## Collaborators

<table>
<thead>
<tr>
<th>ACTION</th>
<th>NAME</th>
<th>DATE</th>
<th>SIGNATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Written By</td>
<td></td>
<td>February 9, 2015</td>
<td></td>
</tr>
</tbody>
</table>

## Revision History

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>DATE</th>
<th>DESCRIPTION</th>
<th>NAME</th>
</tr>
</thead>
</table>
Contents

I Murphy basics 1

1 Media control 2

1.1 Audio playback and recording rights 2

1.1.1 Stream types 2

Regular audio streams 2

Interrupt streams 2

1.1.2 Policy models for playback right 3

Enforced policy model 3

Cooperative policy model 3

1.1.3 Playback rights management 3

Policy aware applications 3

Policy unaware applications 3

1.2 Volume control 4

1.3 Routing 4

1.3.1 Explicit routing 4

1.3.2 Implicit routing 4

1.3.3 Configuration 5

Setting properties on the audio stream 5

Defining rules in the configuration file of pulseaudio-module-murphy-ivi 5

Media roles 5

Binary mapping 5

Environment variables for zone routing 6

2 Architecture 7

2.1 Components 7

2.2 Interfaces 7

2.2.1 HomeScreen interface 7

2.2.2 Resource interface 8

role attribute 8
II Configuration

3 Murphy configuration

3.1 Zone definitions

3.2 Application Class definitions

3.3 Resource Class definitions

3.3.2 Predefined values

value types
4  PulseAudio configuration  17
  4.1  Zone definitions ................................. 17
    4.1.1  Fields ........................................... 17
      name .............................................. 17
    4.1.2  Resulting LUA globals .............................. 17
    4.1.3  Example .......................................... 17
  4.2  Routing Group definitions ............................... 17
    4.2.1  Fields ........................................... 18
      name .............................................. 18
      node_type .......................................... 18
      accept ............................................. 18
      compare ............................................. 18
    4.2.2  Predefined values .................................. 18
      Directions .......................................... 18
      Accept methods ....................................... 18
      Compare methods ....................................... 19
    4.2.3  Resulting LUA globals .............................. 19
    4.2.4  Example .......................................... 19
  4.3  Application Class definitions ....................... 19
    4.3.1  Fields ........................................... 19
      class ............................................... 19
      node_type .......................................... 20
      priority ............................................ 20
      route ................................................ 20
      binaries ............................................. 20
      roles ................................................. 21
    4.3.2  Predefined values .................................. 21
      node types .......................................... 21
    4.3.3  Resulting LUA globals .............................. 21
    4.3.4  Example .......................................... 22
  4.4  Audio Resource definition ............................ 22
    4.4.1  Fields ........................................... 22
      name ............................................... 22
      attributes .......................................... 22
    4.4.2  Example .......................................... 23
  4.5  Importing Murphy database tables ............. 23
    4.5.1  Example .......................................... 23
  4.6  Volume Limit definitions ........................... 23
    4.6.1  Example .......................................... 23
List of Figures

2.1 Components involved in audio stream management .............................................. 7
Part I

Murphy basics
Chapter 1

Media control

1.1 Audio playback and recording rights

1.1.1 Stream types

From policy perspective audio streams are classified as regular or interrupt streams.

Regular audio streams

Typical candidates for regular audio streams are the outputs of entertainment applications (e.g., radio, CD-player, MP3 players, Spotify etc), the voice streams of phone calls and sound streams of games. Based on policy rules Murphy orchestrates which of those audio streams can play and when.

Audio resources, i.e. the audio_playback and the audio_recording resource, are used to manage the actual playback rights. That means that for regular audio streams a resource set should exist in Murphy containing a corresponding audio_playback or audio_recording resource. An audio_playback or audio_recording resource in a resource set can control multiple (1+) audio streams. Typically an audio stream is controlled by just one audio resource.

Regular streams are supposed to play as long as Murphy grants the right for playback via the associated audio resource. Applications that produce regular streams should be prepared for the loss of playback rights and cope with the possible enforcement like getting corked or killed by the policy enforcement in PulseAudio.

Beside the audio streams policy aware application also manage the related audio resources, i.e. policy aware applications create resource sets, ask for playback rights and listen to the policy decisions Murphy makes. However, on behalf on policy unaware applications, that manage just the audio streams and do not deal with playback rights, Murphy can create the audio resources, if configured so.

Regular audio streams are typically last longer than few seconds and can tolerate small delays (< 10msec) caused by policy decision/enforcement.

