Mir

Release v2.18.3

Canonical Group Ltd.

Oct 10, 2024

CONTENTS

Mir is a next generation display server.

It runs on a range of Linux powered devices including traditional desktops, IoT and embedded products. Mir is a replacement for the X window server system, commonly used on Linux desktop devices. It allows device makers and desktop users to have a well-defined, efficient, flexible, and secure platform for their graphical environment.

CHAPTER

ONE

IN THIS DOCUMENTATION

Tutorials

Start here - an introduction to what Mir can do and how to use it

How-to guides

Step-by-step guides covering key operations and common tasks

Reference

Technical information - specifications, APIs, architecture

Explanation

Discussion and clarification of key topics

CHAPTER

TWO

PROJECT AND COMMUNITY

Mir is a member of the Ubuntu family. It's an open source project that warmly welcomes community projects, contributions, suggestions, fixes and constructive feedback.

- Code of conduct
- Get support
- Join our online chat
- Contribute
- Thinking about using Mir for your next project? Get in touch!

2.1 Tutorials

These pages provide first-time introductions to key Mir aspects

- *Learn What Mir Can Do*: A showcase of Mir's capabilities
- *Write Your First Wayland Compositor*: A guide through writing a simple compositor

2.1.1 Learn What Mir Can Do

This tutorial will guide you through using some example programs that showcase Mir's capabilities in different setups. You will see how Mir supports a range of capabilities that can be used by Mir based compositors. Mir makes it easy to write compositors as shown in the *developer tutorial*.

Installation

Mir demos are available on Debian derivatives, Fedora, and Alpine. For distros that don't have prebuilt binaries, examples can be built from source.

To install Mir demos on Debian and its derivatives:

```
sudo apt install mir-demos mir-graphics-drivers-desktop
```

Installing Mir demos on Fedora

```
sudo dnf install mir-demos
```

Installing Mir demos on Alpine

```
sudo apk add mir-demos mir
```

Running

The main script you'll want to play around with is `miral-app`. It runs a shell with some eyecandy by default, but you can run it in kiosk mode if that fits your usecase more.

For the uninitiated: The shell mode runs a floating window manager just like GNOME and such where you can move windows and maximize or minimize them. Kiosk mode on the other hand assumes you want one (or more) applications in fullscreen mode all the time.

To run in shell mode, just run:

```
miral-app
```

for kiosk mode:

```
miral-app -kiosk
```

Note: By default, Mir does not enable Wayland extensions that normal applications should not be using. For demonstration purposes we will override this and allow all supported extensions in some of the following examples by passing `--add-wayland-extensions all` when running the example.

Running Natively

The previous section showed how you can run Mir demos under an X11 or Wayland session. But Mir compositors can also run "natively" by launching them from a virtual terminal or a greeter.

Launching from a virtual terminal

To switch to a virtual terminal, you can press CTRL+ALT+F<Number>. You can then log in and run:

```
miral-app
```

Launching from a greeter

You can also launch `miral-shell` from your greeter by opening the window manager list (bottom right cog on Ubuntu) and choosing "Miral Shell". Once you log in, you'll see `miral-shell` running fullscreen!

Using On-screen Keyboards

Mir based compositors can easily support on-screen keyboards. Note that due to security reasons, the Wayland extensions needed are disabled by default.

For the purpose of demonstation, we'll use `ubuntu-frame-osk`, but you're free to use any Wayland compatible on-screen keyboard.

You can install `ubuntu-frame-osk` by running:

```
sudo snap install ubuntu-frame-osk
sudo snap connect ubuntu-frame-osk:wayland
```

To test your favourite on-screen keyboard, you can start `miral-app` as follows:

```
miral-app --add-wayland-extensions all
```

Once the shell loads, you can start the terminal, then run:

```
ubuntu-frame-osk&
```

And an on-screen keyboard should pop up!

Mixing Wayland and X11 Clients

You can easily run X11 applications inside Mir based compositors. For example, to enable x11 support in `miral-shell`, you can run the following command:

```
miral-app --enable-x11 true
```

When it loads, you can start a terminal and run `xclock` or any other X11 application alongside Wayland applications!

Remote Desktop

Mir also supports remote desktops via the VNC protocol. To demo this, we'll use `wayvnc`. You can install it by running:

```
sudo apt install wayvnc
```

Start the shell with all extensions enabled:

```
miral-app --add-wayland-extensions all
```

From inside, you can run the terminal and run `wayvnc`:

```
wayvnc
```

This should start `wayvnc` and make it listen to `localhost`. To connect, run your favourite VNC viewer and connect to `localhost`. You should see the exact same view in both Mir compositor and the VNC viewer. For example:

```
gvncviewer localhost
```

2.1.2 Write Your First Wayland Compositor

This tutorial will guide you through writing a simple compositor: the installation of dependencies, building it, and running it. You'll be impressed by how easy it is to create a Wayland compositor using Mir!

Assumptions

This tutorial assumes that:

1. The reader is familiar with C++ and CMake.
2. The reader has cmake and a C++ compiler installed

A barebones Mir compositor

This section will cover the needed dependencies and how to install them, the minimum code needed for a Mir compositor, how to build this code, and finally how to run your compositor.

Dependencies

Before you start coding, you'll need to install `libmiral`, and `mir-graphics-drivers-desktop` which can be done on different distros as follows:

Installing dependencies on Debian and its derivatives:

```
sudo apt install libmiral-dev mir-graphics-drivers-desktop
```

Installing dependencies on Fedora

```
sudo dnf install mir-devel libxkbcommon
```

Installing dependencies on Alpine

```
sudo apk add mir-dev
```

Code

The following code block is the bare minimum you need to run a Mir compositor:

```
#include <miral/runner.h>
#include <miral/minimal_window_manager.h>
#include <miral/set_window_management_policy.h>

using namespace miral;

int main(int argc, char const* argv[])
{
    MirRunner runner{argc, argv};

    return runner.run_with(
        {
            set_window_management_policy<MinimalWindowManager>()
        });
}
```

This program creates a floating window manager with basic window management capabilities such as controlling multiple windows, minimizing and maximizing, and handling mouse input. This is done with the help of MirAL (Mir Abstraction Layer) which gives you a high level interface to work with Mir.

Building

To compile this simple program, you can use this `CMakeLists.txt` file:

```
cmake_minimum_required(VERSION 3.5)

project(demo-mir-compositor)

set(CMAKE_CXX_STANDARD 23)

include(FindPkgConfig)
pkg_check_modules(MIRAL miral REQUIRED)
pkg_check_modules(XKBCOMMON xkbcommon REQUIRED)

add_executable(demo-mir-compositor main.cpp)

target_include_directories(demo-mir-compositor PUBLIC SYSTEM ${MIRAL_INCLUDE_DIRS})
target_link_libraries(     demo-mir-compositor               ${MIRAL_LDFLAGS})
target_link_libraries(     demo-mir-compositor               ${XKBCOMMON_LIBRARIES})
```

Then to build:

```
cmake -B build
cmake --build build
```

Running

To run, you can run nested in an X or Wayland session, or from a virtual terminal, just like the demo applications in *Learn What Mir Can Do*. For example, if we were to run inside an existing Wayland session, we'd run the following command:

```
WAYLAND_DISPLAY=wayland-99 ./build/demo-mir-compositor
```

You'll see a window housing the compositor with a black void filling it. To fill this void with some content, you can run the following from another terminal:

```
WAYLAND_DISPLAY=wayland-99 bomber
```

You're free to replace `bomber` with any other Wayland compatible application.

Next steps

Now that you have your base compositor working, feel free to check out these guides on how to further develop your compositor:

- *How to specify start up applications*
- *How to handle user input*
- *Specifying CSD/SSD Preference*

2.2 How-to guides

These how-to guides cover the key aspects of working with Mir.

- *Developing a Compositor*: build your own, custom compositor with Mir
- *Developing protocol extensions for Mir*: extend your compositor with support for a custom protocol extension
- *Getting involved in Mir*: first steps to contributing to Mir
- *Using checkbox-mir*: validating your environment using checkbox-mir
- *Calibrating a touchscreen device*: creating and using a calibration matrix using the libinput snap
- *Enabling graphics-core22 on a device*: creating a `graphics-core22` provider snap for a new board
- *How to specify start up applications*
- *How to handle user input*
- *Specifying CSD/SSD Preference*

2.2.1 Developing a Wayland Compositor using Mir

Overview

The Mir project provides libraries for creating Wayland compositors. The design is intended to make it easy build something simple and easy to customize the compositor in a number of ways.

Within the Mir codebase there are four different Wayland compositors each demonstrating a different behaviour. There are additional compositors maintained and written by the Mir team in a number of Snaps. Most notably, Ubuntu Frame.

Outside the Mir codebase there are at least four compositors based on Mir that support different uses.

Here's the full list, linking to the code:

Compositor	Description
Miriway	A minimal desktop shell
Lomiri	The shell used by Ubuntu Touch, aspires to be "convergent": that is work across various form factors (including desktop)
Ubuntu Frame	Runs an app (or apps) fullscreen, intended for embedded/IoT use
Miracle	A tiling window manager much like i3/sway
egmde	An example desktop environment used to write tutorials about writing Mir compositors
mir_kiosk_x11	Minimal compositor to support fullscreen X11 applications
lomiri-system-compositor	Owns the display and input hardware and supports a nested shell with fewer permissions
miral-kiosk	Runs an app (or apps) maximized
miral-shell	Options for "floating" or "tiling" window management and a limited "desktop" experience with workspaces and a keyboard shortcut for launching a terminal
mir_demo_server	"floating" window management, enables by default all the Wayland extensions Mir supports
miral-system-compositor	Owns the display and input hardware and supports a nested shell with fewer permissions
mirco	The basis of experiments moving MATE shell components to Wayland

For all of these examples Mir provides the code and logic connecting the display and input hardware, and the Wayland protocols connecting to the client. Mir also separates out the concerns such as Window Management, abstracts the graphics stack, and manages Wayland extensions:

- For Window management it provides a default "floating window" strategy, but also provides enough customisation points to support "fullscreen" and "tiling" strategies to be written in high level code;
- For the graphics stack Mir dynamically loads one of several backends (there are five supported by Mir and one more supported by UBports); and,
- Mir implements a range of Wayland extensions (and compositors can add more), some of these are disabled by default (e.g. for security) but can be enabled by compositors either globally or for specific applications.

While much of the above list is examples or experiments of one form or another it proves Mir is flexible and has shipped as a product (in Ubuntu Touch and Ubuntu Frame).

Mir is proven technology that provides a sound base for building your Wayland compositor.

Anatomy of a Wayland Compositor

The approach to building your own compositor with Mir is to start with the basics and defaults and add the things you need.

The simplest Wayland compositor can be written in just a few lines:

```
#include <miral/runner.h>
#include <miral/minimal_window_manager.h>
#include <miral/set_window_management_policy.h>

using namespace miral;

int main(int argc, char const* argv[])
{
    MirRunner runner{argc, argv};

    return runner.run_with(
        {
            set_window_management_policy<MinimalWindowManager>()
        });
}
```

Here, you can see a number of things:

- It is written in C++, a popular high-performance language that supports powerful abstractions;
- It is not much longer than the canonical "Hello World" example; and,
- The code creates a `MirRunner` and then calls `run_with()` supplying the things you wish to add.

When writing your compositor you will deal mostly with the abstractions provided by MirAL (the "Mir Abstraction Layer"). This example shows three:

- `MirRunner` - a class that manages your Mir Wayland compositor
- `MinimalWindowManager` - a simple "floating" window management policy
- `set_window_management_policy` - a functor that applies a window management policy to `MirRunner`.

This example code will build and provides a fully working Wayland compositor that supports the graphics stacks and Wayland protocols available with Mir. (Note you also need to install and use `libmiral-dev` and `mir-graphics-drivers-desktop` for this.)

If you look at any of the examples listed above you will see this pattern repeated. We will look more closely at one of these examples later, but first we'll deal with building and running this example.

Building and running a Wayland Compositor

To compile your program you'll need your normal C++ toolchain and the MirAL [Mir Abstraction Layer] library. In addition to this, running the program needs an appropriate Mir "graphics platform". For a typical development environment this will be something like:

```
sudo apt install libmiral-dev mir-graphics-drivers-desktop
```

If, like the Mir project itself, you're using G++ and CMake, then here's a `CMakeLists.txt` for the "Hello World" example shown above. (If you're using another toolchain, you hopefully know how to do the equivalent with that.)

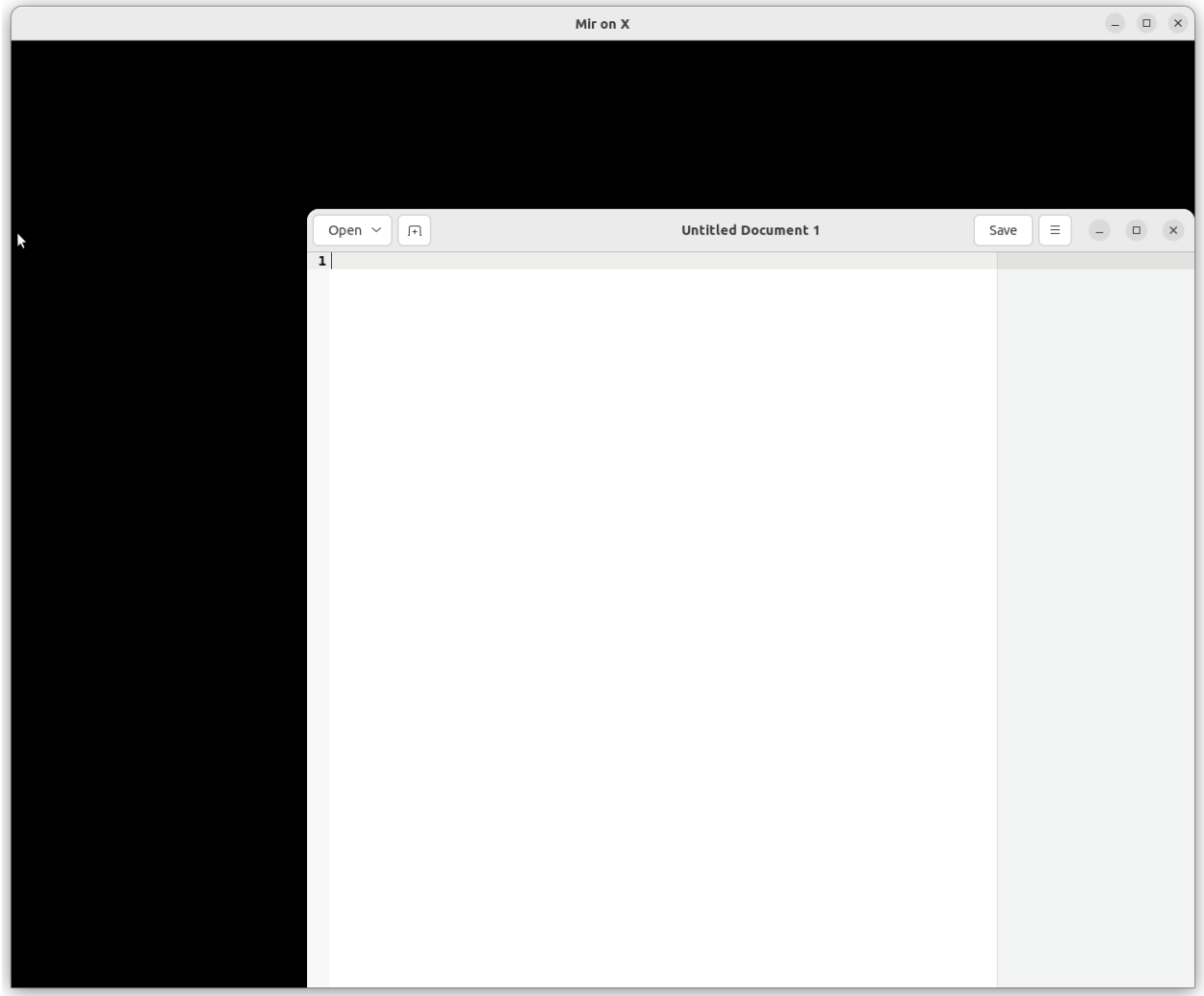
```
cmake_minimum_required(VERSION 3.23)
project(hello_world)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_EXTENSIONS OFF)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

include(FindPkgConfig)
pkg_check_modules(MIRAL miral REQUIRED)

add_executable(hello_world main.cpp)

target_include_directories(hello_world PUBLIC SYSTEM ${MIRAL_INCLUDE_DIRS})
target_link_libraries(     hello_world               ${MIRAL_LDFLAGS})
```

With that, you can build and run as follows:

```
cmake -B cmake-build/
cmake --build cmake-build/
WAYLAND_DISPLAY=wayland-99 cmake-build/hello_world
```

We specify the `WAYLAND_DISPLAY` environment variable for two reasons: because there may be an existing Wayland compositor in your user session; and, so that we can start another application that knows where to connect:

```
WAYLAND_DISPLAY=wayland-99 gedit
```



Of course, this is a very minimalistic example and most compositors will be running as part of an ecosystem that launches appications without typing in these incantations.

