl READ deviceUrl WRITE setDeviceUrl NOTIFY deviceUrlChanged)
public:
   explicit MediaSource(MediaDeviceInterface * device, const QUrl & deviceUrl, QObject * parent = 0 );
   void updateMediaSourcePlaylist() ;
   QJsonObject mediaSourcePlaylist() const;
   bool hasMediaType(const QString & mediaTypeStr) const;
   const QJsonArray mediaArray(const QString & mediaTypeStr) const;
   const QString deviceUrlString() const;
   const QUrl deviceUrl() const;
signals:
   void mediaSourcePlaylistChanged(const MediaSource * mediaSource);
   void deviceUrlChanged(QUrl deviceUrl);
private slots:
   void setDeviceUrl(QUrl deviceUrl);
   void setMediaSourcePlaylist(QJsonObject mediaSourcePlaylist);
private:
   MediaDeviceInterface * m_device;
   QJsonObject m_mediaSourcePlaylist;
   QUrl m_deviceUrl;
};
```

DataStructure: MediaPlaylist


```
{
    "AudioFileMediaType": [
        {
            "Album": "Southernality",
            "Artist": "A Thousand Horses",
            "CompleteName": "/mm_test/audio/a.mp3",
            "Title": "(This Ain’t No) Drunk Dial",
        },
        {
            "Album": "Billboard Top 60 Country Songs",
            "Artist": "Big & Rich",
            "CompleteName": "/mm_test/audio/b.mp3",
            "Title": "Run Away with You",
        }
    ],
    "VideoFileMediaType": [
        {
            "CompleteName": "/mm_test/video/mad_max.mp4",
            "FileName": "mad_max",
            "Format": "MPEG-4",
            "InternetMediaType": "video/mp4",
        },
        {
            "CompleteName": "/mm_test/video/sup-vs-bat.mp4",
            "FileName": "sup-vs-bat",
            "Format": "MPEG-4",
            "InternetMediaType": "video/mp4",
        }
    ]
}
```

Class MediaSession

```
class MediaSession : public QObject
{
   Q_OBJECT
public:
   explicit MediaSession(MediaPlayerInterface * player, QObject *parent = 0);

   void appendMediaSourcePlaylist(const QString deviceUrl, const QJsonArray playlist);
   void removeMediaSourcePlaylist(const QString deviceUrl);

   const MediaPlayerInterface * player() const { return m_player;}

   const QJsonArray mediaSessionPlaylist() const ;

signals:
   void mediaSessionPlaylistChanged(QStringList mediaSessionPlaylist);
private:

   void rebuildMediaSessionPlaylist();

   MediaPlayerInterface * m_player;
   QMap<QString,QJsonArray> m_sourcePlaylists;
};
```

Core Components

- MediaPlayer
  - MediaPlayers control the output of media
  - Implement functionality of media reproduction e.g., play, pause, stop, play index, play next, play previous etc.
  - Can also be controlled to direct output to specific channels through an audio manager component.
- MediaManager
  - MediaManager maintains a set of session objects and a set of source objects.
  - Interfaces with an audio manager for audio channels
  - Interfaces with a device manager for device notifications.
  - Provides a controller interface which allows for direct user interface (UI) implementation using the toolkit of choice as well as remote control through RVI or web interfaces.

Media Manager Core Functionality

- When a device is connected, e.g. a USB pen drive is plugged in:
  - Media Manager receives a notification from the Device Manager plugin.
  - With the help of a suitable MediaDevice indexing results in a MediaSource object - delivered to and received by the Media Manager.
  - Media Manager stores and accesses MediaSession objects corresponding to MediaTypes contained in the MediaSource
  - Appends the playlists coming from the MediaSource object to the MediaSession.
  - During this step, filtering and sorting can be applied.
- MediaSessions store sets of playlists (in JSON Arrays)
  - Identified with the MediaSource they came from
  - It is trivial to update playlists upon removal of a device at the cost of rebuilding the playlist and transferring it to the MediaPlayer again.
  - Use of “implicitly shared” container classes is fundamental to a robust and efficient implementation.

MediaManager Update Session