Interrupt streams

Interrupt streams are short alerts or audio feedbacks that are intolerant for delays.

Interrupt streams are always played back immediately. Due to other policies they can be muted, however. Applications normally should not prepare for preemption, e.g. for getting corked or killed.

Applications that produce interrupt streams alone can be fully policy unaware, i.e. they do not need to deal with any audio resource.
1.1.2 Policy models for playback right

Enforced policy model

When the enforced policy model is used, Murphy informs the applications about granting or revoking their playback rights and in addition sets constraints on the audio streams that are not supposed to play anymore. This is done in order to avoid lousy end-user experiences caused by slowly responding or non-complying applications. Depending on the configuration Murphy might cork or mute audio streams in PulseAudio or in some cases even kill them.

The enforced policy model is also referred sometimes as strict-policy.

Cooperative policy model

When the cooperative policy model applies, Murphy just informs the applications whether their playback rights are granted or revoked and expects that the applications will shortly comply with the decision. Beside the notification no further action is taken and no constraints set.

The cooperative policy model is also referred as relaxed-policy.

1.1.3 Playback rights management

Murphy uses resources and resource sets to manage playback rights. Two resources are defined, one for audio playback and another for audio recording.

To manage audio stream playback rights audio resources should be created and acquired. Streams can be active only if the associated resource usage were granted (see also Policy models for playback right above).

An audio resource can represent the playback right of one or more stream. As a special case multiple resource set might control a single stream. In the latter case the stream is allowed to play as long as the usage of at least one of the associated resources is granted. The usage of multiple resource controlling a single stream is discouraged as it brings several limitations to the actual policies.

Policy aware applications

Policy aware applications create themselves the audio resources/resource sets and acquire them when they wish to play. In addition policy aware applications are supposed to set stream properties in harmony with resource attributes to help Murphy and pulseaudio-module-murphy-ivi (module inside PulseAudio) establish the relationship between resources and streams.

Policy aware applications supposed to either

- generate a system-wide unique id and assign it to both to the resource.set.name stream property and to the name resource attribute
- set the resource.set.id property of the media stream to the value of resource set ID.

Policy unaware applications

On behalf of policy unaware applications pulseaudio-module-murphy-ivi creates the audio resources, if configured so. In other words policy unaware applications just deal with the media streams but do not interface Murphy for playback right management. Such applications should cope with the consequences of policy enforcement such as corking or stream killing.
1.2 Volume control

Volume control means limiting the volume of the streams with certain built-in rules or rules defined in lua configuration. For instance there are built-in methods:

1. for generic volume level setting for all streams on specified outputs (for example speed dependent volume)
2. for suppressing volumes of other streams in case a certain stream is present (for example navigator suppressing underlying music stream)

There’s also possibility to define maximum limitation for stream classes allowing to do different level adjustments for different classes. It can be defined, for example, that navigator streams will always attenuate other streams by -20 dB, but phone is never attenuated more than -10dB. This means that others than phone will be attenuated -20dB and phone is always limited to maximum -10dB attenuation.

If more complex volume limiting is needed lua functions can be added to the configuration file. Such functions are called by appearing and disappearing streams, routing or external events.

1.3 Routing

pulseaudio-module-murphy-ivi is built around the concept of explicit and implicit routing.

1.3.1 Explicit routing

Explicit routing means that the audio application explicitly sets the routing target for its audio stream. The routing target is typically set at stream creation time by setting the target sink or source via the pulseaudio client interface.

Explicit routes are taking precedence over the implicit ones, i.e. Murphy never tries to route the stream anywhere else than the explicit routing target. If the explicit routing targets were temporarily or permanently inaccessible, Murphy will leave the stream alone; the usual pulseaudio mechanisms will kick in and the stream will be re-routed to NULL sink or it will be killed.

1.3.2 Implicit routing

Implicit routing takes places if the streaming application did not set the routing target for a stream: In such case pulseaudio-module-murphy-ivi will try to find the routing target for the stream by:

1. categorizing the stream into a certain application class,
2. determining the audio zone for the stream,
3. applying the routing rules of a routing group defined by the application class and the audio zone.

A routing group is a priority ordered list of routing targets where streams can be automatically routed. The routing target of a routing group is always the highest priority accessible target from the list. For example if a USB target is appearing in or a Bluetooth target disappearing from a routing group the routing target of the group might change and the streams that belong to the group might be re-routed to the new target.

Streams are always assigned to a single routing group, while sinks and sources (i.e. routing targets) can belong to multiple routing groups. Sinks and sources that belong to multiple routing groups might have different priorities in the different routing groups.