An advanced example: Miriway

Compositors can be built for various purposes, but the best example to explain the possibilities is the most recent addition to the family: Miriway. This provides a very "bare bones" graphical shell that can be used as a "Desktop Environment". It can be configured to work with a range of components from Wayland based environments and these include "background", "docks" and other panels, "launchers" and other applications.

Because Miriway attempts to do more than the "Hello World" example it passes a longer list to `run_with()` but the the structure is the same. The various objects passed need to be defined in a scope (such as the body of `main()` as used here) that ensures they live for the duration of the call:

```
int main(int argc, char const* argv[])
{
    MirRunner runner{argc, argv};

    // Change the default Wayland extensions to:
    //  1. to enable screen capture; and,
```

(continues on next page)

(continued from previous page)

```
    //  2. to allow pointer confinement (used by, for example, games)
    // Because we prioritise `user_preference()` these can be disabled by the␣
↪configuration
    WaylandExtensions extensions;
    for (auto const& protocol : {
        WaylandExtensions::zwlr_screencopy_manager_v1,
        WaylandExtensions::zxdg_output_manager_v1,
        "zwp_pointer_constraints_v1",
        "zwp_relative_pointer_manager_v1"})
    {
        extensions.conditionally_enable(protocol, [&](WaylandExtensions::EnableInfo␣
↪const& info)
            {
                return info.user_preference().value_or(true);
            });
    }

    // To support docks, onscreen keyboards, launchers and the like; enable a number of␣
↪protocol extensions,
    // but, because they have security implications only for those applications found in␣
↪`shell_pids`.
    // We'll use `shell_pids` to track "shell-*" processes.
    // We also check `user_preference()` so these can be enabled by the configuration
    ShellPids shell_pids;
    // Protocols we're reserving for shell components_option
    for (auto const& protocol : {
        WaylandExtensions::zwlr_layer_shell_v1,
        WaylandExtensions::zxdg_output_manager_v1,
        WaylandExtensions::zwlr_foreign_toplevel_manager_v1,
        WaylandExtensions::zwp_virtual_keyboard_manager_v1,
        WaylandExtensions::zwp_input_method_manager_v2})
    {
        extensions.conditionally_enable(protocol, [&](WaylandExtensions::EnableInfo␣
↪const& info)
            {
                pid_t const pid = pid_of(info.app());
                return shell_pids.is_found(pid) || info.user_preference().value_
↪or(false);
            });
    }

    ExternalClientLauncher client_launcher;

    // A configuration option to start applications when compositor starts and record␣
↪them in `shell_pids`.
    // Because of the previous section, this allows them some extra Wayland extensions
    ConfigurationOption components_option{
        [&](std::vector<std::string> const& apps)
            {
            for (auto const& app : apps)
            {
                shell_pids.insert(client_launcher.launch(ExternalClientLauncher::split_
```

(continues on next page)

(continued from previous page)

```
↪command(app)));
            }
            },
        "shell-component",
        "Shell component to launch on startup (may be specified multiple times)"};

    // `shell_meta` and `shell_ctrl_alt` provide a lookup to execute the commands␣
↪configured by the corresponding
    // `shell_meta_option` and `shell_ctrl_alt_option` configuration options. These␣
↪processes are added to `shell_pids`
    CommandIndex shell_meta{[&](auto cmd){ shell_pids.insert(client_launcher.
↪launch(cmd)); }};
    CommandIndex shell_ctrl_alt{[&](auto cmd){ shell_pids.insert(client_launcher.
↪launch(cmd)); }};
    ConfigurationOption shell_meta_option{
        [&](std::vector<std::string> const& cmds) { shell_meta.populate(cmds); },
        "shell-meta",
        "meta <key>:<command> shortcut with shell priviledges (may be specified multiple␣
↪times)"};
    ConfigurationOption shell_ctrl_alt_option{
        [&](std::vector<std::string> const& cmds) { shell_ctrl_alt.populate(cmds); },
        "shell-ctrl-alt",
        "ctrl-alt <key>:<command> shortcut with shell priviledges (may be specified␣
↪multiple times)"};

    // `meta` and `ctrl_alt` provide a lookup to execute the commands configured by the␣
↪corresponding
    // `meta_option` and `ctrl_alt_option` configuration options. These processes are NOT␣
↪added to `shell_pids`
    CommandIndex meta{[&](auto cmd){ client_launcher.launch(cmd); }};
    CommandIndex ctrl_alt{[&](auto cmd){ client_launcher.launch(cmd); }};
    ConfigurationOption ctrl_alt_option{
        [&](std::vector<std::string> const& cmds) { ctrl_alt.populate(cmds); },
        "ctrl-alt",
        "ctrl-alt <key>:<command> shortcut (may be specified multiple times)"};
    ConfigurationOption meta_option{
        [&](std::vector<std::string> const& cmds) { meta.populate(cmds); },
        "meta",
        "meta <key>:<command> shortcut (may be specified multiple times)"};

    // Process input events to identifies commands Miriway needs to handle
    ShellCommands commands{
        runner,
        [&] (auto c) { return shell_meta.try_launch(c) || meta.try_launch(c); },
        [&] (auto c) { return shell_ctrl_alt.try_launch(c) || ctrl_alt.try_launch(c); }};

    return runner.run_with(
        {
            X11Support{},
            extensions,
            display_configuration_options,
            client_launcher,
```

(continues on next page)

(continued from previous page)

```
            components_option,
            shell_ctrl_alt_option,
            shell_meta_option,
            ctrl_alt_option,
            meta_option,
            Keymap{},
            PrependEventFilter{[&](MirEvent const*) { shell_pids.reap(); return false; }}
↪,
            AppendEventFilter{[&](MirEvent const* e) { return commands.input_event(e); }}
↪,
            set_window_management_policy<WindowManagerPolicy>(commands)
        });
}
```

The code in `main()` is written using a mixture of classes defined in the `miral` API (such as `ConfigurationOption` and `X11Support`) and ones written as mart of Miriway (such as `ShellPids` and `CommandIndex`).

For the `miral` API there's documentation on the Mir website, and for the `miriway` classes are documented in the code.

2.2.2 Developing Wayland extension protocols for Mir servers

With the Mir 1.3 release we smoothed the way for shells adding Wayland extension protocols in Mir based shells.

The worked example

To create a "worked example" the first thing I needed was a downstream shell that plausibly needed an extension writing. For obvious reasons I chose egmde as the example server. For the extension protocol I chose "primary selection". While a protocol that allows every application to "spy" on the clipboard may not meet the security criteria we use for Mir in IoT, for a desktop environment like egmde it is a great convenience.

The changes to egmde can be seen in this pull request. I'll discuss these in more detail below, but for those that want to follow along the following setup is needed to build the code.

The following Mir packages are needed for building egmde and for developing the Wayland extension:

```
sudo add-apt-repository --update ppa:mir-team/release
sudo apt install libmiral-dev mir-graphics-drivers-desktop
sudo apt install libmirwayland-dev mirtest-dev wlcs
```

The `libmirwayland-dev` provides the tooling needed to implement the protocol in Mir while `mirtest-dev` and `wlcs` provide the testing framework.

wlcs tests for primary selection

While working on this I proposed adding some "primary selection" tests to wlcs and they are included in the 1.1 release of wlcs. Here's an example of what the tests look like:

```
TEST_F(PrimarySelection, source_sees_request)
{
    MockPrimarySelectionSourceListener  source_listener{source_app.source};
    PrimarySelectionOfferListener       offer_listener;
    StubPrimarySelectionDeviceListener  device_listener{sink_app.device, offer_listener};
```

(continues on next page)

(continued from previous page)

```
    source_app.offer(any_mime_type);
    source_app.set_selection();
    sink_app.roundtrip();
    ASSERT_THAT(device_listener.selected, NotNull());

    EXPECT_CALL(source_listener, send(_, _, _))
        .Times(1)
        .WillRepeatedly(Invoke([&](auto*, auto*, int fd) { close(fd); }));

    Pipe pipe;
    zwp_primary_selection_offer_v1_receive(device_listener.selected, any_mime_type, pipe.
↪source);
    sink_app.roundtrip();
    source_app.roundtrip();
}
```

Implementing a protocol extension

The first step in implementing a protocol extension is to use the protocol generator from the `libmirwayland-dev` package, I've added two directories to egmde: `wayland-protocols` and `wayland-generated`. The first of these contains a copy of the protocol definition, the second the generated files and the `cmake` script to generate them. Only the latter is worth reproducing here:

```
set(PROTOCOL "primary-selection-unstable-v1")
set(PROTOCOL_FILE "${CMAKE_CURRENT_SOURCE_DIR}/../wayland-protocols/${PROTOCOL}.xml")
set(GENERATE_FILE "${CMAKE_CURRENT_SOURCE_DIR}/primary-selection-unstable-v1_wrapper")

add_custom_command(
    OUTPUT ${GENERATE_FILE}.cpp
    OUTPUT ${GENERATE_FILE}.h
    DEPENDS ${PROTOCOL_FILE}
    COMMAND "sh" "-c" "mir_wayland_generator zwp_ ${PROTOCOL_FILE} header >${GENERATE_
↪FILE}.h"
    COMMAND "sh" "-c" "mir_wayland_generator zwp_ ${PROTOCOL_FILE} source >${GENERATE_
↪FILE}.cpp"
)

add_custom_target(primary-selection-unstable
    DEPENDS ${GENERATE_FILE}.cpp
    DEPENDS ${GENERATE_FILE}.h
    SOURCES
        ${GENERATE_FILE}.cpp
        ${GENERATE_FILE}.h
)
```

The generator cannot implement the semantics of the protocol: it just provides the "hooks" into which to write the code. The latter can be found in a couple of files `egprimary_selection_device_controller.cpp` and `primary_selection.cpp` (this is only split this way as a lot of logic can be shared with `gtk_primary_selection`.)

All these generated and and-written files are combined into a static library that is used both by egmde and by a new module `egmde-wlcs.so` that provides the wlcs test fixture for egmde.

wlcs test fixture

To run wlcs tests requires a "test fixture" that allows the test executable to load and control the compositor. Mir makes it very easy to implement a fixture: here is the entire "fixture" code for egmde:

```
#include "primary_selection.h"
#include "gtk_primary_selection.h"

#include <miral/wayland_extensions.h>
#include <miral/test_wlcs_display_server.h>

namespace
{
struct TestWlcsDisplayServer : miral::TestWlcsDisplayServer
{
    miral::WaylandExtensions wayland_extensions;

    TestWlcsDisplayServer(int argc, char const** argv) :
        miral::TestWlcsDisplayServer{argc, argv}
    {
        wayland_extensions.add_extension(egmde::primary_selection_extension());
        wayland_extensions.add_extension(egmde::gtk_primary_selection_extension());
        add_server_init(wayland_extensions);
    }
};

WlcsExtensionDescriptor const extensions[] = {
    {primary_selection_extension.name.c_str(), 1},
    {gtk_primary_selection_extension.name.c_str(), 1},
};

WlcsIntegrationDescriptor const descriptor{
    1,
    sizeof(extensions) / sizeof(extensions[0]),
    extensions
};

WlcsIntegrationDescriptor const* get_descriptor(WlcsDisplayServer const* /*server*/)
{
    return &descriptor;
}

WlcsDisplayServer* wlcs_create_server(int argc, char const** argv)
{
    auto server = new TestWlcsDisplayServer(argc, argv);

    server->get_descriptor = &get_descriptor;
    return server;
}

void wlcs_destroy_server(WlcsDisplayServer* server)
{
    delete static_cast<TestWlcsDisplayServer*>(server);
```

(continues on next page)

(continued from previous page)

```
}
}

extern WlcsServerIntegration const wlcs_server_integration {
    1,
    &wlcs_create_server,
    &wlcs_destroy_server,
};
```

Running the tests

With all this in place, we just need to run our updated wlcs with the egmde test fixture:

```
$ `pkg-config --variable test_runner wlcs` cmake-build-debug/egmde-wlcs.so --gtest_
↪filter=Prim*
Note: Google Test filter = Prim*
[==========] Running 5 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 5 tests from PrimarySelection
[ RUN      ] PrimarySelection.source_can_offer
[       OK ] PrimarySelection.source_can_offer (9 ms)
[ RUN      ] PrimarySelection.sink_can_listen
[       OK ] PrimarySelection.sink_can_listen (8 ms)
[ RUN      ] PrimarySelection.sink_can_request
[       OK ] PrimarySelection.sink_can_request (11 ms)
[ RUN      ] PrimarySelection.source_sees_request
[       OK ] PrimarySelection.source_sees_request (9 ms)
[ RUN      ] PrimarySelection.source_can_supply_request
[       OK ] PrimarySelection.source_can_supply_request (8 ms)
[----------] 5 tests from PrimarySelection (45 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test cases run. (45ms total elapsed)
[  PASSED  ] 5 tests
```

The real world

I've not (yet) found a client toolkit that uses "zwp_primary_selection_device_manager_v1", so I've yet to check whether this works with real applications. But the principle steps of this article remain valid even if there are errors in the implementation of this protocol.

2.2.3 Getting Involved in Mir

Getting involved

The Mir project website is https://mir-server.io/, the code is hosted on GitHub

For announcements and other discussions on Mir see: Mir on community.ubuntu

For other questions and discussion about the Mir project: the #mir-server IRC channel on Libera.Chat.

Getting Mir source and dependencies

You can get the source with:

```
git clone https://github.com/canonical/mir.git
cd mir
```

You may need to install git for the system you are working on.

You'll also need a few development tools installed. The exact dependencies and instructions vary across distros.

On Ubuntu

```
sudo apt-get build-dep ./
```

On Fedora and Alpine

As we build these distros in Mir's CI you can copy the instructions from the corresponding files under `spread/build`.

Building

```
cmake -S . -Bbuild
cd build
cmake --build .
```

This creates an example shell (miral-shell) in the bin directory. This can be run directly:

```
bin/miral-app
```

With the default options this runs in a window on X (which is convenient for development).

The miral-shell example is simple, don't expect to see a sophisticated launcher by default. Within this window you can start a terminal with Ctrl-Alt-Shift-T (the "Shift" is needed to avoid Ubuntu's DE intercepting the input). From this terminal you can start apps. For example:

```
$ gedit
```

To exit from miral-shell press Ctrl-Alt-BkSp.

You can install the Mir examples, headers and libraries you've built with:

```
sudo cmake --build . --target install
```

Contributing to Mir

Please file bug reports at: https://github.com/canonical/mir/issues.

The Mir Discourse category can be found at: https://discourse.ubuntu.com/c/mir/15.

Working on code

- Hacking guidelines can be found here: Mir Hacking Guides
- You can configure *Mir* to provide runtime information helpful for debugging by enabling component reports: *Component Reports*
- A guide on versioning Mir DSOs: *DSO Versioning Guide*

2.2.4 Using checkbox-mir to validate your snap graphical environment

This document explains how to use checkbox-mir to validate the graphical environment (drivers, userspace, snap setup) on your device.

The graphics-coreXX interface

To support graphics on Linux on a given piece of hardware, usually a kernel and userspace driver is required. When using snaps, the kernel driver is managed by the operating system, but the userspace driver needs to be available to your application.

The recommended approach for this is using a content snap, through the `graphics-coreXX` interface. The current iteration is `graphics-core22`, see here for more information on that setup: The graphics-core22 interface.

Checkbox

Checkbox is a test automation software we're using extensively at Canonical. We chose it as a framework for providing these tests so that it's easy to run without specific knowledge.

You can find out more about Checkbox on its GitHub page: Checkbox.

checkbox-mir

checkbox-mir is a suite of tests maintained by the Mir team that range from validating the DRM setup through to performance testing of Mir servers. It's provided as a snap package, so on most Linux distributions it's enough to:

```
$ sudo snap install checkbox-mir --devmode
checkbox-mir 0+git.7970755 from Canonical Certification Team (ce-certification-qa)␣
↪installed
```

The `--devmode` flag is required for Checkbox to be able to perform the testing.