```
void MediaManager::updateMediaSession(const MediaSource * mediaSource)
{
    const QString deviceUrlStr=mediaSource->deviceUrlString();
    for (int mt=mmTypes::NoType+1;mt<mmTypes::EndType;++mt) {
        MediaType mediaType=(const MediaType)mt;
        QString mediaTypeStr=mmTypes::Media(mediaType);
        const QJsonArray playListArray=mediaSource->mediaArray(mediaTypeStr);
        if (playListArray.isEmpty()) {
            continue;
        }
        MediaSession * mediaSession=0;
        if (mediaSessions.contains(mediaType)) {
            mediaSession=mediaSessions[mediaType];
        }
        else {
            MediaPlayerInterface * player=0;
            if (mediaPlayerPlugins.contains(mediaType)) {
                player=mediaPlayerPlugins[mediaType];
            }
            else {
                qWarning() << Q_FUNC_INFO << "no plugin found for MediaType" << mediaType;
                continue;
            }
            mediaSession=new MediaSession(player, this);
            mediaSessions.insert(mediaType, mediaSession);
        }
        mediaSession->appendMediaSourcePlaylist(deviceUrlStr,playListArray);
        if (mediaSessions.count()==1)
            setActiveMediaSession(mediaType);
        else
            emit activeMediaSessionPlaylist(mediaSessions[activeMediaSessionType]->mediaSessionPlaylist());
    }
}
```

MediaSession: Rebuild Playlist

```
void MediaSession::rebuildMediaSessionPlaylist()
{
   QStringList  playList;
   foreach (const QJsonArray jsa, m_sourcePlaylists) {
       foreach (QJsonValue jv, jsa) {
           QJsonObject jo=jv.toObject();
           const QString & f=jo["CompleteName"].toString();
           playList << QString("file:%1").arg(f);
       }
   }
   emit mediaSessionPlaylistChanged(playList);
   m_player->setMediaPlaylist(playList);
   qDebug() << Q_FUNC_INFO << playList;
}
```

Indexer - MediaInfo

- The problem of indexing media is two-fold:
  - Files must be found, identified and the results stored.
  - Media contained in files and streams must be classified.
  - We are looking to answer questions like:
    - How long is this “mp3” file? Who sang this song? Who directed this orchestra?
  - All of this information should be available to the user as fast as the medium permits while preserving an always responsive, modern user experience.

Indexer - MediaInfo

- Notable Open Source indexing solutions:
  - Gnome projects Tracker and KDE's Nepomuk are powerful and complex search and indexing solutions
  - appropriate for desktop solutions.
  - Light Media Scanner (LMS), and FFMpeg project's ffprobe are more suited for constrained environments and use cases.
- Another widely used option is MediaInfo
  - Highly customizable,
  - can easily be integrated in C++ based applications,
  - has support for hundreds of media types and is a fast and robust solution with a long standing track record.

Indexer - MediaInfo

- While MediaInfo itself provides a command line utility that can produce a wide variety of output formats we will integrate MediaInfo as a library into a Qt based application that will output metadata information in JSON format.

FileSystemDevice uses MediaInfo

```
class IndexingWorker : public QObject
{
public:
   IndexingWorker(QObject * parent=0)
       : QObject(parent) {
       audioFilters << "*.mp3" << "*.ogg" << "*.wav";
       videoFilters << "*.mp4" << "*.mkv" << "*.m4v" << "*.avi" << "*.wmv" << "*.mov";
   }
   ~IndexingWorker(){}

   void startIndexing(const QUrl url);