If an application sets an explicit routing target (sink or source) for an implicitly routed stream the routing scheme of the stream will be changed to explicit.
1.3.3 Configuration

The first step is to decide whether

1. the streaming application is responsible for routing by setting explicit routing targets for its streams or
2. the routing decision is left for pulseaudio-module-murphy-ivi.

In the later case, i.e. when implicit routing is used, certain boundary conditions have to be fulfilled and pulseaudio-module-murphy-ivi has to be able

1. to classify the audio stream into an application class and
2. to determine the audio zone for the stream

There are currently couple of mechanism to do this:

1. Setting properties on the audio stream when creating the stream
2. Defining rules in the configuration file of pulseaudio-module-murphy-ivi that would apply if those properties were not set
3. Setting environment variables (this feature is still in experimental stage and to be released)

Setting properties on the audio stream

Application might set the following properties on a media stream

- **media.role** - that is mapped to an application class
- **zone.name** - that species the target audio zone for the stream

**gststream example:**

```bash
gst-launch-1.0 playbin uri=file:///test.mp3 audio-sink="pulsesink stream-properties="props,←
media.role=music,zone.name=passanger1"
```

In the above example the stream properties tell to pulseaudio-module-murphy-ivi that the stream is a music stream (**media.role=music**) and the stream is destined to **passanger1** zone. There are definitions in murphy-ivi.lua configuration file that map **media.role=music** to application class **player** and define a routing group **passenger1** zone.

Some hardwired fallback value is used in case a property was not set or there were no matching definition for the property value. The hardwired default value is **player** for application class and **driver** for audio zone.

Defining rules in the configuration file of pulseaudio-module-murphy-ivi

Media roles

The media roles are mapped to application classes in the configuration file of pulseaudio-module-murphy-ivi murphy-ivi.lua.

The mapping is 1:n i.e. many media role can map to the same application class but a media role always belongs to a single application class. Consequently, the mapping is defined in such a way that for each application class the relevant list of media roles are specified. Adding the same media role to multiple application classes will result in parsing errors.

Binary mapping

Application classes can also be determined by the **binary name** that is stored in the stream property **application.process.binary**.

The **binary name** is the basename of the application executable. In the configuration file of pulseaudio-module-murphy-ivi this binary name can be bound to an application class. This mechanism is intended to classify legacy applications that don’t otherwise set the properties needed to classify audio streams.

Binary mapping might be tricky for application servers, like crosswalk, where the binary name is usually the application server or launcher name and not the “real” application name.
Environment variables for zone routing

We are experimenting with a feature, where you can set the routing zone as an environment variable and we could read that and the stream would be routed to different zone (and hence possibly to different sound card if defined so)
Chapter 2

Architecture

2.1 Components

2.2 Interfaces

2.2.1 HomeScreen interface

This interface is used to manage resources. It is an alternative interface to the native resource management interfaces of Murphy.
2.2.2 Resource interface

This interface is used by policy aware applications and pulseaudio-module-murphy-ivi to create the actual audio resources and resource sets in Murphy.

The audio resources have the following mandatory attributes:

role attribute

Must be set to same as the media.role property of the corresponding media stream in PulseAudio. Currently it is not used. If not set it defaults to “music”.

pid attribute

Refers to the process that actually creates the media stream in PulseAudio. Please note that the audio resource and the media stream can be managed by different processes in some cases. The pid attribute is the string representation of the actual pid value.

policy attribute

Determines the policy model to apply and can be one of the string “strict” or “relaxed” to set the policy model enforced or cooperative respectively. If not set defaults to “relaxed”.

name attribute

A system-wide unique identifier to associate resources and audio streams. The same value, what is stored in this attribute, must be set for the resource.set.name property of the audio stream(s) that is (are) controlled by the resource.

2.2.3 Pulseaudio interface

Native applications and the WebRuntime (i.e. crosswalk) use the usual PulseAudio interfaces to manage their audio streams, i.e. they can use the native or simple interfaces over Unix domain sockets or TCP.

The following stream properties can be set to influence policies:

media.role property

One of the roles defined in /etc/pulse/murphy-ivi.lua. Typically “music”, “video”, “navigator” and “phone” are defined. media.roles are mapped in the configuration file to resource classes which in turn are used to determine playback rights. media.roles are also used to apply volume constrains, e.g. by the volume.suppress constrain.

resource.set.id property

Identifies the resource set that contains the relevant audio-playback or audio_recording resource. When an application creates a resource set using one of the resource API’s it gets back a system wide unique resource ID. The resource.set.id property is supposed to carry that resource ID as a string.