You can find out more about it provides through the `snap info` command:

```
$ snap info checkbox-mir
name:      checkbox-mir
summary:   Checkbox tests for the Mir project
publisher: Canonical Certification Team (ce-certification-qa)
store-url: https://snapcraft.io/checkbox-mir
license:   unset
description: |
  Collection of tests to be run on devices that are part of the mir project
commands:
  - checkbox-mir.checkbox-cli
  - checkbox-mir.graphics
  - checkbox-mir.mir
  - checkbox-mir.shell
  - checkbox-mir.snaps
  - checkbox-mir.test-runner
services:
  checkbox-mir.service: simple, enabled, active
snap-id:      zPeO6SzpA2i39SNFjpm8qAkMPTOMvMxh
tracking:     latest/stable
refresh-date: today at 12:04 CEST
channels:
  latest/stable:    0+git.7970755 2023-07-20 (49) 17MB -
  latest/candidate: ↑
  latest/beta:      ↑
  latest/edge:      0+git.7970755 2023-07-19 (49) 17MB -
installed:          0+git.7970755            (49) 17MB devmode
```

Prerequisites

Depending on the test being ran, there are different requirements for it to start, and the test framework will try and determine if it's possible to run the test, rather than fail it.

Here's a list of the different requirements:

1. `graphics-test-tools` installed,
2. `mir-test-tools` installed,
3. a `graphics-coreXX` provider snap installed (by default that's mesa-core20 or mesa-core22, depending on which track of the above you install), and the above snaps, and `checkbox-mir` itself connected to it,
4. to run as a non-privileged user and confirm Mir interactions with the display manager, a local user session needs to be active,

checkbox-mir.checkbox-cli

`checkbox-mir.checkbox-cli` is the interactive entry point, in which you can manually select the test plan and tests to run, as well as control some aspects of the run:

```
 Select test plan
──────────────────────────────────────────────────────────────────────────────
┌──────────────────────────────────────────────────────────────────────────────┐
│                                                                              │
│    ( ) Mir - smoke and performance testing of Mir itself                     │
│    ( ) Snap - verify that the snaps work fine                                │
│    ( ) graphics-core - verify the graphics-core enablement                   │
│                                                                              │
│                                                                              │
│                                                                              │
│                                                                              │
│                                                                              │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
 Press <Enter> to continue                                        (H) Help
```

See Checkbox documentation for more information on the framework itself and how to interact with it.

The graphics-core test plan

The graphics-coreXX test plan attempts to verify the graphics-coreXX enablement by running a handful of utilities verifying a number of aspects of graphics support:

- DRM
- KMS
- eglinfo across GBM, Wayland and X
- libVA
- VDPAU
- Vulkan

```
 Choose tests to run on your system:
──────────────────────────────────────────────────────────────────────────────
┌──────────────────────────────────────────────────────────────────────────────┐
│ [X] - Uncategorised                                                          │
│ [X]    Collect information about connections                                 │
│ [X]    Enumerate available system executables                                │
│ [X]    Dump DRM information for card 1: drm-pci-0000_00_02_0                 │
│ [X]    Run kmscube to verify GBM-KMS setup for card 1:                       │
│        drm-pci-0000_00_02_0                                                  │
│ [X]    Run eglinfo to verify EGL setup for the GBM platform                  │
│ [X]    Run eglinfo to verify EGL setup with a Wayland server                 │
│ [X]    Run eglinfo to verify EGL setup with a X server                       │
│ [X]    Run vainfo to verify libVA setup                                      │
│ [X]    Run vdpauinfo to verify VDPAU setup                                   │
│ [X]    Run vkcube to verify Vulkan setup                                     │
│                                                                              │
```

(continues on next page)

(continued from previous page)

```
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
 Press (T) to start Testing                                           (H) Help
```

The log results from each of these tests should provide insight into the status of the aspect in question. It's out of scope for this document to discuss the results beyond their success / failure state.

The Mir test plan

This test plan runs the smoke and performance tests as found in the mir-test-tools snap. This is what we use to verify updated snaps continue to work across a number of hardware and scenarios.

```
 Choose tests to run on your system:
──────────────────────────────────────────────────────────────────────────────
┌────────────────────────────────────────────────────────────────────────────┐
│[X] - Uncategorised                                                         │
│[X]    Collect information about connections                                │
│[X]    Hardware Manifest                                                    │
│[X]    Provide information about logind sessions                            │
│[X]    Run Mir performance test case:                                       │
│       CompositorPerformance.regression_test_1563287                        │
│[X]    Collect the test report from                                         │
│       mir/performance_CompositorPerformance.regression_test_1563287        │
│[X]    Run Mir performance test case: GLMark2Wayland.fullscreen             │
│[X]    Collect the test report from                                         │
│       mir/performance_GLMark2Wayland.fullscreen                            │
│[X]    Run Mir performance test case: GLMark2Wayland.windowed               │
│[X]    Collect the test report from                                         │
│       mir/performance_GLMark2Wayland.windowed                              │
│[X]    Run Mir performance test case: GLMark2Xwayland.fullscreen            │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
 Press (T) to start Testing                                           (H) Help
```

After selecting the tests, there's another step that we use to skip tests we know fail on a given piece of hardware:

```
System Manifest:
──────────────────────────────────────────────────────────────────────────────
┌────────────────────────────────────────────────────────────────────────────┐
│ Does this machine have this piece of hardware?                             │
│   Run Hosted tests                  ( ) Yes   ( ) No                       │
│   Run Wayland tests                 ( ) Yes   ( ) No                       │
│   Run Xwayland tests                ( ) Yes   ( ) No                       │
│                                                                            │
│                                                                            │
│                                                                            │
│                                                                            │
│                                                                            │
└────────────────────────────────────────────────────────────────────────────┘
 Press (T) to start Testing                                     Shortcuts: y/n
```

The Snap test plan

This test plan is primarily intended for the Mir team, allowing us to test updates to the snaps we maintain:

```
Choose tests to run on your system:
──────────────────────────────────────────────────────────────────────────────
[X] - Uncategorised
[X]    snaps/confined-shell-edge
[X]    snaps/egmde-beta
[X]    snaps/mir-kiosk-kodi-edge
[X]    snaps/mir-kiosk-neverputt-edge
[X]    snaps/mir-kiosk-scummvm-edge
[X]    snaps/mir-test-tools-20-beta
[X]    snaps/mir-test-tools-22-beta
[X]    snaps/mircade-beta
[X]    snaps/miriway-beta
[X]    snaps/ubuntu-frame-osk-20-beta
[X]    snaps/ubuntu-frame-osk-22-beta
──────────────────────────────────────────────────────────────────────────────
Press (T) to start Testing                                    (H) Help
```

Launchers

We provide a couple of launchers for automated runs:

- `checkbox-mir.graphics`: runs the graphics-core test plan
- `checkbox-mir.mir`: runs the Mir test plan

If you want to integrate checkbox-mir into your CI, these are likely what you want to run after having set your device up. You may also create a custom launcher, see the Checkbox launchers tutorial.

`checkbox-mir.snaps` preselects the Snap test plan, but lets you choose which snaps to test.

Running remotely

The recommended way to run Checkbox tests is over the network - and using Checkbox itself, rather than a SSH connection (see Checkbox Remote for more information).

To do this, install checkbox-mir and dependencies on the target device as you would normally, but rather than running the tests there, install Checkbox on your host and issue `checkbox remote <ip.address>`, pointing at the device you want to test.

You'll see the same user interface, be able to run the tests as you would locally - but from a distance, without impacting the test results by the environment you run them from etc. The test results will also be brought into your host at the end.

We'd love your feedback, please come to https://github.com/MirServer/checkbox-mir and file issues, pull and enhancement requests. This test suite will grow for sure, and we'd love to know if you think things could be better.

2.2.5 How to calibrate a touchscreen device

Identifying the touch device

Install the libinput snap:

```
$ sudo snap install libinput
```

Use the libinput snap to identify the device:

```
$ sudo libinput.list-devices | grep "Calibration:      [^n][^/][^a]" --before 11 --after␣
↪5 | grep -v n/a
Device:           QDtech MPI7003
Kernel:           /dev/input/event12
Group:            7
Seat:             seat0, default
Capabilities:     touch
Calibration:      identity matrix
Scroll methods:   none
Click methods:    none
```

From this take the "Kernel" device path:

```
$ udevadm info -q property /dev/input/event21 | grep -e ID_VENDOR_ID -e ID_MODEL_ID
ID_VENDOR_ID=0483
ID_MODEL_ID=5750
```

We will need those IDs for the udev rule. (In this case we will have `ATTRS{idVendor}=="0483"`, `ATTRS{idProduct}=="5750"`,.)

Generating the calibration matrix

Now calibrate the touchscreen by pressing three target spots in turn:

```
$ libinput.calibrate-touchscreen
Starting the calibrate-touchscreen app: touch the target spots.
(Or press <ESC> to quit!)
Calibration = 1.008, 0.001, -0.002, -0.012, 1.003, 0.002
```

Setting the udev rule

We now have the information needed to calibrate the touchscreen:

```
echo 'ATTRS{idVendor}=="0483",ATTRS{idProduct}=="5750", ENV{LIBINPUT_CALIBRATION_MATRIX}=
↪"0 1.035 -0.021 1.007 0 -0.018"' | \
sudo tee /etc/udev/rules.d/99-QDtech-MPI7003.rules
```

And ensure the new rule has been processed by udev:

```
sudo udevadm control --reload
```

Verifying the updated calibration

Now *replug the touchscreen* and see the changes:

```
$ sudo libinput.list-devices | grep "Calibration:      [^n][^/][^a]" --before 11 --after␣
↪5 | grep -v n/a
Device:           QDtech MPI7003
Kernel:           /dev/input/event12
Group:            7
Seat:             seat0, default
Capabilities:     touch
Calibration:      0.00 1.03 -0.02 1.01 0.00 -0.02
Scroll methods:   none
Click methods:    none
```

2.2.6 How to enable graphics-core22 on a device

This document will guide you through the process of assembling a `graphics-core22` snap. With this snap, your device will be ready to run graphical snaps like `ubuntu-frame` or any other *Mir*-based compositor in a snapped environment.

To this end, we will accomplish the following in order:

- How to assemble a `graphics-core22` provider snap
- How to test a `graphics-core22` provider snap
- How to test the confinement of your *Mir*-based snap that relies on the provider snap

Prerequisites

Before we get started, let's make sure that we have everything that we require.

1. Development Board

For starters, make sure that you have some sort of development board on hand.

2. GPU Support Check

Next, make sure that this board runs a `snapd`-enabled Ubuntu image with a kernel that includes GPU support. To do this, we'll run a series of checks.

First, you should ensure that *at least* `/dev/dri/card0` exists, but you may have more depending on your setup. You may check this from the command line.

Next, install the graphics-test-tools snap:

```
sudo snap install graphics-test-tools --channel 22/stable
```

This snap will provide you with `graphics-test-tools.drm-info`. Run it and you should see something like the following:

```
Node: /dev/dri/card1
├──Driver: amdgpu (AMD GPU) version 3.54.0 (20150101)
│   └── ...
├──Device: PCI 1002:747e Advanced Micro Devices, Inc. [AMD/ATI] Device 747e
│   └──Available nodes: primary, render
├──Framebuffer size
│   ├──Width: [0, 16384]
│   └──Height: [0, 16384]
├──Connectors
│   ...
│   ├──Connector 3
│   ├──Object ID: 101
│   ├──Type: HDMI-A
│   ├──Status: connected
│   ├──Physical size: 530x300 mm
│   ├──Subpixel: unknown
│   ├──Encoders: {3}
│   ├──Modes
│   │   ├──1920x1080@60.00 preferred driver phsync pvsync
│   │   ├──1920x1080@60.00 driver phsync pvsync 16:9
│   │   └──...
│   ...
├──Encoders
│   ├──Encoder 0
│   │   ├──Object ID: 82
│   │   ├──Type: TMDS
│   │   ├──CRTCS: {0, 1, 2, 3}
│   │   └──Clones: {0}
│   ...
├──CRTCs
│   ├──CRTC 0
│   │   ├──Object ID: 72
│   │   ├──Legacy info
│   │   │   ├──Mode: 1920x1080@60.00 preferred driver phsync pvsync
│   │   │   └──Gamma size: 256
│   ...
└──Planes
    ├──Plane 0
    │   ├──Object ID: 40
    │   ├──CRTCs: {3}
    │   ├──Legacy info
    │   │   ├──FB ID: 0
    │   ...
```

You want to make sure that you have:

- 1 Node
- 1 or more Connector(s)
- 1 or mor CRTC(s)
- 1 or more Plane(s)

Finally, If you have a display connected, then one of your `Connectors` should read `Status: connected`. This `Connector` will also list many `Modes` that the display can use.

Troubleshooting

If `drm-info` does *not* produce results similar to those above, then this indicates that your kernel **is NOT yet ready** to support the GPU. This means that kernel enablement is required before Ubuntu Frame can be enabled.

3. Driver Support Check

Finally, we will need to check that you have the proper userspace drivers installed on the system, namely:

- `EGL` (`libEGL.so.1`)
- `GLES` (`libGLESv2.so.2`)
- `GBM` (`libgbm.so.1`)

While *Mir* can be enabled on platforms that do not support `GBM/KMS`, this requires a paltform implementation to be written in *Mir*. This work is out of the scope of this guide.

How to assemble a `graphics-core22` provider snap

With the prerequisites out of the way, it is time to assemble a `graphics-core22` snap.

The snap itself must provide the libraries described in this document. Unless you are incredibly space-constrained or in an unusual circumstance, it is a good idea to provide **all** of these libraries. Even if your board doesn't support these libraries, a failure to include them will result in applications that would otherwise fall back to a supported API failing to start.

`libEGL`, `libGLESv2` and `libgbm` should be provided by the vendor's drivers. `libvulkan` *may* be provided by the vendor as well. The remaining packages should be available in the Ubuntu archive.

Most of these libraries do **NOT** need to be in a fixed location in the snap. However, the following paths **MUST** be provided in a fixed location by your snap:

1. `libdrm`:

```
parts:
  drm:
    # DRM userspace
    ...
    organize:
      # Expected at /libdrm by the `graphics-core22` interface
      usr/share/libdrm: libdrm
    ...
```

2. `drirc.d` (optional, if your vendor drivers supply `drirc` configuration):

```
parts:
  dri:
    # Userspace drivers
    ...
    organize:
      # Expected at /drirc.d by the `graphics-core22` interface
      usr/share/drirc.d: drirc.d
    ...
```

3. `X11`:

```
parts:
  x11:
    ...
    organize:
      # Expected at /X11 by the `graphics-core22` interface
      usr/share/X11: X11
    ...
```

The remainder of the paths will be established using the required script at the location `bin/graphics-core22-provider-wrapper`. This script will be invoked with `graphics-core22-provider-wrapper $EXECUTABLE $ARGS...`. This script does the following:

1. It exports any environment variables that are required for the binary to find and use the vendor drivers
2. It then executes the provided `$EXECUTABLE` with `$ARGS...`.