   QJsonArray audioFiles() const {return m_audioFiles;}
   QJsonArray videoFiles() const {return m_videoFiles;}
private:
   void indexDirectory(const QString dirPath);
   void mediaInfo(const QStringList fileList);

private:
   QJsonArray m_audioFiles;
   QJsonArray m_videoFiles;
   QStringList audioFilters;
   QStringList videoFilters;
};
```

Using MediaInfo

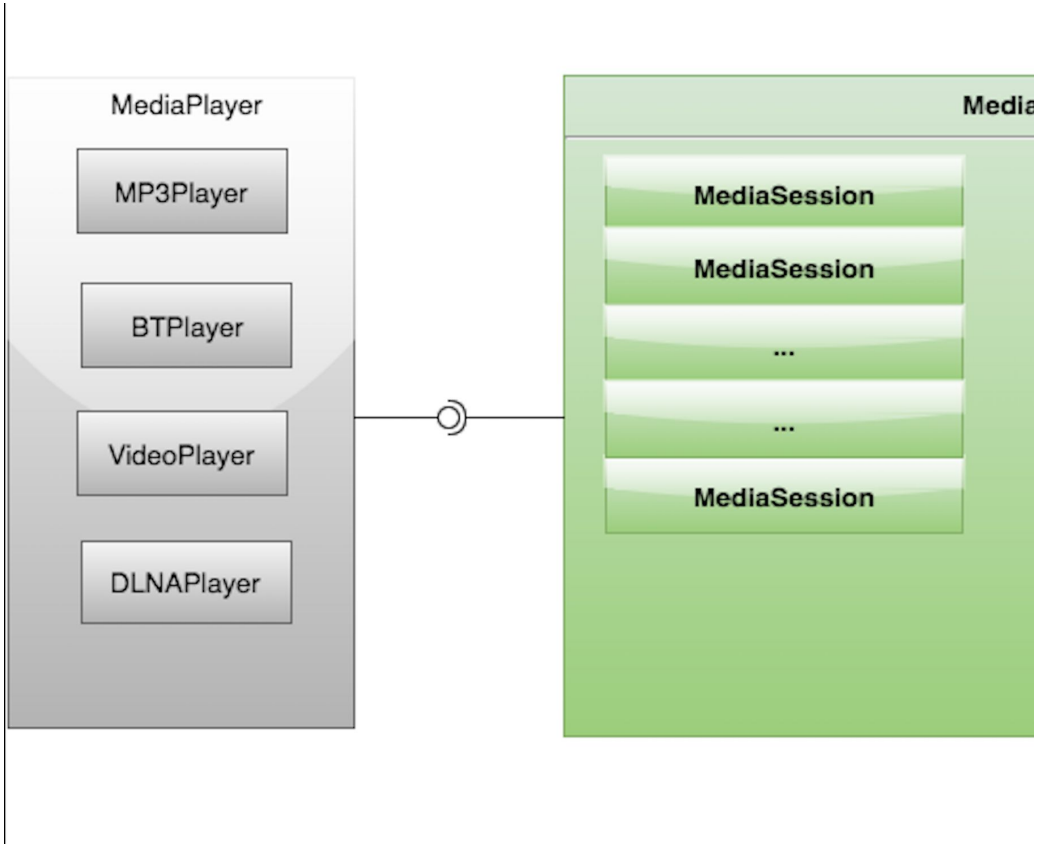
```
void IndexingWorker::mediaInfo(const QStringList fileList)
{
   if (fileList.isEmpty()) return;
   QStringList generalParams;
   generalParams << "CompleteName" << "FolderName" << "FileName" << "FileExtension" << "Artist" << "Cover_Data"
                 << "Format" << "InternetMediaType"
                 << "Title" << "Season" << "Movie"
                 << "Album" << "Album_More" << "Album/Performer";
   QString generalInform;
   generalInform="General;";
   foreach(QString s, generalParams) {
       generalInform += QString("\%%1\%|").arg(s);
   }
   generalInform+="\\n";
   MediaInfoLib::MediaInfoList MI;
   MI.Option(QSLWSTR("ParseSpeed"), QSLWSTR("0"));
   MI.Option(QSLWSTR("Language"), QSLWSTR("raw"));
   MI.Option(QSLWSTR("ReadByHuman"), QSLWSTR("0"));
   MI.Option(QSLWSTR("Legacy"), QSLWSTR("0"));
   int nfiles;
   foreach (QString file, fileList) {
       nfiles=MI.Open(file.toStdWString(), MediaInfoLib::FileOption_NoRecursive);
   }
   if (nfiles!=fileList.count()) {
       qWarning() << Q_FUNC_INFO << "some files could not be opened, have" << nfiles << "out of" << fileList.count();
   }