Alternatively, if the resource ID was unknown, resource.set.id property can be set as “pid” in which case the audio stream will be associated with an audio resource having a matching pid. This method might be useful if the audio stream and the resource set is managed by different processes and for some reason it is difficult to pass the resource set id to the renderer stream. Setting the resource.set.id to “pid” is not recommended. Use the resource.set.name instead.

The usage of resource.set.id and resource.set.name is mutually exclusive.
resource.set.name property

A system-wide unique name can be used to link audio resources to audio streams. If the value of name resource attribute matches the value of resource.set.name stream property the resource set, that contains the audio resource, and audio stream is linked together, the resource set will control the playback of the audio stream.

Audio resources and streams can be freely linked with the exception of linking multiple resources to multiple streams. In other words a single resource set can control one or more streams, or multiple resource sets can control a single stream.

However, linking multiple resources to a single stream might be tricky for regular applications, it is implemented to support crosswalk where the streams of multiple <audio> tags are mixed inside crosswalk while each of them have a separate resource set. In such setup Murphy can control the playback rights individually but all streams are routed to the same target.

2.2.4 Murphy internal interface

pulseaudio-module-murphy-ivi subscribes changes in audio_playback_users and audio_recording_users table in Murphy database. Consequently, after new policy decisions are made for each altered resource set the following fields are delivered to pulseaudio-module-murphy-ivi:

rsetid field

that is the resource set ID that contains the audio resource.

autorel field

true if the application sets auto-release flag of the resource set. If a resource set loses all of its resources and the auto-release flag is set the resource set state changes automatically from acquire to release meaning that the playback will not resumed after the blocking condition goes away. In addition if the policy attribute of the audio_playback resource were set to “strict” the playback stream will be killed.

state field

is either acquire or release. Resource sets in release state would never get any grant for resource usage.

grant field

shows whether the audio playback/recording is allowed or not (0 = forbidden, 1 = allowed).

pid field

contains the process ID of the renderer, i.e. the process that supposed to create the media stream. It is used to associate a resource set with a media stream if the resource.set.id property of the media stream was set to “pid”. The source of the pid field is the pid attribute of the resource.

policy field

is either “relaxed” or “strict” and refers to the policy model to apply. If an audio resource with “strict” policy looses the playback rights the associated media stream will be corked or muted. If the autorel field was set, the stream gets killed.

However, when the policy field is set to “relaxed”, loss of playback rights will not cause any corking, muting or killing. The source of the policy field is the policy attribute in the resource.
name field

It contains the value of the name resource attribute i.e. a system-wide unique id. It is used in pulseaudio-module-murphy-ivi to link resource sets to streams as described above.

2.3 Data flow

Applications use the Pulseaudio interface to manage their media streams. To manage playback rights for regular audio streams some entity should create/maintain suitable resource sets. The entity, that creates and maintains the resource set can be the application itself in which case the application is a policy aware application. However, pulseaudio-module-murphy-ivi also can create resource sets on behalf of policy unaware applications if configured so.

In order to apply playback rights related policies, pulseaudio-module-murphy-ivi needs to maintain the relationship between resource sets and regular audio streams, i.e. what audio stream belong to what resource set. Playback rights are communicated over Murphy internal interface accompanied by the ID of the managing resource set as well as the PID of the rendering process and a system-wide unique name. On the other hand policy aware applications supposed to set the resource.set.id or resource.set.name properties on their media streams. Thus pulseaudio-module-murphy-ivi can establish the relationship between usage rights and media streams using either

- the resource set ID or
- a system-wide unique name that is stored in a resource attribute as well as in a stream property,
- the process ID that is stored in a resource attribute as well as in the application.process.id stream property (set by PulseAudio automatically).

Policy aware native applications

Policy aware applications use either one of the Murphy resource APIs or the ico-API1 to manage their audio playback rights. In addition, policy aware applications supposed to set the resource.set.id property to “pid” on their audio streams. The pid and name attributes of the audio resources along with the playback rights are passed to pulseaudio-module-murphy-ivi over the Murphy internal interface. pulseaudio-module-murphy-ivi in turn enforces the playback rights based on matching resource set IDs, system-wide unique names or the pid of the rendering process.

Policy aware native applications are required to set on their media streams the media.role property along with either the resource.set.name or the resource.set.id property. Optionally they can also set zone.name stream property.