An example of a script that provides `mesa-core22` is as follows:

```
# Dervied from:
# https://github.com/canonical/mesa-core22/blob/main/scripts/bin/graphics-core22-
↪provider-wrapper.in


#!/bin/bash
set -euo pipefail


# Find the parent directory of the wrapper script. This will be
# the path of the provider snap in the client. Your files with
# be found at a path relative to this path.
SELF="$( cd -- "$(dirname "$0")/.." ; pwd -P )/usr"


# The arch triplet(s) for this driver (eg: arm64-linux-gnu, i386-linux-gnu, etc)
ARCH_TRIPLETS=( x86_64-linux-gnu )


# VDPAU_DRIVER_PATH only supports a single path, rely on LD_LIBRARY_PATH instead
for arch in ${ARCH_TRIPLETS[@]}; do
  LD_LIBRARY_PATH=${LD_LIBRARY_PATH:+$LD_LIBRARY_PATH:}${SELF}/lib/${arch}:${SELF}/lib/$
↪{arch}/vdpau
  LIBGL_DRIVERS_PATH=${LIBGL_DRIVERS_PATH:+$LIBGL_DRIVERS_PATH:}${SELF}/lib/${arch}/dri/
  LIBVA_DRIVERS_PATH=${LIBVA_DRIVERS_PATH:+$LIBVA_DRIVERS_PATH:}${SELF}/lib/${arch}/dri/
done

__EGL_VENDOR_LIBRARY_DIRS=${__EGL_VENDOR_LIBRARY_DIRS:+$__EGL_VENDOR_LIBRARY_DIRS:}$
↪{SELF}/share/glvnd/egl_vendor.d
__EGL_EXTERNAL_PLATFORM_CONFIG_DIRS=${__EGL_EXTERNAL_PLATFORM_CONFIG_DIRS:+$__EGL_
↪EXTERNAL_PLATFORM_CONFIG_DIRS:}${SELF}/share/egl/egl_external_platform.d
VK_LAYER_PATH=${VK_LAYER_PATH:+$VK_LAYER_PATH:}${SELF}/share/vulkan/implicit_layer.d/:$
↪{SELF}/share/vulkan/explicit_layer.d/
XDG_DATA_DIRS=${XDG_DATA_DIRS:+$XDG_DATA_DIRS:}${SELF}/share
```

(continues on next page)

(continued from previous page)

```
# These are in the default LD_LIBRARY_PATH, but in case the snap dropped it inadvertently
if [ -d "/var/lib/snapd/lib/gl" ] && [[ ! ${LD_LIBRARY_PATH} =~ (^|:)/var/lib/snapd/lib/
↪gl(:|$) ]]; then
  LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/var/lib/snapd/lib/gl
fi

if [ -d "/var/lib/snapd/lib/glvnd/egl_vendor.d" ]; then
  # This needs to be prepended, as glvnd goes depth-first on these
  __EGL_VENDOR_LIBRARY_DIRS=/var/lib/snapd/lib/glvnd/egl_vendor.d:${__EGL_VENDOR_LIBRARY_
↪DIRS}
fi

if [ -d "/var/lib/snapd/lib/vulkan/icd.d" ]; then
  XDG_DATA_DIRS=${XDG_DATA_DIRS}:/var/lib/snapd/lib
fi

if [ -d "/var/lib/snapd/lib/gl/vdpau" ]; then
  LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/var/lib/snapd/lib/gl/vdpau
fi

export LD_LIBRARY_PATH
# Mesa-specific environment variable pointing to directory containing $VENDOR_dri.so␣
↪files
export LIBGL_DRIVERS_PATH
export LIBVA_DRIVERS_PATH
# glvnd-specific environment variable pointing to directory containing ICD descriptor␣
↪files
export __EGL_VENDOR_LIBRARY_DIRS
# EGL external platform interface specific environment
export __EGL_EXTERNAL_PLATFORM_CONFIG_DIRS
export VK_LAYER_PATH
export XDG_DATA_DIRS

exec "$@"
```

This script has exported the proper paths via environment variables so that any snap that connects to our provider snap can properly resolve these libraries.

Multiarch Support

If the vendor drivers provide libraries for multiple sub-architectures (for example, 32-bit and 64-bit ARM) your provider snap can supply both. In this case, you will need to include all provided arch triples in the `ARCH_TRIPLETS` array in the above script (for example: `ARCH_TRIPLETS=( armhf-linux-gnu arm64-linux-gnu )`), and will need to set a few more environment variables in the `graphics-core22-provider-wrapper`. An example from `mesa-core22` (which provides both `x86_64-linux-gnu` and `i386-linux-gnu`) adds these lines:

```
# /bin/graphics-core22-provider-wrapper

...
if [ "$SNAP_ARCH" == "amd64" ]; then
  GCONV_PATH=${GCONV_PATH:+$GCONV_PATH:}${SELF}/lib/i386-linux-gnu/gconv
```

(continues on next page)

(continued from previous page)

```
fi

if [ -d "/var/lib/snapd/lib/gl32" ] && [[ ! ${LD_LIBRARY_PATH} =~ (^|:)/var/lib/snapd/
↪lib/gl32(:|$) ]]; then
  LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/var/lib/snapd/lib/gl32
fi

if [ -d "/var/lib/snapd/lib/gl32/vdpau" ]; then
  LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/var/lib/snapd/lib/gl32/vdpau
fi

...

[ -z ${GCONV_PATH+x} ] || export GCONV_PATH
```

The `snapcraft.yaml` file will also need to be updated to point at the 32 bit paths. You may find an example of using `i386` in mesa-core22.

How to supply the vendor libraries

The following example supplies drivers from `libmali`. Using a parameter for `make`, the libraries will be installed to `$SNAPCRAFT_PART_INSTALL/usr/lib`. These libraries do not need to be in a particular path.

```
parts:
  ...
  vendor-drivers:
  plugin: make
  source: libmali
  make-parameters:
    - LIBDIR=$SNAPCRAFT_PART_INSTALL/usr/lib/$CRAFT_ARCH_TRIPLET_BUILD_FOR
  prime:
    - usr/lib
  ...

...
slots:
 graphics-core22:
   interface: content
   read:
     - [$SNAP]
```

How to supply the remaining libraries

As mentioned before, you should be able to get the remaining libraries from the Ubuntu archive. The part below can supply these libraries, and they may be organized into any directory, in this case they are in the same path as would be found in a classic Ubuntu system. Here is an example from the `mesa-core22` snapcraft.yaml.

```
parts:
  drm:
    # DRM userspace
```

(continues on next page)

(continued from previous page)

```
    #   o libdrm.so.2
    plugin: nil
    stage-packages:
      - libdrm2
      - libdrm-common
    organize:
      # Expected at /libdrm by the `graphics-core22` interface
      usr/share/libdrm: libdrm
    prime:
      - usr/lib
      - usr/share/doc/*/copyright
      - libdrm

  va:
    # Video Acceleration API
    #   o libva.so.2
    plugin: nil
    stage-packages:
      - libva2
      - libva-drm2
      - libva-x11-2
      - libva-wayland2
    prime:
      - usr/lib
      - usr/share/doc/*/copyright

  dri:
    # Userspace drivers
    plugin: nil
    stage-packages:
      - libgl1-mesa-dri
      - va-driver-all
      - vdpau-driver-all
      - libvdpau-va-gl1
      - mesa-vulkan-drivers
      - libglx-mesa0
    organize:
      # Expected at /drirc.d by the `graphics-core22` interface
      usr/share/drirc.d: drirc.d
    prime:
      - usr/lib
      - usr/share/doc/*/copyright
      - usr/share/vulkan
      - drirc.d
    override-stage: |
      # Strip any absolute paths out of the Vulkan ICD driver manifests;
      # the driver `.so` files will not be in a fixed location, so must use
      # relative paths and rely on `LD_LIBRARY_PATH` (set by the graphics-core22-
↪proivder-wrapper script)
      # to find the driver `.so`s
      sed -i 's@/usr/lib/[a-z0-9_-]\+/@@' ${CRAFT_PART_INSTALL}/usr/share/vulkan/*/*.json
      craftctl default
```

(continues on next page)

(continued from previous page)

```
    craftctl set version=$( apt-cache policy libgl1-mesa-dri | sed -rne 's/^\
↪s+Candidate:\s+([^-]*)-.+$/\1/p' )

  x11:
    # X11 support (not much cost to having this)
    #   o libGLX.so.0
    plugin: nil
    stage-packages:
      - libglx0
      - libx11-xcb1
      - libxau6
      - libxcb-dri2-0
      - libxcb-dri3-0
      - libxcb-present0
      - libxcb-sync1
      - libxcb-xfixes0
      - libxcb1
      - libxdamage1
      - libxdmcp6
      - libxshmfence1
    organize:
      # Expected at /X11 by the `graphics-core22` interface
      usr/share/X11: X11
    prime:
      - usr/lib
      - usr/share/doc/*/copyright
      - X11

  wayland:
    # Wayland support (not much cost to having this)
    plugin: nil
    stage-packages:
      - libwayland-client0
      - libwayland-cursor0
      - libwayland-egl1
      - libwayland-server0
      - libnvidia-egl-wayland1
    prime:
      - usr/lib
      - usr/share/doc/*/copyright
      - usr/share/egl/egl_external_platform.d
```

When you're ready, build your snap using `snapcraft`.

Troubleshooting

Vendor drivers require libraries newer than core22 provides

Some vendor driver binaries might be built against newer libraries than are provided in the Ubuntu 22.04 repositories. Particularly, the core `libwayland` libraries may introduce new features since 22.04 that vendor binaries might require. This will manifest as missing symbol errors at runtime.

Generally, new version requirements for libraries can be handled by adding a snapcraft part building the required library version. An example of this for `libwayland` is:

```
parts:
 ...
 wayland:
   plugin: meson
   meson-parameters:
     - -Ddocumentation=false
     - -Dtests=false
     - --prefix=$SNAPCRAFT_PART_INSTALL/usr
   build-packages:
     - libxml2-dev
     - libffi-dev
     - libexpat1-dev
     - pkg-config
   override-build: |
     snapcraftctl build
   source: https://gitlab.freedesktop.org/wayland/wayland.git
   source-type: git
   source-tag: 1.21.0
   prime:
     - usr/lib
```

How to test a `graphics-core22` provider snap

Now we have our `graphics-core22` provider snap built. However, we don't yet know if it works properly. Testing this snap is our next task.

Assuming that you have already installed the `graphics-test-tools` snap, connect the snap that you've built to `graphics-test-tools`:

```
sudo snap disconnect graphics-test-tools:graphics-core22
sudo snap connect graphics-test-tools:graphics-core22 <your-snap>:graphics-core22
```

With that installed, we can begin testing.

Use `eglinfo` to test

This is the most basic test that can be performed, and is a good indication of baseline GPU driver setup. This should list *at least* a GBM platform with the expected vendor and `OpenGL_ES` client API. An abbreviated example (on a `mesa-core22` system):

```
$ graphics-test-tools.eglinfo
EGL client extensions string:
        EGL_EXT_device_base EGL_EXT_device_enumeration EGL_EXT_device_query
        EGL_EXT_platform_base EGL_KHR_client_get_all_proc_addresses
        EGL_EXT_client_extensions EGL_KHR_debug EGL_EXT_platform_device
        EGL_EXT_platform_wayland EGL_KHR_platform_wayland
        EGL_EXT_platform_x11 EGL_KHR_platform_x11 EGL_EXT_platform_xcb
        EGL_MESA_platform_gbm EGL_KHR_platform_gbm
        EGL_MESA_platform_surfaceless


GBM platform:
EGL API version: 1.5
EGL vendor string: Mesa Project
EGL version string: 1.5
EGL client APIs: OpenGL OpenGL_ES
EGL driver name: iris
EGL extensions string:
        EGL_ANDROID_blob_cache EGL_ANDROID_native_fence_sync
        EGL_EXT_buffer_age EGL_EXT_create_context_robustness
        EGL_EXT_image_dma_buf_import EGL_EXT_image_dma_buf_import_modifiers
        EGL_IMG_context_priority EGL_KHR_cl_event2 EGL_KHR_config_attribs
        EGL_KHR_context_flush_control EGL_KHR_create_context
        EGL_KHR_create_context_no_error EGL_KHR_fence_sync
        EGL_KHR_get_all_proc_addresses EGL_KHR_gl_colorspace
        EGL_KHR_gl_renderbuffer_image EGL_KHR_gl_texture_2D_image
        EGL_KHR_gl_texture_3D_image EGL_KHR_gl_texture_cubemap_image
        EGL_KHR_image EGL_KHR_image_base EGL_KHR_image_pixmap
        EGL_KHR_no_config_context EGL_KHR_reusable_sync
        EGL_KHR_surfaceless_context EGL_EXT_pixel_format_float
        EGL_KHR_wait_sync EGL_MESA_configless_context EGL_MESA_drm_image
        EGL_MESA_image_dma_buf_export EGL_MESA_query_driver
        EGL_WL_bind_wayland_display
Configurations:
         bf lv colorbuffer dp st  ms        vis   cav bi  renderable  supported
  id sz  l  r  g  b  a th cl ns b        id   eat nd gl es es2 vg surfaces
------------------------------------------------------------------------
0x01 32  0 10 10 10  2  0  0  0 0 0x30335241--             y  y  y         win
0x02 32  0 10 10 10  2 16  0  0 0 0x30335241--             y  y  y         win
0x03 32  0 10 10 10  2 24  0  0 0 0x30335241--             y  y  y         win
0x04 32  0 10 10 10  2 24  8  0 0 0x30335241--             y  y  y         win
0x05 32  0 10 10 10  2  0  0  2 1 0x30335241--             y  y  y         win
... lots more configurations, then other platform details ...
```

If `eglinfo` does **NOT** list a GBM platform, or generates errors then you want to look at *possible complications*. If `eglinfo` does list a GBM platform then we can proceed to testing `ubuntu-frame`.

Test if `ubuntu-frame` works

First, install `ubuntu-frame`:

```
sudo snap install ubuntu-frame --devmode
```

Next, disconnect it from the default graphics interface:

```
sudo snap disconnect ubuntu-frame:graphics-core22
```

Then, connect it to your new provider snap:

```
sudo snap connect ubuntu-frame:graphics-core22 <your-snap>:graphics-core22
```

Finally, run `ubuntu-frame`:

```
ubuntu-frame
```

If everything previously has gone correctly this should result in `ubuntu-frame` coming up on the connected outputs. If not, the log messages will hopefully help identify issues.

How to test the confinement of your *Mir*-based snap

Once you have your `graphics-core22` provider snap setup, and you are confident that it works well, you'll want to make sure that you can run the snap that *depends on* your provider snap in a confined mode. Since we already have `ubuntu-frame` setup, we will use it as an example. However, you should be able to run these tests on any *Mir*-based snap.

First, let's uninstall `ubuntu-frame` if it is installed:

```
sudo snap remove ubuntu-frame
```

Next, we can install `ubuntu-frame` without the `--devmode` flag and connect it to the `graphics-core22` slot of your provider snap:

```
sudo snap install ubuntu-frame
sudo snap disconnect ubuntu-frame:graphics-core22
sudo snap connect ubuntu-frame:graphics-core22 <your-snap>:graphics-core22
```

Finally, we will run `ubuntu-frame` like before:

```
ubuntu-frame
```

If everything works, then `ubuntu-frame` is ready. If not, we'll have to troubleshoot.

Troubleshooting

When bringing up a confined ubuntu-frame snap on a new board with new drivers there are two separate access control mechanisms:

- AppArmor
- The devices cgroup

AppArmor

The first is relatively easy to debug. When AppArmor would deny access to a resource, it outputs a nice message in `dmesg`:

```
apparmor="DENIED" operation="open" profile="snap.ubuntu-frame.ubuntu-frame" name="/run/
↪udev/data/c189:1" <other stuff>
```

You can find the AppArmor profile at `/var/lib/snapd/apparmor/profiles/snap.ubuntu-frame.ubuntu-frame`. Make modifications to the profile using your favorite text editor. Finally, apply the change to replace the default confinement:

```
sudo apparmor_parser -r /var/lib/snapd/apparmor/profiles/snap.ubuntu-frame.ubuntu-frame
```

Warning: Please note that this profile is regenerated automatically by many `snapd` actions. As such, the modifications that you just made are *temporary*. To make them permanent, you will need to upstream the fix against `snapd`.

The devices cgroup

The cgroup confinement is less easy to debug. The kernel emits no logs when the devices cgroup denies access to a device node. The device node on the filesystem will appear to have the correct permissions (and, for example, you will be able to touch it), but calls to `open()` will fail with `EPERM`. If everything seems to be set up correctly, and there are no errors in `dmesg`, but `Mir` is failing it's likely that the devices cgroup confinement is to blame.

Additionally, it is more difficult to test as the cgroup is only set up during snap run startup, and so the knobs we need to twiddle only exist while the snap is running.

We can get around this by running:

```
snap run --shell ubuntu-frame
```

This command will get snapd to do all the initialisation and drop us into a shell. Once the shell exists, there will be two(?!) cgroup folders found under `/sys/fs/cgroup/devices`:

1. /sys/fs/cgroup/devices/system.slice/snap.ubuntu-frame.ubuntu-frame.${UUID}.scope, and
2. /sys/fs/cgroup/devices/snap.ubuntu-frame.ubuntu-frame

It is (2) that we're after. You can check that you've got the right directory, because the `devices.list` file will contain a bunch of lines like:

```
c 5:1 rwm
c 5:2 rwm
c 136:* rwm
c 137:* rwm
c 138:* rwm
```

You now need to give the cgroup permission to access the necessary devices, for which you need the `major:minor` of the device nodes. You can find that with ls:

```
$ ls -la /sys/fs/cgroup/devices/snap.ubuntu-frame.ubuntu-frame
crw-rw-rw- 1 root root 10, 60 Jan 10 04:56 /dev/mali0
```

Here we see that the `/dev/mali0` device node has `major:minor` equal to `10:60`.