   MI.Option(QStringLiteral("Inform").toStdWString(), generalInform.toStdWString());
   QString informOptionExample=QString::fromStdWString(MI.Inform());
```

Retrieving Data from MediaInfo

```
QStringList informResult=informOptionExample.split('\n',QString::SkipEmptyParts);
QVariantMap resMap;
foreach (QString res, informResult) {
    QStringList resList=res.split("|");
    Q_ASSERT((resList.count()-1)==generalParams.count());
    for (int i=0;i<resList.count()-1;++i) {
        resMap[generalParams[i]] = resList[i];
    }
    QJsonObject resObject=QJsonObject::fromVariantMap(resMap);
    QString mimeType=resMap["InternetMediaType"].toString();
    if (mimeType.startsWith("audio")) m_audioFiles.append(resObject);
    else if (mimeType.startsWith("video")) m_videoFiles.append(resObject);
    else {
        qWarning() << Q_FUNC_INFO << "mimetype for file"
                   << resMap["CompleteName"]<< "not one of audio or video but"
                   << resMap["InternetMediaType"];
    }
}
```

MediaPlayers

- MediaPlayers do not provide UI control elements!
  - They do however have visible elements
  - E.g. video surfaces
  - Control is through a plugin interface



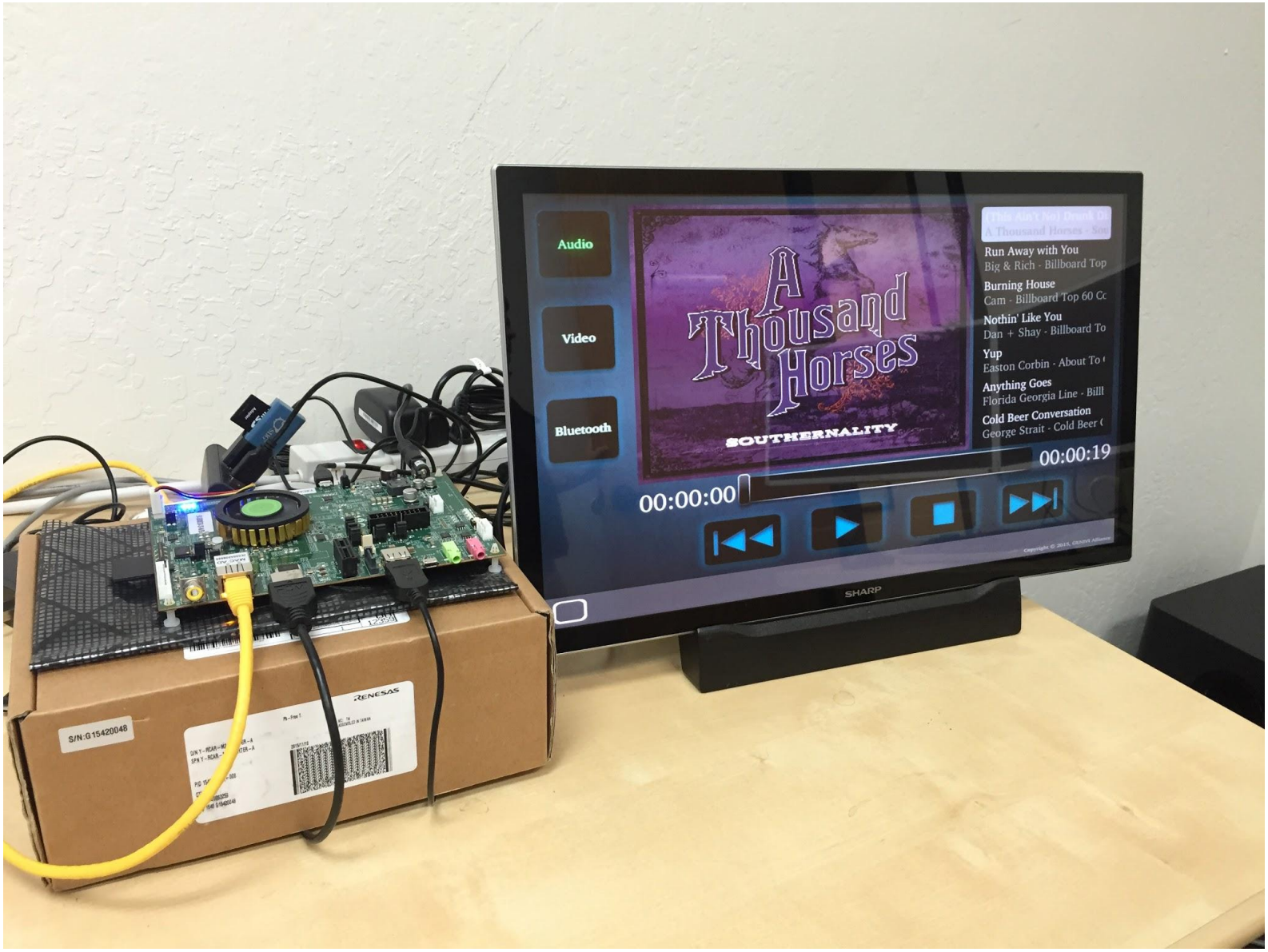
Media Players