Policy unaware native applications

Policy unaware applications create and manage their PulseAudio streams the usual way and might set the media.role property on them. However, policy unaware applications do not create any audio resources by themselves and consequently shall not set neither the resource.set.id nor the resource.set.name property on their streams. When pulseaudio-module-murphy-ivi detects a newly created stream it looks up a resource set definition for the stream from /etc/pulse/murphy-ivi.lua configuration file. If it founds one, it creates the specified resource set over the Resource Interface.

Policy unaware native applications are not expected to set any properties on their media stream. However, they can set the standard media.role property if they wish to do so.

Policy aware web applications

Crosswalk, that is the rendering engine of the web applications, is policy aware and <audio> and <video> tags will produce resource sets beside the usual audio streams. The resource sets are created via the native Murphy resource interfaces. Crosswalk will automatically generate a system-wide unique id and store it the resource.set.name property on its audio streams as well as store it in the name attribute of the audio resources. Crosswalk apps are also setting the media.role property, which can be defined in the manifest file of the web app:
The same value also used to define the **application class** when creating the resource set through the native **Murphy** resource interface. These concepts, **media.role** and **application class** of the resource set, are closely related but not required to be necessarily identical. It is highly advised that **application class** that is used in **pulseaudio-module-murphy-ivi** matches the one that is used in the resource set managed by Murphy. As described above **pulseaudio-module-murphy-ivi** uses the mapping definitions in the configuration file (eg. **murphy-ivi.lua**) to determine the application class.

At this point it is not possible to define the media roles for audio tags and consequently all audio tags from a single application will have one role, which is the same as application class through resource interface. So all audio tags will follow same policy rules defined for that class. If the playback right for one audio tag is granted, others are also allowed to play. If one audio tag is stopped, all tags within same applications are stopped.

Currently **Crosswalk** is creating an audio resource for every audio tag in HTML, but it will mix all the streams from these tags inside the web engine resulting in a single stream sent to pulseaudio. This means that multiple resource sets control a single stream in **Pulseaudio**.
Part II

Configuration
Chapter 3

Murphy configuration

In this chapter we discuss Murphy configuration from audio management perspective. Accordingly, this chapter does not give a full description of Murphy configuration, only those definitions and fields are presented here that are directly involved in the audio management configuration or have some effect on it.

3.1 Zone definitions

Zone definitions define resource conflict zones. The zone definitions must be a consistent super set of the zone definitions for pulseaudio-module-murphy-ivi. That will guarantee that the resource conflicts are resolved on entertainment zone basis.

3.1.1 Fields

name

The name field is used to assign a unique identifier to the zone. The name field is mandatory.

attributes

Defines the zone attributes in a key/value pair format. Currently it does not play any role in audio management.

3.2 Application Class definitions

Application Class definitions must be a consistent super set of the Application Class definitions for pulseaudio-module-murphy-ivi.

3.2.1 Fields

name

The name field is specifying a unique symbolic name for the application class. Mandatory field.
priority

Sets the priority for the Application Class. Higher values represent higher priorities. Application Class Priorities are used at resource conflict resolution, i.e. resource usage requests inherit the priority of the application that creates them. The requests are granted in reverse priority order, i.e. higher priority requests have precedence over lower priority requests. Priorities play important role in playback right management. However Application Class Priorities defined for Murphy have no effect on routing and volume control.

modal

Deprecated field that should be set to false.
Mandatory field.

share

Field that defines whether application that belong to the Application Class will share their resources or not. Currently not in use.
Mandatory field.

order

Specifies the request serving order within an Application Class. The possible values for the order field are “lifo” and “fifo”.
If order is set to “lifo” a newer request will have higher priority than an older request of the same Application Class. This might be used for e.g. in media playback to guarantee that recent user playback requests will preempt older ones if the resource usage were defined as “exclusive”. If the playback is finished or the user stops the playback the preempted media stream will continue.
If order is set to “fifo” the newer request will have the lowest priority in the same Application Class. This can be used e.g. in queuing notifications and playing back them in the resource acquisition order if the resource usage were defined “exclusive”.
Mandatory field.

3.3 Resource Class definitions

3.3.1 Fields

name

The name field is specifying a unique symbolic name for the Resource Class. The value of this field is also used to derive the names of the related resource management database tables. For instance, for every Resource Class a database table is created to store information for each resource requests of that particular Resource Class. The database table name is composed as follows:

```
<name>_users
```

It is important that the audio resources defined for pulseaudio-module-murphy-ivi have their matching counterpart definition in Murphy. Resource names for audio playback and recording are defined in the audio_resource section of the configuration file of pulseaudio-module-murphy-ivi. Resources with the same name should be defined in the Murphy config file.