Now we can enable access to the relevant device node, via:

```
echo "c 10:60 rw" | sudo tee /sys/fs/cgroup/devices/snap.ubuntu-frame.ubuntu-frame/
↪devices.allow
```

And now we can go back to the shell, and try running `ubuntu-frame`:

```
$SNAP/usr/local/bin/frame
```

2.2.7 How to test Mir for a release

The following document details the criteria that must be met before we can publish an official release of Mir. When adding to this test plan, please do not fail to consider how your additions may be automated in the future, as this is the end goal of our release testing.

Note that all of these tests must be run *after* a release branch is created in the Mir repository, such that the repository is updated beforehand.

Setup

We will run the test plan on the `miral-app` test compositor.

1. Install mir from the release candidate PPA

   ```
   sudo add-apt-repository --update ppa:mir-team/rc
   sudo apt install mir-demos mir-platform-graphics-virtual mir-graphics-drivers-
   ↪desktop mir-test-tools
   ```

2. Confirm that you can run `miral-app`

Testing each graphics platform

Each test must be performed across a combination of different display platforms and Ubuntu releases. The following matrix provides the environments in which we need to test:

	24.04	24.10
gbm-kms		
eglstream-kms		
eglstream-kms + gbm-kms hybrid		
x11		
wayland		
virtual		

To check which display platform we've selected, we can run `miral-app` and grep for the platform string as follows:

```
miral-app | grep "Selected display driver:"
```

Given the types of outputs that you have configured in your environment, you should encounter one of the following scenarios for each output:

1. When NOT on an Nvidia platform and NOT in a hosted environment, then `mir:gbm-kms` is selected
2. When you have an Nvidia card connected to an output *and* the system is using Nvidia's proprietary drivers, then `mir:eglstream-kms` is selected
3. When you are running the compositor hosted in a session that supports X11, then `mir:x11` is selected
4. When you are running the compositor hosted in a session that supports wayland *and* you force Mir to use the `mir:wayland` platform using:

   ```
   miral-app --wayland-host=$WAYLAND_DISPLAY
   ```

   then `mir:wayland` is selected.

5. Check that the virtual platform can run and you can connect to it via a VNC:

   ```
   MIR_SERVER_PLATFORM_DISPLAY_LIBS=mir:virtual MIR_SERVER_VIRTUAL_OUTPUT=1280x1024␣
   ↪WAYLAND_DISPLAY=wayland-1 miral-app \
       --add-wayland-extension=zwp_virtual_keyboard_manager_v1:zwlr_virtual_pointer_
   ↪manager_v1:zwlr_screencopy_manager_v1
   ```

   After, in a separate VT, connect to the VNC:

   ```
   WAYLAND_DISPLAY=wayland-1 ubuntu-frame-vnc
   ```

   Then, check that you can connect via a VNC viewer:

   ```
   gvncviewer localhost
   ```

The smoke tests

These verify our demo clients work with `mir_demo_server` and should be run for each platform on each of the ubuntu series (see previous section).

1. Decide which platform that you want to run the smoke tests on (e.g. gbm-kms, virtual, X11, etc.)
2. Run `mir-smoke-test-runner` like so:

```
MIR_SERVER_PLATFORM_DISPLAY_LIBS=<YOUR_PLATFORM> mir-smoke-test-runner
```

   For example, if I wanted to run the tests on the virtual platform, I would run:

```
MIR_SERVER_PLATFORM_DISPLAY_LIBS=mir:virtual MIR_SERVER_VIRTUAL_OUTPUT=1280x1024␣
↪mir-smoke-test-runner
```

   (Note that you do **not** have to connect via VNC for the smoke tests to run with the virtual platform.)

3. Confirm that the final output of the test is: `Smoke testing complete with returncode 0`.

Applications

For each empty box in the matrix above, ensure that the following applications can start

1. Test that the following QT Wayland applications can be started and can receive input:

```
sudo apt install qtwayland5 kate qterminal

# First...
kate

# Then...
qterminal
```

2. Test that `weston-terminal` can be started and can receive input:

```
sudo apt install weston
weston
```

3. Test that `glmark2-wayland` can be run:

```
sudo apt install glmark2-wayland
glmark2-wayland
```

4. (If using gbm-kms on a system with multiple GPUs) Test hybrid support with `glmark2-wayland`

```
sudo apt install glmark2-wayland
glmark2-wayland
DRI_PRIME=0 glmark2-wayland
DRI_PRIME=1 glmark2-wayland
```

   (If more than 2 GPUs, may do `DRI_PRIME=2 glmark2-wayland`, etc)

5. Test that `gnome-terminal` can be started and can receive input:

```
sudo apt install gnome-terminal
gnome-terminal
```

6. Test that `X11` apps can be started and can receive input:

```
sudo apt install x11-apps

# first,
xeyes

# then
xedit

# finally
xcalc
```

Mir Console Providers

For each Ubuntu release ensure that the compositor can start with each of the console providers:

	24.04	24.10
vt		
logind		
minimal		

The following describes how to select each console provider:

1. **vt**:

```
miral-app --console-provider=vt --vt=4
```

   - This requires running with root privileges
   - You need to ensure that `XDG_RUNTIME_DIR` is set in the environment. If using `sudo`, it might strip this out; running something like `sudo env XDG_RUNTIME_DIR=/run/user/1000 miral-shell ...` will ensure this is set.

2. **logind**:

```
miral-app --console-provider=logind
```

   - You can switch to vt4 and sign in

3. **minimal**:

```
miral-app --console-provider=minimal
```

   - This is used when all others fail
   - This does not provide VT switching capabilities (Ctrl-Alt-F1, etc)
   - This is *only* used for the `gbm-x11`, `gbm-wayland`, and `virtual` platforms

Window Manager Examples

Run with different window managers and confirm that the window management styles work as expected:

```
miral-app --window-manager=floating # traditional floating window manager
miral-app --window-manager=tiling # a tiling window manager
miral-app -kiosk # a typical kiosk
```

Testing Downstream Snaps (e.g. Ubuntu Frame and Miriway)

For each of our downstream snaps, check that you have installed a build with the Mir version under test (typically from the `beta` channel). Then run the tests for that snap.

E.g. for `mir-test-tools`:

```
  /snap/mir-test-tools/current/bin/selftest
```

2.2.8 How to Update Symbols Map Files

The Mir project is a collection of C++ libraries that a consumer can use to create a Wayland compositor. In order for consumers of Mir's libraries to be confident in what they're building against, the symbols of each library are versioned.

To describe these versions, each user-facing library defines a `symbols.map` file. These files are comprised of version stanzas which contain the symbols associated with that version. There are `symbols.map` files for the following libraries, although only the first three are typically used in practice:

- *miral*
- *miroil*
- *mirserver*
- mircore
- mircommon
- mirplatform
- mirwayland

It is tedious to edit these files. Luckily for us, this versioning business is (mostly) automated so that we have to think very little about it.

When are symbols.map files updated

Two different circumstances will prompt a developer to update these files:

1. **A new symbol is added to a library**

   This means that we have extended the existing interface. The new symbols can simply be put in a new stanza with a "bumped" minor version. Note that the minor version must only be bumped once per release cycle. This means that if I add a new symbol two months before a release and you add a symbol 1 day before the release, then our new symbols will go in the same stanza. In that situation, I would have been the one to create the new stanza.

2. **A symbol is changed or removed** (e.g. a new parameter is added to a method call, a class is removed, etc.)

   This is an API break. This means that a consumer of the library can no longer safely compile against the new version of that library without maybe breaking their build. In this scenario, we are forced to create a single new stanza at a brand new version in the `symbols.map` file. This new stanza will contain all of the symbols in that library.

If you forget to update the `symbols.map` file, Github CI will complain and prevent the merge until the issue is addressed. You can find results of this action here. Each time a pull request is open or updated on Github, this action will be run. If you check the log of this action, you will see which symbols have been changed so that you can address the issue.

Warning: Please be aware that this action is not sophisticated enough to know if you changed the definition of a symbol. If you add or remove a symbol, the action will inform you. However, if you modify a symbol (e.g. by changing the parameters that a method takes), you will *not* be informed that the symbol has changes. Such changes will need to be moved to a new stanza manually.

Setup

Before we can update the symbols of Mir's libraries, we'll want to set up our environment so that the our tools can work correctly.

To do this, you will first want to install **clang-19**. Note that it is very important that you have the most recent version of clang, as that works the best.

```
sudo apt install clang-19
```

Afterwards, you will want to set the following environment variables to point at the path of `libclang.so` and `libclang.so`'s `lib` directory. Note that these paths are likely to be the same on most Ubuntu 24.04 setups, but may vary on other distros:

```
export MIR_SYMBOLS_MAP_GENERATOR_CLANG_SO_PATH=/usr/lib/llvm-19/lib/libclang.so.1
export MIR_SYMBOLS_MAP_GENERATOR_CLANG_LIBRARY_PATH=/usr/lib/llvm-19/lib
```

It is recommended that you put those `export ...` commands in your `.bashrc` so that you don't have to think about it in the future.

And that's it! Now we are ready to update our symbols automatically.

How to update miral symbols

Scenario 1: Adding a new symbol

1. Make the additive change to the interface (e.g. by adding a new method to an existing class)
2. If not already bumped in this release cycle, bump `MIRAL_VERSION_MINOR` in `src/CMakeLists.txt`
3. From the root of the project, run:

   ```
   cmake --build <BUILD_DIRECTORY> --target generate-miral-symbols-map
   ```

4. Check that your new symbols is in the `symbols.map` file:

   ```
   git diff src/miral/symbols.map
   ```

5. Regenerate Debian symbols:

```
cmake --build <BUILD_DIRECTORY> --target regenerate-miral-debian-symbols
```

Scenario 2: Removing or changing a symbol

1. Make a destructive change to the interface (e.g. remove a parameter from a method).
2. If not already bumped in this release cycle, bump `MIRAL_VERSION_MAJOR` in `src/CMakeLists.txt`. Set `MIRAL_VERSION_MINOR` and `MIRAL_VERSION_PATCH` to `0`
3. If not already bumped in this release cycle, bump `MIRAL_ABI` in `src/miral/CMakeLists.txt`
4. From the root of the project, run:

   ```
   cmake --build <BUILD_DIRECTORY> --target generate-miral-symbols-map
   ```

5. Check that your symbols are reflected properly in the `symbols.map` file:

   ```
   git diff src/miral/symbols.map
   ```

6. Regenerate the debian symbols:

   ```
   cmake --build <BUILD_DIRECTORY> --target regenerate-miral-debian-symbols
   ```

If `MIRAL_ABI` needed to be updated, you should also run:

```
./tools/update_package_abis.sh
```

How to update miroil symbols

Scenario 1: Adding a new symbol

1. Make the additive change to the interface (e.g. by adding a new method to an existing class)
2. If not already bumped in this release cycle, bump `MIROIL_VERSION_MINOR` in `src/miroil/CMakeLists.txt`
3. From the root of the project, run:

   ```
   cmake --build <BUILD_DIRECTORY> --target generate-miroil-symbols-map
   ```

4. Check that your new symbols is in the `symbols.map` file:

   ```
   git diff src/miroil/symbols.map
   ```

Scenario 2: Removing or changing a symbol

1. Make a destructive change to the interface (e.g. remove a parameter from a method).
2. If not already bumped in this release cycle, bump `MIROIL_ABI` in `src/miroil/CMakeLists.txt`. Set `MIROIL_VERSION_MINOR` and `MIROIL_VERSION_PATCH` to `0`
3. From the root of the project, run:

   ```
   cmake --build <BUILD_DIRECTORY> --target generate-miroil-symbols-map
   ```

4. Check that your symbols are reflected properly in the `symbols.map` file:

```
git diff src/miroil/symbols.map
```

5. Regenerate the debian symbols:

```
cmake --build <BUILD_DIRECTORY> --target regenerate-miroil-debian-symbols
```

If `MIROIL_ABI` needed to be updated, you should also run:

```
./tools/update_package_abis.sh
```

How to update mirserver symbols map

Scenario 1: Adding a new symbol

1. Make the additive change to the interface (e.g. by adding a new method to an existing class)
2. If not already bumped in this release cycle, bump `MIR_VERSION_MINOR` in `CMakeLists.txt`
3. If not already bumped in this release cycle, bump `MIRSERVER_ABI` in `src/server/CMakeLists.txt`
4. From the root of the project, run:

```
cmake --build <BUILD_DIRECTORY> --target generate-mirserver-symbols-map
```

5. Check that your new symbols is in the `symbols.map` file:

```
git diff src/server/symbols.map
```

If `MIRSERVER_ABI` needed to be updated, you should also run:

```
./tools/update_package_abis.sh
```

Scenario 2: Removing or changing a symbol

1. Make a destructive change to the interface (e.g. remove a parameter from a method).
2. If not already bumped in this release cycle, bump `MIR_VERSION_MAJOR` in `CMakeLists.txt`. Set `MIR_VERSION_MINOR` and `MIR_VERSION_PATCH` to `0`
3. If not already bumped in this release cycle, bump `MIRSERVER_ABI` in `src/server/CMakeLists.txt`
4. From the root of the project, run:

```
cmake --build <BUILD_DIRECTORY> --target generate-mirserver-symbols-map
```

5. Check that your symbols are reflected properly in the `symbols.map` file:

```
git diff src/server/symbols.map
```

If `MIRSERVER_ABI` needed to be updated, you should also run:

```
./tools/update_package_abis.sh
```

2.2.9 How to Specify Startup Applications

When starting a compositor or display server, you usually want to start a few other applications. For example you might want to start a background viewer, a terminal, and a status bar. Doing so manually like what we've do would be immensely annoying, so let's add an option for users to specify startup apps.

Note: *Write Your First Wayland Compositor* is a prerequisite for this how-to.

We'll start by creating an instance of `miral::ExternalClientLauncher`, this will provide a way to launch programs by name. Then, we'll create a function that will take the list of startup apps and launch them. Finally, we just need to pass the external client launcher and our configuration option to `run_with()` so that they're registered with the compositor.

```
 #include <miral/runner.h>
+#include <miral/configuration_option.h>
+#include <miral/external_client.h>
 #include <miral/minimal_window_manager.h>
 #include <miral/set_window_management_policy.h>

@@ -9,8 +10,22 @@ int main(int argc, char const* argv[])
 {
     MirRunner runner{argc, argv};

+    miral::ExternalClientLauncher external_client_launcher;
+
+    auto run_startup_apps = [&](std::vector<std::string> const& apps)
+    {
+        for(auto const& app : apps)
+        {
+            external_client_launcher.launch(std::vector<std::string>{app});
+        }
+    };
+
     return runner.run_with(
         {
-            set_window_management_policy<MinimalWindowManager>()
+            set_window_management_policy<MinimalWindowManager>(),
+            external_client_launcher,
+            miral::ConfigurationOption{run_startup_apps, "startup-app", "App to run at␣
↪startup (can be specified multiple times)"},
         });
 }
```

Now, to start `kgx` and `bomber` on startup, we can do:

```
./build/demo-mir-compositor --startup-app kgx --startup-app bomber
```

2.2.10 How to Handle Keyboard Input

This how-to guide will show you the basics of handling keyboard input in Mir based compositors.

Handling keyboard input allows for great flexibility in your compositor. For example, it's widely known that ALT + F4 closes the currently open application. Some window managers take this a step further by allowing all navigation between windows to be done via the keyboard.

We'll implement a couple of shortcuts:

- CTRL + ALT + T / CTRL + ALT + SHIFT + T: To launch a terminal emulator
- CTRL + ALT + Backspace: To stop the compositor

For ease of reading (and copying), we'll split the changes into three parts: headers, code, and options.

Note: *Write Your First Wayland Compositor* is a prerequisite for this how-to.