```
class MediaPlayerInterface : public QObject
{
   Q_OBJECT
public:
   typedef mmTypes::PlayState PlayState;
   explicit MediaPlayerInterface(QObject * parent=0) : QObject(parent) {}
   virtual ~MediaPlayerInterface() {}
   virtual void setMediaPlaylist(const QStringList playList) = 0;
   virtual void addMediaPlaylist(const QStringList playList) = 0;
   virtual const QSet<QString> supportedFileSuffixes() const = 0;
   virtual PlayState playState() const = 0;
   virtual int currentTrackIndex() const = 0;

public slots:
   virtual void play() const = 0;
   virtual void pause() const = 0;
   virtual void stop() const = 0;
   virtual void next() const = 0;
   virtual void previous() const = 0;
   virtual void setCurrentTrack(int index) const = 0;
   virtual void setVideoRectangle(const QRect rect) const {Q_UNUSED(rect);}

signals:
   /** Must emit this signal so that the Controllers are notified
    *  when the current state changes **/
   void playStateChanged(PlayState state) const;
   /** Must emit this signal so that the Controllers are notified
    *  when the current track changes **/
   void currentTrackIndexChanged(int index);
};
```

Controllers

- Media Manager employs the concept of active MediaSessions
  - Control the actual playback of media.
  - It calls the active sessions player with the standard actions of playing
  - E.g. play, pause, next, previous, play by index etc..
  - The control of the Media Manager itself is through a MediaManagerControllerInterface that is implemented by a variety of “stateless” plugins.
  - E.g., a simple UI plugin allows for a graphical user interface to be implemented while a “remote controller” plugin allows mobile devices to control the Media Manager and thus its playing functionality.

QtQuick - UI Controller



Integration Genivi Development Platform

JSON Rpc Controller

- TCP based JSON-RPC
- Utilizes QJsonRpc
- Implements MediaManagerControllerInterface
- Same Interface as UI based Controllers
- Stateless Controller Architecture guarantees that all Controllers are in the same state
- Signal and Slot implementation allows to add controllers at will without changing the MediaManager code

Media Manager - App Demo

Part 1:

Part 2:

Next Steps

- Plugins and Indexers for Phones and Tablet devices - this will require mobile apps
- Plugins for DLNA devices
- Plugins for Bluetooth device playback
  - BlueZ, AVRCP and A2DP protocols

Conclusion

- As the vision of autonomous driving changes the role of the automobile itself:
  - Our Vision is to create software that allows the Automobile to be an integration point for Media
  - Similar to the “Connected Home”
  - Central point where Media “comes together”
- Architecture of our components should not withstand the developments of the future but adopt to it.
- Visit us - Talk to us - Work with us!
- How can we help you?