The resource attributes, that pulseaudio-module-murphy-ivi uses when creates audio resources on behalf of policy unaware applications and the attributes in the resource.class sections in the Murphy config files should be consistent. More precisely the Murphy attributes should be a super set of the attributes used in Pulseaudio.

Example definition for pulseaudio-module-murphy-ivi:
The corresponding *Murphy* definitions for the same example:

```plaintext
resource.class {
    name = "audio_playback",
    shareable = true,
    attributes = {
        role = { mdb.string, "music", "rw" },
        pid = { mdb.string, "<unknown>" },
        policy = { mdb.string, "strict", "rw" },
        name = { mdb.string, "<unknown>" } "rw"
    }
}
resource.class {
    name = "audio_recording",
    shareable = true,
    attributes = {
        role = { mdb.string, "music", "rw" },
        pid = {mdb.string, "<unknown>" },
        policy = {mdb.string, "strict", "rw" },
        name = {mdb.string, "<unknown>" } "rw"
    }
}
```

The *name* field is mandatory.

**shareable**

Specifies whether the *Resource Class* is shareable in general. The effective sharing is calculated as the logical AND of this field and the shareable field of the actual resource instance (the former is defined by this field and latter is defined by the application over the resource APIs).

If *shareable* was set to *false* the resource will effectively not be shared regardless what the application says at resource definition time. If *shareable* was set to *true* the effective sharing will be determined by application that created the resource instance via the resource creation API.

The application with the highest priority resource usage grant owns the resource. If the effective sharing is *false*, the resource usage will be exclusively granted to the owner application. If the effective sharing is *true*, the resource usage might be granted for other applications in the same or lower priority Application Classes. The effective sharing will also be calculated for the subsequent grants and the resource remains shareable as long as the effective sharing remains *true*. In other words the resource remains shareable as long as an Application Class or the application itself does not want to share it any more.

The *exclusive* use of a resource means that it is not shared with lower priority requestors. However, depending on the actual scenario, higher priority requestors might also use it.

The *shareable* field is mandatory.

**attributes**

Specifies the list of attributes. The value of this field is a table composed of the entries as follows:

```plaintext
<attr_name> = ( <value_type>, <default_value>, <accessibility> )
```
where

- `<attr_name>` is the actual name of the attribute
- `<value_type>` is the data type of the attribute. One of the predefined `value types` below.
- `<default_value>` is the value that will be used if the attribute would not be set by the application. The format of the value must be consistent with the format of the value type, e.g. if `<value_type>` were `mdb.string` the `<default_value>` must be a quoted string.
- `<accessibility>` in practice it should be set to “rw”.

See example for attribute definition in the `name` section above.

The `attributes` field is mandatory.

### 3.3.2 Predefined values

**value types**

- `mdb.string`
- `mdb.integer`
- `mdb.unsigned`
- `mdb.floating`
Chapter 4

PulseAudio configuration

4.1 Zone definitions

Zone definitions are used to define the entertainment zones within a vehicle. The zone definitions should be consistent with the zone definitions in Murphy.

4.1.1 Fields

name

The name field is used to assign a unique identifier to the zone. The name field is mandatory.

4.1.2 Resulting LUA globals

Zone definitions are automatically assigned to global variables and can be accessed as zone.xxxxx where xxxxx is the name of the Zone. For instance the definition of the following

```lua
zone { name = "driver" }
```

can be accessed from LUA as zone.driver.

4.1.3 Example

```lua
zone { name = "driver" }
zone { name = "passenger1" }
```

4.2 Routing Group definitions

Routing groups are used for implicit routing only. Routing groups are a prioritized list of nodes of possible routing targets. All the list member nodes should have the same direction, i.e. either output or input. Two functions are used to maintain the list of target nodes. First is a filter function to decide what nodes are accepted in the routing group and a second one is a compare function to determine the order of the list, i.e. sort it.

Due to physical or other constrains target nodes can temporarily become unavailable. Implicit routing in practice means to figure out what routing group to be used and pick the highest priority available target in the routing group.
4.2.1 Fields

name

The *name* field is used to assign a unique identifier to the routing group. The *name* field is mandatory.

node_type

defines the direction of the nodes that are accepted by the routing group. It determines whether the routing group is for input or output routing. The numeric node type can be one of the predefined values *node.input* or *node.output*. The *node_type* field is mandatory.

accept

is a filter function that is used to determine what node is part of the routing group. The *accept* function can be one of the predefined methods or a LUA function as well. The signature of the LUA function is as follows:

```lua
function accept(self, node_to_accept)
    -- Implementation goes here
end
```

where

- *self* is the *Routing Group* object (i.e. the table that defines the *Routing Group*)
- *node_to_accept* is the node object to be accepted/rejected

The *accept* function should return a boolean value, i.e. a *true* if the *node_to_accept* should be inserted into the target list.