Header changes

Here we just include `append_event_filter.h` for the declaration of `AppendEventFilter` and `toolkit_event` to get definitions for event types and functions. We import everything from `miral::toolkit` to make the code a bit easier to read.

```
@@ -1,10 +1,15 @@
 #include <miral/runner.h>
+#include <miral/append_event_filter.h>
 #include <miral/minimal_window_manager.h>
 #include <miral/set_window_management_policy.h>
+#include <miral/toolkit_event.h>

 using namespace miral;
+using namespace miral::toolkit;
```

Code changes

This big block of code can be broken down into three parts:

1. Declaring the name of the terminal we'll use and an external client launcher that we'll use to launch it.
2. Filtering Mir events until we obtain a keyboard key down event.
3. Handling the combinations we want.

```
+    std::string terminal_cmd{"kgx"};
+    miral::ExternalClientLauncher external_client_launcher;
+
+    auto const builtin_keybinds = [&](MirEvent const* event)
+        {
+            // Skip non-input events
+            if (mir_event_get_type(event) != mir_event_type_input)
+                return false;
```

(continues on next page)

(continued from previous page)

```
+
+            // Skip non-key input events
+            MirInputEvent const* input_event = mir_event_get_input_event(event);
+            if (mir_input_event_get_type(input_event) != mir_input_event_type_key)
+                return false;
+
+            // Skip anything but down presses
+            MirKeyboardEvent const* kev = mir_input_event_get_keyboard_event(input_
↪event);
+            if (mir_keyboard_event_action(kev) != mir_keyboard_action_down)
+                return false;
+
+            // CTRL + ALT must be pressed
+            MirInputEventModifiers mods = mir_keyboard_event_modifiers(kev);
+            if (!(mods & mir_input_event_modifier_alt) || !(mods & mir_input_event_
↪modifier_ctrl))
+                return false;
+
+            switch (mir_keyboard_event_keysym(kev))
+            {
+            // Exit program
+            case XKB_KEY_BackSpace:
+                runner.stop();
+                return true;
+
+            // Launch terminal
+            case XKB_KEY_t:
+            case XKB_KEY_T:
+                external_client_launcher.launch({terminal_cmd});
+                return false;
+
+            // Do nothing
+            default:
+                return false;
+            };
+        };
```

Options changes

Here we simply add the external client launcher we declared in the previous subsection, and an event filter that uses the code we specified in the previous subsection.

```
     return runner.run_with(
         {
             set_window_management_policy<MinimalWindowManager>(),
+            external_client_launcher,
+            miral::AppendEventFilter{builtin_keybinds}
         });
 }
```

This can be easily extended to read keybinds from config files or control various other aspects of the compositor. The only limit here is your imagination.

2.2.11 Specifying CSD/SSD Preference

Clients can ask the compositor to use server or client side decorations, or request the compositor to choose for them. *miral::Decorations* allows you to customize how the server deals with these requests.

This how-to will show you how to specify the behavior of the compositor regarding these requests.

Note: *How to Specify Startup Apps* is a prerequisite for this how-to.

To make the compositor prefer server side decorations when the client doesn't specify a preference, you only need to change two lines:

```
@@ -1,6 +1,7 @@
 #include <miral/runner.h>
 #include <miral/configuration_option.h>
+#include <miral/decorations.h>
 #include <miral/external_client.h>
 #include <miral/minimal_window_manager.h>
 #include <miral/set_window_management_policy.h>
@@ -74,5 +75,6 @@ int main(int argc, char const* argv[])
             external_client_launcher,
             miral::ConfigurationOption{run_startup_apps, "startup-app", "App to run at␣
↪startup (can be specified multiple times)"},
+            miral::Decorations::prefer_ssd(),
         });
 }
```

That's it! You can now build and run your compositor and try running a Wayland compatible application to see how its decorations change:

```
./build/demo-mir-compositor --startup-app kgx --startup-app bomber
```

MirAL also has other strategies: *miral::Decorations::prefer_csd()*, *miral::Decorations::always_ssd()*, and *miral::Decorations::always_csd()*. Try playing around with different strategies and seeing how they behave differently.

2.3 Explanation

These pages provide additional detail about a number of aspects related to using Mir.

- *Architecture*: an overview of Mir's architecture for contributors
- *Libraries*: an overview of Mir's libraries and how they depend on one another
- *Graphics support*: what's required to run Mir compositors
- *Security*: a deep dive into the security aspects of Mir compositors
- *What is Wayland anyway?*: so what is it?
- *Windowing paradigms*: how are windows managed?
- *Component reports*: information on Mir's debug and performance reporting infrastructure

2.3.1 Architecture

This document introduces the architecture of *Mir* at a high-level.

Audience

This document is intended to provide contributors to *Mir* an overview of *Mir*'s systems. It is *not* intended to guide compositor authors.

APIs for compositor authors

Mir provides compositor authors with a set of libraries that they can use to build Wayland based shells. These libraries are:

- *miral*
- *miroil*
- *mirwayland*

Miral

The most commonly used library is **miral**. `miral` (the "*Mir* Abstraction Layer") is an API that makes *Mir* easy for compositor authors to work with. It provides core window management concepts and an interface that is more ABI stable than Mir's internal API. While `miral` is built on the Mir *engine*, compositor authors are encouraged to only interact with the high-level `miral` library.

Miroil

miroil is a custom library for the Lomiri folks. It is like `miral` in that it provides an abstraction over the Mir *engine*. However, most compositor authors will not interact with `miroil`.

Mirwayland

Compositor authors may want to define their own wayland protocol extensions in addition to the ones that the core *Mir* implementation defines. The `mirwayland` library satisfies this requirement. This library may be used in conjunction with either `miral` or `miroil`.

The Mir engine

The **mirserver** library is the core implementation of *Mir*. It serves as the engine for both `miral` and `miroil`. This library does the heavy-lifting for the compositor and, as a result, is the largest piece of *Mir*. This section will explain the primary concepts that drive the engine.

Core Concepts

At the heart of `mirserver` are two interfaces: the **Shell** and the **Scene**. The `Shell` is responsible for fielding requests from the rest system. The `Shell` modifies the state of the `Scene` to reflect the requested changes.

For example, the `Frontend` would ask the `Shell` to initiate dragging a window. The `Shell` would then decide how to move that window to and update the state of the `Scene` to reflect that change.

From Scene to Screen

Knowing that the `Scene` holds the state of what is to be displayed, we can talk about the **Compositor**. The `Compositor` gets the collection of items to render from the `Scene`, renders them with the help of the *rendering platform*, and sends them off to the *display platform* to be displayed.

From Interaction to Shell

As stated previously, the `Shell` handles requests from the system and updates the state of the `Scene` accordingly. These requests come from a variety of sources, which we will investigate now.

Frontend Wayland: Responsible for connecting the Wayland protocol with the rest of *Mir*. The `WaylandConnector` class connects requests made via the Wayland protocol to the core state of the compositor.

Frontend XWayland: Responsible for connecting requests sent by an `XWayland` server to the rest of *Mir*. The `XWaylandConnector` establishes this connection. This frontend spawns an `XWayland` server as a subprocess.

Input: Handles everything having to do with input, including mouse, keyboard, touch, and more. This system interacts with the specific *input platform* and bubbles up events through a list of `InputDispatcher`s, which enables different pieces of the software to react to events.

For example, a compositor's window manager may respond to a key press event by opening up a new terminal via a request to the `Shell`.

Platforms

We briefly hinted at the existence of so-called "platforms" previously, but they are deserving of a dedicated section. A **Platform** is an adapter that allows the system to work across different graphics, input, and rendering stacks. They come in three flavors:

- **Display Platform**: Determines what the compositor is rendering to. This may be a physical monitor via GBM/KMS (or EGLStreams for Nvidia), an X11 or Wayland window, or a completely virtual buffer.
- **Input Platform**: Determines where the compositor is getting input from. This could be native event via `libinput`, X input events, or Wayland input events.
- **Rendering Platform**: Determines how the compositor renders the final image. For now, only a GL backend is supported.

The GBM/KMS platform is most typically what will be used, as it is the native platform. The X11 platform is useful for development. The Wayland platform is specifically useful for Ubuntu Touch, where they are hosting *Mir* in another Wayland compositor.

Supporting Libraries

Mir leans on a few core libraries to support the entire system. These libraries contain data structures and utilities that are shared throughout the project, including `miral` and `miroil`.

- **Core**: Fundamental data concepts, like file descriptors and rectangles. These data structures tend not to be *Mir*-specific.
- **Common**: *Mir*-specific data concepts like *Mir* event building, logging, and timing utilities.

2.3.2 Libraries

The Mir project is a collection of C++ libraries for writing Wayland compositors. This document describes what those libraries are and how they depend on one another.

Public Libraries

The following libraries are intended for published for public consumption:

- `miral`
- `mircommon`
- `mircore`
- `miroil`
- `mirplatform`
- `mirserver`
- `mirserverlttng`
- `mirwayland`

When you build the project, these libraries are found in `<build_directory>/lib`.

Additionally, Mir publishes libraries that add support for different platforms. These can be found in `<build_directory>/lib/server-modules`. These libraries are loaded at runtime by Mir and provide access to the underlying graphics and input hardware of that platform. The libraries that Mir provides are:

- `graphics-gbm-kms`
- `graphics-eglstream-kms`
- `graphics-wayland`
- `graphics-dummy`
- `server-x11`
- `server-virtual`
- `renderer-egl-generic`
- `input-evdev`
- `input-stub`

Dependency Graph

In the following diagram, each arrow denotes that the library at the start of the arrow depends on the library at the end of the arrow.

There are a few things of note in this diagram:

- `miral` and `mircore` are intended to be the primary user-facing libraries
- `miroil` is a special compatibility layer for the Lomiri project is not intended for general consumption
- `mirserver` depends on the most libraries as it is the brains of the operation
- `mircommon` provide common functionalities for every other library in the system
- The input and graphics platforms (represented as `input_X` and `graphics_X` in the diagram) are dynamically loaded by `mirserver` when the engine starts up.

2.3.3 Mir graphics support

This document describes the requirements for running Mir.

Mir supports graphics and input "platforms" by using modules loadable at runtime. This means that applications built with Mir can run across a range of graphics stacks.

There are Mir platforms supplied with Mir and the potential to develop more. There is also one platform we know of developed by a third party.

The most widely used and tested platform is "mesa-kms" which works with the Mesa collection of open-source graphics drivers. That's used on Intel-graphics based desktops and a variety of SBCs and other devices.

Summary list of current Mir graphics platforms

Platform	Status	Description
gbm-kms	Released	Works with any driver providing KMS, `libgbm` and an EGL supporting EGL_WL_bind_wayland_display. The open-source Mesa drivers are the obvious example, but other drivers supporting these interfaces should work - for example, the binary MALI drivers provided by ARM have been tested on some devices.
x11	Released	Provides "Mir-on-X11" primarily for development.
eglstre kms	Released	Works with proprietary Nvidia drivers.
wayland	Released	Provides "Mir-on-Wayland" both for nested compositors and for development.
android	Released	3rd party (UBports) works with a libhybris container for Android drivers. (Typically for phone hardware with Android-only drivers.)
dispmanx	Released	Works with Broadcom proprietary DispmanX API.

Each of these platforms depends on having the corresponding kernel and userspace drivers installed on the system. On startup Mir will "probe" the system for support of the above and select the best supported platform.

Choosing hardware and drivers

When configuring a system for use with Mir it is not only necessary to check the hardware and drivers for compatibility with Mir, but also with the application software.

For example, if the application software uses 3D or video acceleration, then that needs to be supported by the application (possibly through a supporting library) on the selected drivers and hardware. It is very possible that the support differs between the open-source drivers and the proprietary ones. Video decoding, in particular, varies a lot between graphics stacks.

There's a further complication in that Mir expects applications to use Wayland and some features on some drivers don't work with Wayland. (An example is that the intel-vaapi-driver for Ubuntu 18.04 doesn't work with Wayland. That is fixed in more recent versions, but may need to be addressed for core18 snaps.)

Also, there are applications that do not support Wayland directly. These can be supported with Xwayland - a program that translates from X11 to Wayland for the application. But that leads to further limitations as the GL acceleration support through Xwayland depends upon the graphics drivers. It works with Intel on Mesa, but beyond that it is patchy.

Some hardware examples

RPi 3

GPU: VideoCore IV

Mesa open-source graphics stack	Proprietary driver
VC4 - open source kernel/mesa stack.	Dispmanx - Broadcom proprietary(ish) stack.
Mir works, and is tested on this.	Mir works, not yet incorporated into CI lab testing
Supports GL.	Requires out-of-tree patches to https://github.com/raspberrypi/userland to enable 3D and video decode clients.
Does not support OMX.May support some video encode/decode via mmal interface. This is apparently slower than OMX.	Potentially higher performance (particularly OMX for video decode)
As a mesa/gbm-based platform, would expect 3D to work in XWayland.Mmal may work in XWayland.	As a non-mesa/gbm platform would likely not have 3D (or video decode) in XWayland.

i.MX6

GPU: Vivante GC something (varies by model)

Mesa open-source graphics stack	Proprietary driver
Etnaviv - Full open-source stack, using standard KMS/dma-buf/gbm interfaces. Reverse-engineered.*As it's using a full open-source stack, this would be easiest to support.*	Vivante - proprietary driver.
Mir would use same platform as on the desktop - gbm-kms. *We've tested the Boundary Devices i.MX6 on classic Ubuntu. Ubuntu Core would require some enablement work (adding the etnaviv driver).*	Would require writing a Mir platform (this is clearly possible; there are patches for Weston to support vivante)
Supports 3D (mesa GL) + video decoding (CODA v4l2)	Supports 3D + video decoding
Performance may be an issue (for example, https://github.com/Igalia/meta-webkit/issues/13)	Performance may be better; supported by downstream projects (again, cf: https://github.com/Igalia/meta-webkit/wiki/i.MX6)
Notably - the open source stack should provide 3D acceleration (and potentially video acceleration) in XWayland.	Support for 3D in XWayland is unknown; would likely require significant out-of-tree patches.

i.MX8

GPU: Vivante GC7000

Mesa open-source graphics stack	Proprietary driver
Etnaviv - Full open-source stack, using standard KMS/dma-buf/gbm interfaces. Reverse-engineered.	Vivante - proprietary driver.
Same comments as etnaviv on i.MX6 apply, but the GPU is newer and the driver support is likewise newer; there may be more missing features/bugs than i.MX6 support.	same comments as i.MX6. *It's likely that a Mir vivante platform would work on both i.MX6 and i.MX8; likewise, there is apparently Weston support (again, in the form of out-of-tree patches)*
Looks like there might not be open-source video decode support (cf: https://community.nxp.com/thread/489829#comment-1160206). Unknown whether we could interface the IMX bits with the etnaviv DRM bits.	Proprietary video decode solution.

Driver requirements

Different Mir platforms require different features of the underlying driver stack. The features needed to enable a given Mir platform are:

gbm-kms

The gbm-kms platform requires a DRM device node with KMS support, a `libgbm` implementation for buffer allocation, and an EGL implementation supporting at least EGL_KHR_platform_gbm (or the equivalent EGL_MESA_platform_gbm extension) and EGL_WL_bind_wayland_display.

Optionally, the EGL implementation can support EGL_EXT_image_dma_buf_import, EGL_EXT_image_dma_buf_import_modifiers, and use the zwp_linux_dmabuf_unstable_v1 Wayland protocol for client buffer transfer. Support for this was added in Mir 2.3. Composite-bypass support depends on this implementation. Some future performance improvements, such as overlay plane usage, are likely to require this support from the driver stack.

eglstream-kms

The eglstream-kms platform requires a DRM device node with Atomic KMS support and an EGL implementation supporting EGL_EXT_platform_base, EGL_EXT_platform_device, EGL_EXT_device_base, EGL_EXT_device_drm, and EGL_EXT_output_base.

2.3.4 Security

This document aims to explore in depth the security considerations around Mir based compositors.

Threat Model

Mir is a C++ library for building compositors, not a product itself. As such, when discussing the threat model for Mir, it is useful to discuss it in terms of an actual product that is built on Mir. With this in mind, we will define the threat model of **Ubuntu Frame** in this document.

Ubuntu Frame Threat Model Diagram

Ubuntu Frame is published as a snap. As such, the threat model for frame assumes that the snap is secure, and proceeds to outline the frame snap's interactions with the outside world.

Cryptography

There is no cryptography used in Mir, no direct dependency on en/decryption, hashing or digital signatures.

2.3.5 OK, so what is this Wayland thing anyway?

Or: things are complicated. Let's see if one more explanation will incrementally reduce the Internet's Wrongness™

We've recently (ok, recently-ish) released Mir 1.0 with usable Wayland support. Yay!

That brought a bunch of publicity, including on LWN. Some of the comments there and elsewhere betray a misunderstanding about what Wayland *is* (and is not), and this still occasionally comes up in `#wayland`, so I'll dust off an old blog post, polish up the rusty bits, and see if I can make this clearer for people again!

I'll give a run-down as to what the various projects in this space are and aim to do, throwing in X11 as a reference point.

Wayland, Mir, and X - different projects, with different goals

X11, and X.org

Everyone's familiar with their friendly neighbourhood X server. This is what we've currently got as the default desktop Linux display server. For the purposes of this blog post, X consists of:

The X11 protocol

You're all familiar with the X11 protocol, right? This gentle beast specifies how to talk to an X server - both the binary format of the messages you'll send and receive, and what you can expect the server to do with any given message (the semantics). There are also plenty of protocol extensions; new messages to make the server do new things, like handle more than one monitor in a non-stupid way.

The X11 client libraries

No-one actually fiddles with X by sending raw binary data down a socket; they use the client libraries - the modern XCB, or the boring old Xlib (also known as `libX11.so.6`). They do the boring work of throwing binary data at the server and present the client with a more civilised view of the X server, one where you can just `XOpenDisplay(NULL)` and start doing stuff.

Actually, the above is a bit of a lie. Almost all the time people don't even use XCB or Xlib, they use toolkits such as GTK+ or Qt, and *they* use XLib or XCB.

The Xorg server

This would be the bit most obviously associated with X - the one, the only, the X.org X server! This is the actual `/usr/bin/X` display server we all know and love. Although there are other implementations of X11, this is all you'll ever see on the free desktop. Or on OS X, for that matter.

So that's our baseline stack - a protocol, one or more client libraries, a display server implementation. How about Wayland and Mir?

Wayland

The Wayland protocol

The Wayland protocol is, like the X11 protocol, a definition of the binary data you can expect to send and receive over a Wayland socket, and the semantics associated with those binary bits. This is handled a bit differently to X11; the protocol is specified in XML, which is processed by a scanner and turned into C code. There is a binary protocol, and you can *technically* implement that protocol without using the `wayland-scanner`-generated code, but since the Wayland EGL platform is defined in terms of the `wl_display*` structure from `libwayland` your implementation will need to be ABI compatible with the existing implementation in order for 3D accelerated clients to work.

Also different from X11 is that everything's treated as an extension - you deal with all the interfaces in the core protocol the same way as you deal with any extensions you create. And you create a lot of extensions - for example, the core protocol doesn't have any buffer-passing mechanism other than SHM, so there's an extension for drm buffers in Mesa. Likewise, the NVIDIA drivers define their own internal protocol and a protocol for the compositor to support. The Weston reference compositor also has a bunch of extensions, both for ad-hoc things like the compositor<->desktop-shell interface, and for things like XWayland.

The Wayland client library

Or `libwayland`. A bit like XCB and Xlib, this is basically just an IPC library. Unlike XCB and Xlib it can also used by a Wayland server for server→client communication. Also unlike XCB and Xlib, it's programmatically generated from the protocol definition. It's quite a nice IPC library, really. Like XCB and Xlib, you're not really expected to use this, anyway; you're expected to use a toolkit like Qt or GTK+, and EGL + your choice of Khronos drawing API if you want to be funky. There's also a library for reading X cursor themes in there.

The Wayland server?

This is where it diverges; there *is no* Wayland server in the sense that there's an Xorg server. There's Weston, the reference compositor, which is growing `libweston` to do some custom shell work and is used in various places. There's `mutter`, the GNOME window manager which has grown a Wayland server implementation. Likewise, there's `kwin`, KDE's window manager that grew a Wayland server implementation. There's also a swathe of lesser-known display-server-compositor-window-manager projects, like sway.

Of course, there's also now anything based on Mir . Unity8 and EGMDE are both now Wayland servers!

Desktop environments are expected to write their own Wayland server, using the protocol and client libraries, and they have.

Mir

And so we get to Mir.

Where the Wayland libraries are all about IPC, Mir is about producing a library to do the drudge work of a display-server-compositor-window-manager-thing, so in this way it's more like Xorg than Wayland.

Mir *does* still have a (not-guaranteed-stable) protocol and client library in `libmirclient` - the Wayland server implementation is an *additional* client frontend to Mir's core - but `libmirclient` is deprecated and we will not be pursuing further development of it. Supporting the standard desktop Wayland window management extensions means much less work for the Mir team - we only need to focus on the Mir code itself, not *also* on an EGL platform + associated Mesa patches and GTK+ patches and Qt patches and SDL patches and…

2.3.6 Window positions under Wayland

Preamble

Design is all about balance, and the design of graphical user interfaces has a lot of interests to be balanced. In the context of Wayland, the principle interested parties are the designer of the compositor/window manager/shell [server] and the designer of the application toolkit/application [client]. Wayland is the "language" these two parties use to communicate.

A recent thread on the wayland-devel mailing list highlighted once again that these interests are not identical. The thread focuses on the placement of windows and that is what I want to address here.

The designers

For the designer of a server it is clear that applications ought to be "well behaved" and not be allowed to place windows in arbitrary positions. There are even security concerns about placing windows to intercept input such as passwords. And only the server can provide consistent window behaviours across multiple applications and toolkits.

For the designer of a client it is clear that the toolkit is best placed to decide where windows are best placed. And only the client can provide consistent window behaviours across multiple servers and operating systems.

The conflict

Both sides have a point that they should be "in control", but clearly that isn't going to happen.

Legacy windowing systems such as X11 allowed clients complete freedom, and the issues that caused were among the motivations for doing things differently in Wayland.

Throughout the design of Wayland the approach has been for the client to make requests and the server to make the final decisions.

Obviously, that approach leads to frustration on the part of designers of clients as they are losing some of the flexibility they are accustomed to.

The balance

Although a lot of freedom is indeed being lost, when looking at applications in the real world they only use a small fraction of that and for some specific cases. That's not surprising, as they are supporting some visual "idioms" that users recognise because they are shared across applications.

1. Regular or "top level" windows These are handled by `xdg-shell`'s `xdg_toplevel`.
2. Dialog windows These are handled by `xdg-shell`'s `xdg_toplevel` with the "parent" set.
3. Menu and Popup windows These are handled by `xdg-shell`'s `xdg_popup` with a "grab"
4. Tooltip and hover windows These are handled by `xdg-shell`'s `xdg_popup`.
5. Satellite and toolbox windows These are not currently handled by a Wayland extension, but could be added in a way consistent with the support for other window types.

Conclusion

While client developers (of applications and toolkits) may be surprised that they can't set absolute geometry there are a number of objections to allowing it:

1. preventing placement of windows to interfere with elements of the shell such as docks;
2. allowing for how the shell is laid out (docs, multi-screen display walls, etc);
3. no need for application to know how its own windows are laid out

The approach taken by Wayland supports most of the windowing idioms recognised by users and can be extended to cover more without allowing clients complete freedom.

2.3.7 Mir component reports

Both the server library and the client library include facilities to provide debugging and tracing information at runtime. This is achieved through component reports, which are sets of interesting events provided by many Mir components. A component report can be usually handled in a number of different ways, configured using command-line options and/or environment variables. By default, component reports are turned off.

Server reports

The way component reports are handled on the server can be configured using either command-line options or environment variables. The environment variables are prefixed with `MIR_SERVER_` and contain underscores ('_') instead of dashes ('-'). The available component reports and handlers for the server are:

Environment variable	Command line option	Handlers
MIR_SERVER_COMPOSITOR_REPORT	–compositor-report	log,lttng
MIR_SERVER_DISPLAY_REPORT	–display-report	log,lttng
MIR_SERVER_INPUT_REPORT	–input-report	log,lttng
MIR_SERVER_LEGACY_INPUT_REPORT	–legacy-input-report	log
MIR_SERVER_SEAT_REPORT	–seat-report	log
MIR_SERVER_SCENE_REPORT	–scene-report	log,lttng
MIR_SERVER_SHARED_LIBRARY_PROBER_REPORT	–shared-library-prober-report	log,lttng

For example, to enable the LTTng input report, one could either use the `--input-report=lttng` command-line option to the server, or set the `MIR_SERVER_INPUT_REPORT=lttng` environment variable.

LTTng support

Mir provides LTTng tracepoints for various interesting events. You can enable LTTng tracing for a Mir component by using the corresponding command-line option or environment variable for that component's report:

```
$ lttng create mirsession -o /tmp/mirsession
$ lttng enable-event -u -a
$ lttng start
$ mir_demo_server --compositor-report=lttng
$ lttng stop
$ babeltrace /tmp/mirsession/<trace-subdir>
```

LTTng-UST versions up to and including 2.1.2, and up to and including 2.2-rc2 contain a bug (lttng #538) that prevents event recording if the tracepoint provider is dlopen()-ed at runtime, like in the case of Mir. If you have a version of LTTng affected by this bug, you need to preload the server tracepoint provider library:

```
$ LD_PRELOAD=libmirserverlttng.so mir_demo_server --compositor-report=lttng
```

2.4 Reference

These pages provide detailed reference to the Mir's programming and other interfaces.

- *API*: Mir API reference
- *Introducing the MirAL API*: The primary external interface for compositor authors
- *Continuous integration*: A detailed guide through Mir's testing infrastructure
- *DSO Versioning guide*: How is ABI managed in the Mir project
- *Kernel requirements*: The kernel features required to run Mir-based compositors

2.4.1 Mir API

Page Hierarchy

- page_deprecated

Class Hierarchy

- ***Namespace mir***
  - ***Namespace mir::geometry***
    - ***Namespace mir::geometry::generic***
      - *Template Struct Displacement*
      - *Template Struct Point*
      - *Template Struct Rectangle*
      - *Template Struct Size*
      - *Template Struct Value*
    - *Struct DeltaXTag*
    - *Struct DeltaYTag*
    - *Struct HeightTag*
    - *Struct StrideTag*
    - *Struct WidthTag*
    - *Struct XTag*
    - *Struct YTag*
    - *Class Rectangles*
  - *Struct IntOwnedFd*

- *Enum @0*
- *Enum MirBufferFlag*
- *Enum MirDepthLayer*
- *Enum MirEdgeAttachment*
- *Enum MirEventType*
- *Enum MirFocusMode*
- *Enum MirFormFactor*
- *Enum MirInputEventModifier*
- *Enum MirInputEventType*
- *Enum MirKeyboardAction*
- *Enum MirLifecycleState*
- *Enum MirMirrorMode*
- *Enum MirOrientation*
- *Enum MirOrientationMode*
- *Enum MirOutputGammaSupported*
- *Enum MirOutputType*
- *Enum MirPixelFormat*
- *Enum MirPlacementGravity*
- *Enum MirPlacementHints*
- *Enum MirPointerAcceleration*
- *Enum MirPointerAction*
- *Enum MirPointerAxis*
- *Enum MirPointerAxisSource*
- *Enum MirPointerButton*
- *Enum MirPointerConfinementState*
- *Enum MirPointerHandedness*
- *Enum MirPowerMode*
- *Enum MirPromptSessionState*
- *Enum MirResizeEdge*
- *Enum MirShellChrome*
- *Enum MirSubpixelArrangement*
- *Enum MirTouchAction*
- *Enum MirTouchAxis*
- *Enum MirTouchpadClickMode*
- *Enum MirTouchpadScrollMode*
- *Enum MirTouchscreenMappingMode*

- *Enum MirTouchTooltype*
- *Enum MirWindowAttrib*
- *Enum MirWindowFocusState*
- *Enum MirWindowState*
- *Enum MirWindowType*
- *Enum MirWindowVisibility*

File Hierarchy

- **dir_examples**
  - **dir_examples_example-server-lib**
    - file_examples_example-server-lib_decoration_provider.h
    - file_examples_example-server-lib_floating_window_manager.h
    - file_examples_example-server-lib_splash_session.h
    - file_examples_example-server-lib_sw_splash.h
    - file_examples_example-server-lib_tiling_window_manager.h
    - file_examples_example-server-lib_wallpaper_config.h
    - file_examples_example-server-lib_wayland_app.h
    - file_examples_example-server-lib_wayland_shm.h
    - file_examples_example-server-lib_wayland_surface.h
  - **dir_examples_miral-kiosk**
    - file_examples_miral-kiosk_kiosk_window_manager.h
  - **dir_examples_miral-shell**
    - **dir_examples_miral-shell_spinner**
      - file_examples_miral-shell_spinner_eglapp.h
      - file_examples_miral-shell_spinner_miregl.h
      - file_examples_miral-shell_spinner_splash.h
- **dir_include**
  - **dir_include_core**
    - **dir_include_core_mir**
      - **dir_include_core_mir_geometry**

        file_include_core_mir_geometry_dimensions.h

        file_include_core_mir_geometry_displacement.h

        file_include_core_mir_geometry_forward.h

        file_include_core_mir_geometry_point.h

        file_include_core_mir_geometry_rectangle.h

        file_include_core_mir_geometry_rectangles.h

file_include_core_mir_geometry_size.h

- file_include_core_mir_abnormal_exit.h
- file_include_core_mir_anonymous_shm_file.h
- file_include_core_mir_depth_layer.h
- file_include_core_mir_fatal.h
- file_include_core_mir_fd.h
- file_include_core_mir_int_wrapper.h
- file_include_core_mir_optional_value.h
- file_include_core_mir_proof_of_mutex_lock.h
- file_include_core_mir_shm_file.h
- file_include_core_mir_synchronised.h

* **dir_include_core_mir_toolkit**
  - **dir_include_core_mir_toolkit_events**

    file_include_core_mir_toolkit_events_enums.h
  - file_include_core_mir_toolkit_common.h
  - file_include_core_mir_toolkit_mir_input_device_types.h
  - file_include_core_mir_toolkit_mir_native_buffer.h
  - file_include_core_mir_toolkit_mir_version_number.h

– **dir_include_miral**

* **dir_include_miral_miral**
  - file_include_miral_miral_add_init_callback.h
  - file_include_miral_miral_append_event_filter.h
  - file_include_miral_miral_application.h
  - file_include_miral_miral_application_authorizer.h
  - file_include_miral_miral_application_info.h
  - file_include_miral_miral_canonical_window_manager.h
  - file_include_miral_miral_command_line_option.h
  - file_include_miral_miral_config_file.h
  - file_include_miral_miral_configuration_option.h
  - file_include_miral_miral_cursor_theme.h
  - file_include_miral_miral_custom_renderer.h
  - file_include_miral_miral_decorations.h
  - file_include_miral_miral_display_configuration.h
  - file_include_miral_miral_display_configuration_option.h
  - file_include_miral_miral_external_client.h
  - file_include_miral_miral_idle_listener.h

- file_include_miral_miral_input_configuration.h
- file_include_miral_miral_internal_client.h
- file_include_miral_miral_keymap.h
- file_include_miral_miral_lambda_as_function.h
- file_include_miral_miral_minimal_window_manager.h
- file_include_miral_miral_output.h
- file_include_miral_miral_prepend_event_filter.h
- file_include_miral_miral_runner.h
- file_include_miral_miral_session_lock_listener.h
- file_include_miral_miral_set_command_line_handler.h
- file_include_miral_miral_set_terminator.h
- file_include_miral_miral_set_window_management_policy.h
- file_include_miral_miral_toolkit_event.h
- file_include_miral_miral_version.h
- file_include_miral_miral_wayland_extensions.h
- file_include_miral_miral_window.h
- file_include_miral_miral_window_info.h
- file_include_miral_miral_window_management_options.h
- file_include_miral_miral_window_management_policy.h
- file_include_miral_miral_window_manager_tools.h
- file_include_miral_miral_window_specification.h
- file_include_miral_miral_x11_support.h
- file_include_miral_miral_zone.h

– **dir_include_miroil**

* **dir_include_miroil_miroil**

- file_include_miroil_miroil_compositor.h
- file_include_miroil_miroil_display_configuration_controller_wrapper.h
- file_include_miroil_miroil_display_configuration_policy.h
- file_include_miroil_miroil_display_configuration_storage.h
- file_include_miroil_miroil_display_id.h
- file_include_miroil_miroil_display_listener_wrapper.h
- file_include_miroil_miroil_edid.h
- file_include_miroil_miroil_event_builder.h
- file_include_miroil_miroil_eventdispatch.h
- file_include_miroil_miroil_input_device.h
- file_include_miroil_miroil_input_device_observer.h

- file_include_miroil_miroil_mir_prompt_session.h
- file_include_miroil_miroil_mir_server_hooks.h
- file_include_miroil_miroil_mirbuffer.h
- file_include_miroil_miroil_open_gl_context.h
- file_include_miroil_miroil_persist_display_config.h
- file_include_miroil_miroil_prompt_session_listener.h
- file_include_miroil_miroil_prompt_session_manager.h
- file_include_miroil_miroil_set_compositor.h
- file_include_miroil_miroil_surface.h
- file_include_miroil_miroil_surface_observer.h

Full API

Namespaces

Namespace mir

Namespaces

- *Namespace mir::geometry*
- *Namespace mir::renderer*

Classes

- *Struct IntOwnedFd*
- *Class AbnormalExit*
- *Class AnonymousShmFile*
- *Class ExitWithOutput*
- *Class FatalErrorStrategy*
- *Class Fd*
- *Template Class IntWrapper*
- *Template Class optional_value*
- *Class ProofOfMutexLock*
- *Class ShmFile*
- *Template Class Synchronised*
- *Template Class Synchronised::LockedImpl*

Functions

- *Function mir::fatal_error_abort*
- *Function mir::fatal_error_except*
- *Function mir::mir_depth_layer_get_index*
- *Template Function mir::operator!=(optional_value<T> const&, optional_value<T> const&)*
- *Template Function mir::operator!=(optional_value<T> const&, T const&)*
- *Template Function mir::operator!=(T const&, optional_value<T> const&)*
- *Template Function mir::operator<<*
- *Template Function mir::operator==(optional_value<T> const&, T const&)*
- *Template Function mir::operator==(optional_value<T> const&, optional_value<T> const&)*
- *Template Function mir::operator==(T const&, optional_value<T> const&)*

Typedefs

- *Typedef mir::EventUPtr*

Variables

- *Variable mir::fatal_error*

Namespace mir::geometry

Basic geometry types. Types for dimensions, displacements, etc. and the operations that they support.