Do not use *node*, or other globally defined identifiers, as argument name since it may lead to confusion/errors. The *accept* field is mandatory.

compare

function is used when sorting the list of targets. The *compare* function can be one of the predefined methods or a LUA function:

```lua
function compare(node1, node2)
    -- Implementation goes here
end
```

The *compare* field is mandatory.

4.2.2 Predefined values

Directions

- *node.input*
- *node.output*

Accept methods

- *builtin.method.accept_default* for the default routing group
- *builtin.method.accept_phone* for call routing
Compare methods

- `builtin.method.compare_default`
- `builtin.method.compare_phone` for call routing

### 4.2.3 Resulting LUA globals

Routing Group definitions are automatically assigned to global variables and can be accessed as `routing_group.xxxx_dddd` where `xxxx` is the name of the Routing Group and `dddd` is the direction. For instance the definition of the following

```lua
routing_group {
    name = "phone",
    node_type = node.input,
    ...
}
```

can be accessed from LUA as `routing_group.phone_input`.

### 4.2.4 Example

```lua
routing_group {
    name = "default_driver",
    node_type = node.output,
    accept = function(self, n)
        return (n.type ~= node.bluetooth_carkit and n.type ~= node.hdmi)
    end,
    compare = builtin.method.compare_default
}

routing_group {
    name = "phone",
    node_type = node.input,
    accept = builtin.method.accept_phone,
    compare = builtin.method.compare_phone
}

routing_group {
    name = "phone",
    node_type = node.output,
    accept = builtin.method.accept_phone,
    compare = builtin.method.compare_phone
}
```

### 4.3 Application Class definitions

Application Class definitions are used to define the Application Class itself i.e. what streams belong to the Application Class and what Routing Groups should be used for implicit routing.

#### 4.3.1 Fields

**class**

The `class` field holds the name of the resource class if a resource set was created for a matching executable or media.role (see the binaries and roles fields for details).

The `class` field is optional.
**node_type**

The `node_type` field specifies the type of the node created for media streams of `Application Class`. For classification rules see below the sections `binaries` and `roles`.

The `node_type` field accepts numeric values. Usually one of the predefined constants is used to specify the `node_type`, e.g. `node.navigator`.

The `node_type` field is mandatory.

**priority**

The `priority` field defines the priority of the application class. Currently it is not used. However, it is mandatory.

**route**

The `route` field defines the rules for implicit media stream routing. The routing rules are defined by a table with input sub fields:

- **input** sub field: a table that has a per zone entry in the form of `zone-name = routing-group-definition`. Typically the LUA globals created by `Routing Group definitions` are used as `routing-group-definition`, e.g. `routing_group.name`.

- **output** sub field: a table with identical syntax to the `input` sub field.

The routing table to be used for implicit routing is looked up based on the stream direction (i.e. `input` or `output`) and the `zone.name` stream property.

The `route` field is mandatory. The `input` and `output` sub fields are optional. However, at least one of them needs to be present.

**binaries**

The `binaries` field is used to define rules to classify streams into `Application Classes` based on their `media.role` property or the executable name. These rules also determine the type of the related stream nodes, control whether to override the `media.role` stream property and define the audio resources to create on behalf of policy unaware executables, if at all.

The rules are defined by a table in terms of a list of `key/value` pairs. The `key` is either the id of a webruntime (i.e. `crosswalk`) application or the `base name` of an executable, e.g. `pacat`. From LUA perspective a `key/value` pair can be defined either as `key = value` or `['key'] = value`. However, if the first form is used `key` must be a valid LUA identifier and therefore have more restrictions than file names have. For instance `tjzf32324.MediaPlayer` is a valid file name but invalid LUA identifier because of the dot in the middle. Therefore the bracket form should be used where the `['tjzf32324.MediaPlayer'] = foobar` is valid.

In case a binary rule matches the definitions of the `application class` that hosts the rule will apply:

- the node, that is created in `pulseaudio-module-murphy-ivi` for the stream, will have the type that is defined by `node_type` section of the application class definition,

- the routing queue that is used for implicit routing is defined in the `route` section of the application class.

Based on the `value` of a matching binary the following rules apply:

- If `value` is `no_resource`, no resources/resource-sets will be created by `pulseaudio-module-murphy-ivi` for the streams of the executable.

- If `value` is a string (e.g. “music”) the `media.role` property will be set to `value` (e.g. `media.role = “music”`) on the streams of the executable.
• If value is a table, containing a list of fields, audio resources will be created on behalf of executable. The first value in the table must be a numeric priority value followed by an arbitrary set of the “autorelease”, “mandatory”, “shared”, “optional” and “exclusive” strings that describe the corresponding audio resource or resource-set flags. Strings, other than the listed flag names, are treated as override values for the media.role property of the stream. For instance if value were [0, “autorelease”, “mandatory”, “shared”, “navigator”] a resource-set containing a single audio_playback resource will be created for the sink-inputs/source-outputs of the executable. The resource-set will have zero relative priority and the autorelease flag will be set. The audio_playback resource will have the mandatory and shared flags on. In addition the media.role property on the streams of the executable will be forcibly set to “navigator” (possibly overriding the media.role set by the application).

The binaries field is optional.

roles

The roles field is used to define rules to classify streams with matching media.role property. The rules will determine the type of the nodes that are created for the matching streams. Optionally audio resources/resource-sets can be created for the matching streams.

The actual rules are defined by a table that contains key/value pairs. The key is the value of the media.role property to match. The value can be either either no_resource or a table similar to the the ones described above in the binaries section.

Rules defined by the binaries field have higher priorities than the ones defined by roles, i.e. if both a binaries rule and a roles rule is applicable the binaries rule will be used.

The roles field is optional.

4.3.2 Predefined values

node types

• node.radio
• node.player
• node.navigator
• node.game
• node.browser
• node.camera
• node.phone
• node.alert
• node.event
• node.system

4.3.3 Resulting LUA globals

Application Class definitions are automatically assigned to global variables and can be accessed as application_class.xxxx where xxxx is the node type. For instance the definition of the following

```lua
application_class { 
  ...
  node_type = node.phone,
  ...
}
```

can be accessed from LUA as application_class.phone.
4.3.4 Example

```plaintext
class = "navigator",
node_type = node.navigator,
priority = 3,
route = {
    output = { driver = routing_group.default_driver_output,
        passenger1 = routing_group.default_passenger1_output }
},
roles = {
    navigator = {0, "autorelease", "mandatory", "shared"}
},
binaries = {
    ['net.zmap.navi'] = { 0, "autorelease", "mandatory", "shared" } 
}
```

4.4 Audio Resource definition

4.4.1 Fields

name

Specifies the resource names for managing audio playback and recording rights on behalf of policy unaware applications:

```plaintext
name = { recording = "<name_of_audio_recording_resource>", playback = "< →
    name_of_audio_playback_resource>" }
```

where

- `<name_of_audio_recording_resource>` is the name of the resource to be used to manage audio recording rights
- `<name_of_audio_playback_resource>` is the name of the resource to be used to manage audio playback rights

Note, that these names must match the `name` field of a suitable Resource Class definition in the Murphy configuration file (e.g. `murphy.lua`)

`name` is mandatory field.

attributes

Specifies the attributes for the resources created by `pulseaudio-module-murphy-ivi` on behalf of policy unaware applications. The value of this field is a table that has a definition entry for each attribute as follows:

```plaintext
<attribute_name> = {<stream_property>, <attribute_type>, <default_value> }
```

where

- `<attribute_name>` is the name of the attribute. Must match an attribute name in the corresponding Resource Class definition in Murphy.
- `<stream_property>` specifies a Pulseaudio stream property. The value of this property is copied to the resource attribute when the audio resource is created for the stream.
- `<default_value>` specifies the resource attribute value if `<property>` were not set on the audio stream.

`attributes` is mandatory field.
4.4.2 Example

```yaml
daudio_resource {
    name = { recording = "audio_recording", playback = "audio_playback" },
    attributes = {
        role = {"media.role",mdb.string,"music"},
        pid = {"application.process.id",mdb.string,"<unknown>"},
        name = {"resource.set.name",mdb.string,"<unknown>"}
    }
}
```

4.5 Importing *Murphy* database tables

4.5.1 Example

```yaml
mdb.import {
    table = "speedvol",
    columns = {"value"},
    condition = "zone = 'driver' AND device = 'speaker'",
    maxrow = 1,
    update = builtin.method.make_volumes
}
```

4.6 Volume Limit definitions

4.6.1 Example

```yaml
volume_limit {
    name = "navi_suppress",
    type = volume_limit.class,
    limit = -20,
    node_type = {node.navigator, node.phone},
    calculate = builtin.method.volume_suppress
}
```