Namespaces

- *Namespace mir::geometry::generic*

Classes

- *Struct DeltaXTag*
- *Struct DeltaYTag*
- *Struct HeightTag*
- *Struct StrideTag*
- *Struct WidthTag*
- *Struct XTag*
- *Struct YTag*
- *Class Rectangles*

Functions

- *Template Function mir::geometry::as_delta(generic::X<T> const&)*
- *Template Function mir::geometry::as_delta(generic::Height<T> const&)*
- *Template Function mir::geometry::as_delta(generic::Width<T> const&)*
- *Template Function mir::geometry::as_delta(generic::Y<T> const&)*
- *Template Function mir::geometry::as_height(generic::DeltaY<T> const&)*
- *Template Function mir::geometry::as_height(generic::Y<T> const&)*
- *Template Function mir::geometry::as_width(generic::X<T> const&)*
- *Template Function mir::geometry::as_width(generic::DeltaX<T> const&)*
- *Template Function mir::geometry::as_x(generic::Width<T> const&)*
- *Template Function mir::geometry::as_x(generic::DeltaX<T> const&)*
- *Template Function mir::geometry::as_y(generic::Height<T> const&)*
- *Template Function mir::geometry::as_y(generic::DeltaY<T> const&)*
- *Function mir::geometry::operator<<*

Typedefs

- *Typedef mir::geometry::DeltaX*
- *Typedef mir::geometry::DeltaXF*
- *Typedef mir::geometry::DeltaY*
- *Typedef mir::geometry::DeltaYF*
- *Typedef mir::geometry::Displacement*
- *Typedef mir::geometry::DisplacementD*
- *Typedef mir::geometry::DisplacementF*
- *Typedef mir::geometry::Height*
- *Typedef mir::geometry::HeightF*
- *Typedef mir::geometry::Point*
- *Typedef mir::geometry::PointD*
- *Typedef mir::geometry::PointF*
- *Typedef mir::geometry::Rectangle*
- *Typedef mir::geometry::RectangleD*
- *Typedef mir::geometry::RectangleF*
- *Typedef mir::geometry::Size*
- *Typedef mir::geometry::SizeD*
- *Typedef mir::geometry::SizeF*
- *Typedef mir::geometry::Stride*

- *Typedef mir::geometry::Width*
- *Typedef mir::geometry::WidthF*
- *Typedef mir::geometry::X*
- *Typedef mir::geometry::XF*
- *Typedef mir::geometry::Y*
- *Typedef mir::geometry::YF*

Namespace mir::geometry::generic

Classes

- *Template Struct Displacement*
- *Template Struct Point*
- *Template Struct Rectangle*
- *Template Struct Size*
- *Template Struct Value*

Functions

- *Template Function mir::geometry::generic::as_displacement(Size<T> const&)*
- *Template Function mir::geometry::generic::as_displacement(Point<T> const&)*
- *Template Function mir::geometry::generic::as_point(Size<T> const&)*
- *Template Function mir::geometry::generic::as_point(Displacement<T> const&)*
- *Template Function mir::geometry::generic::as_size(Displacement<T> const&)*
- *Template Function mir::geometry::generic::as_size(Point<T> const&)*
- *Template Function mir::geometry::generic::intersection_of*
- *Template Function mir::geometry::generic::operator*(Width<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator*(Scalar, Width<T> const&)*
- *Template Function mir::geometry::generic::operator*(Size<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator*(Height<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator*(Scalar, Height<T> const&)*
- *Template Function mir::geometry::generic::operator*(DeltaX<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator*(Scalar, DeltaX<T> const&)*
- *Template Function mir::geometry::generic::operator*(Scalar, Size<T> const&)*
- *Template Function mir::geometry::generic::operator*(DeltaY<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator*(Scalar, DeltaY<T> const&)*
- *Template Function mir::geometry::generic::operator*(Displacement<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator*(Scalar, Displacement<T> const&)*

- *Template Function mir::geometry::generic::operator+(Y<T>, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator+(Point<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+(Point<T> const&, Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator+(DeltaX<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+(Width<T>, Width<T>)*
- *Template Function mir::geometry::generic::operator+(Displacement<T> const&, Point<T> const&)*
- *Template Function mir::geometry::generic::operator+(Width<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+(Point<T>, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator+(DeltaY<T>, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator+(Height<T>, Height<T>)*
- *Template Function mir::geometry::generic::operator+(Displacement<T> const&, Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator+(X<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+(Height<T>, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator+=(Height<T>&, Height<T>)*
- *Template Function mir::geometry::generic::operator+=(Point<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+=(Point<T>&, Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator+=(Point<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator+=(Width<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+=(X<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+=(DeltaY<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator+=(Height<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator+=(DeltaX<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator+=(Width<T>&, Width<T>)*
- *Template Function mir::geometry::generic::operator+=(Y<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-(Width<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-(Point<T> const&, Point<T> const&)*
- *Template Function mir::geometry::generic::operator-(DeltaY<T>, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-(X<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-(X<T>, X<T>)*
- *Template Function mir::geometry::generic::operator-(Height<T>, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-(DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-(Y<T>, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-(Y<T>, Y<T>)*
- *Template Function mir::geometry::generic::operator-(DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-(Height<T>, Height<T>)*
- *Template Function mir::geometry::generic::operator-(Point<T>, DeltaY<T>)*

- *Template Function mir::geometry::generic::operator-(Width<T>, Width<T>)*
- *Template Function mir::geometry::generic::operator-(Point<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-(Point<T> const&, Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator-(Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator-(Displacement<T> const&, Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator-(DeltaX<T>, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-=(DeltaX<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-=(Y<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-=(Point<T>&, Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator-=(DeltaY<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-=(Point<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-=(X<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator-=(Point<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-=(Height<T>&, DeltaY<T>)*
- *Template Function mir::geometry::generic::operator-=(Width<T>&, DeltaX<T>)*
- *Template Function mir::geometry::generic::operator/(X<T> const&, Width<U> const&)*
- *Template Function mir::geometry::generic::operator/(Y<T> const&, Height<U> const&)*
- *Template Function mir::geometry::generic::operator/(Height<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator/(Size<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator/(DeltaX<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator/(Width<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator/(DeltaY<T> const&, Scalar)*
- *Template Function mir::geometry::generic::operator<*
- *Template Function mir::geometry::generic::operator<<(std::ostream&, Value<T, Tag> const&)*
- *Template Function mir::geometry::generic::operator<<(std::ostream&, Point<T> const&)*
- *Template Function mir::geometry::generic::operator<<(std::ostream&, Displacement<T> const&)*
- *Template Function mir::geometry::generic::operator<<(std::ostream&, Rectangle<T> const&)*
- *Template Function mir::geometry::generic::operator<<(std::ostream&, Size<T> const&)*

Typedefs

- *Typedef mir::geometry::generic::DeltaX*
- *Typedef mir::geometry::generic::DeltaY*
- *Typedef mir::geometry::generic::Height*
- *Typedef mir::geometry::generic::Width*
- *Typedef mir::geometry::generic::X*
- *Typedef mir::geometry::generic::Y*

Variables

- *Variable mir::geometry::generic::is_exactly_representable*

Namespace mir::renderer

Namespace miral

Mir Abstraction Layer.

Detailed Description

A thin, hopefully ABI stable, layer over the Mir libraries exposing only those abstractions needed to write a shell. One day this may inspire a core Mir library.

Namespaces

- *Namespace miral::detail*
- *Namespace miral::toolkit*

Classes

- *Struct ApplicationInfo*
- *Struct FdHandle*
- *Struct Output::PhysicalSizeMM*
- *Struct WaylandExtensions::Builder*
- *Struct WindowInfo*
- *Struct WindowManagerOption*
- *Struct WindowSpecification::AspectRatio*
- *Class AddInitCallback*
- *Class AppendEventFilter*
- *Class ApplicationAuthorizer*
- *Class ApplicationCredentials*
- *Class BasicSetApplicationAuthorizer*
- *Class CanonicalWindowManagerPolicy*
- *Class ConfigFile*
- *Class ConfigurationOption*
- *Class CursorTheme*
- *Class CustomRenderer*
- *Class Decorations*

- *Class DisplayConfiguration*
- *Class ExternalClientLauncher*
- *Class IdleListener*
- *Class InputConfiguration*
- *Class InputConfiguration::Mouse*
- *Class InputConfiguration::Touchpad*
- *Class InternalClientLauncher*
- *Class Keymap*
- *Class MinimalWindowManager*
- *Class MirRunner*
- *Class Output*
- *Class PrependEventFilter*
- *Class SessionLockListener*
- *Template Class SetApplicationAuthorizer*
- *Class SetCommandLineHandler*
- *Class SetTerminator*
- *Class SetWindowManagementPolicy*
- *Class StartupInternalClient*
- *Class WaylandExtensions*
- *Class WaylandExtensions::Context*
- *Class WaylandExtensions::EnableInfo*
- *Class Window*
- *Class WindowManagementPolicy*
- *Class WindowManagerOptions*
- *Class WindowManagerTools*
- *Class WindowSpecification*
- *Class X11Support*
- *Class Zone*

Functions

- *Template Function miral::add_window_manager_policy*
- *Function miral::application_for(wl_resource *)*
- *Function miral::application_for(wl_client *)*
- *Function miral::apply_lifecycle_state_to*
- *Function miral::display_configuration_options*
- *Function miral::equivalent_display_area*

- *Function miral::kill*
- *Template Function miral::lambda_as_function*
- *Function miral::name_of*
- *Function miral::operator!=(Window const&, std::shared_ptr<mir::scene::Surface> const&)*
- *Function miral::operator!=(Output::PhysicalSizeMM const&, Output::PhysicalSizeMM const&)*
- *Function miral::operator!=(std::shared_ptr<mir::scene::Surface> const&, Window const&)*
- *Function miral::operator!=(Window const&, Window const&)*
- *Function miral::operator<*
- *Function miral::operator<=*
- *Function miral::operator==(Window const&, std::shared_ptr<mir::scene::Surface> const&)*
- *Function miral::operator==(std::shared_ptr<mir::scene::Surface> const&, Window const&)*
- *Function miral::operator==(Window const&, Window const&)*
- *Function miral::operator==(Output::PhysicalSizeMM const&, Output::PhysicalSizeMM const&)*
- *Function miral::operator>*
- *Function miral::operator>=*
- *Function miral::pid_of*
- *Function miral::pre_init*
- *Function miral::PrintTo*
- *Template Function miral::set_window_management_policy*
- *Function miral::socket_fd_of*
- *Function miral::window_for*

Typedefs

- *Typedef miral::Application*
- *Typedef miral::BufferStreamId*
- *Typedef miral::CommandLineOption*
- *Typedef miral::WindowManagementPolicyBuilder*

Namespace miral::detail

Classes

- *Template Struct FunctionType*
- *Template Struct FunctionType< Return(Lambda::*)(Arg…) const >*
- *Template Struct FunctionType< Return(Lambda::*)(Arg…) >*

Namespace miral::toolkit

Functions

- *Function miral::toolkit::mir_event_get_input_event*
- *Function miral::toolkit::mir_event_get_type*
- *Function miral::toolkit::mir_input_event_get_event*
- *Function miral::toolkit::mir_input_event_get_event_time*
- *Function miral::toolkit::mir_input_event_get_keyboard_event*
- *Function miral::toolkit::mir_input_event_get_pointer_event*
- *Function miral::toolkit::mir_input_event_get_touch_event*
- *Function miral::toolkit::mir_input_event_get_type*
- *Function miral::toolkit::mir_keyboard_event_action*
- *Function miral::toolkit::mir_keyboard_event_input_event*
- *Function miral::toolkit::mir_keyboard_event_key_text*
- *Function miral::toolkit::mir_keyboard_event_keysym*
- *Function miral::toolkit::mir_keyboard_event_modifiers*
- *Function miral::toolkit::mir_keyboard_event_scan_code*
- *Function miral::toolkit::mir_pointer_event_action*
- *Function miral::toolkit::mir_pointer_event_axis_value*
- *Function miral::toolkit::mir_pointer_event_button_state*
- *Function miral::toolkit::mir_pointer_event_buttons*
- *Function miral::toolkit::mir_pointer_event_input_event*
- *Function miral::toolkit::mir_pointer_event_modifiers*
- *Function miral::toolkit::mir_touch_event_action*
- *Function miral::toolkit::mir_touch_event_axis_value*
- *Function miral::toolkit::mir_touch_event_id*
- *Function miral::toolkit::mir_touch_event_input_event*
- *Function miral::toolkit::mir_touch_event_modifiers*
- *Function miral::toolkit::mir_touch_event_point_count*
- *Function miral::toolkit::mir_touch_event_tooltype*

Namespace miroil

Classes

- *Struct DisplayConfigurationOptions*
- *Struct DisplayConfigurationOptions::DisplayMode*
- *Struct DisplayId*
- *Struct Edid*
- *Struct Edid::Descriptor*
- *Struct Edid::PhysicalSizeMM*
- *Class Compositor*
- *Class DisplayConfigurationControllerWrapper*
- *Class DisplayConfigurationPolicy*
- *Class DisplayConfigurationStorage*
- *Class DisplayListenerWrapper*
- *Class EventBuilder*
- *Class EventBuilder::EventInfo*
- *Class GLBuffer*
- *Class InputDevice*
- *Class InputDeviceObserver*
- *Class MirPromptSession*
- *Class MirServerHooks*
- *Class OpenGLContext*
- *Class PersistDisplayConfig*
- *Class PromptSessionListener*
- *Class PromptSessionManager*
- *Class SetCompositor*
- *Class Surface*
- *Class SurfaceObserver*

Functions

- *Function miroil::dispatch_input_event*

Typedefs

- *Typedef miroil::CompositorID*
- *Typedef miroil::CreateNamedCursor*
- *Typedef miroil::OutputId*

Unions

- *Union Descriptor::Value*

Namespace std

STL namespace.

Classes

- *Template Struct hash< ::mir::IntWrapper< Tag, ValueType > >*

Functions

- *Specialized Template Function std::swap*

Namespace wallpaper

Functions

- *Function wallpaper::font_file(std::string const&)*
- *Function wallpaper::font_file()*

Classes and Structs

Struct DeltaXTag

- Defined in file_include_core_mir_geometry_forward.h

Struct Documentation

struct **DeltaXTag**

Struct DeltaYTag

- Defined in file_include_core_mir_geometry_forward.h

Struct Documentation

struct **DeltaYTag**

Template Struct Displacement

- Defined in file_include_core_mir_geometry_displacement.h

Struct Documentation

template<typename **T**>

struct **Displacement**

Public Types

using **ValueType** = *T*

Public Functions

inline constexpr **Displacement**()

constexpr **Displacement**(*Displacement* const&) = default

Displacement &**operator=**(*Displacement* const&) = default

template<typename **U**>
inline explicit constexpr **Displacement**(*Displacement*<*U*> const &other) noexcept

template<typename **DeltaXType**, typename **DeltaYType**>
inline constexpr **Displacement**(*DeltaXType* &&dx, *DeltaYType* &&dy)

template<typename **Q** = *T*>
inline constexpr std::enable_if<std::is_integral<*Q*>::value, long long>::type **length_squared**() const

template<typename **Q** = *T*>
inline constexpr std::enable_if<!std::is_integral<*Q*>::value, *T*>::type **length_squared**() const

Public Members

DeltaX<*T*> **dx**

DeltaY<*T*> **dy**

Friends

friend bool **operator==**(*Displacement* const &lhs, *Displacement* const &rhs) = default

friend bool **operator!=**(*Displacement* const &lhs, *Displacement* const &rhs) = default

Template Struct Point

- Defined in file_include_core_mir_geometry_point.h

Struct Documentation

template<typename **T**>

struct **Point**

Public Types

using **ValueType** = *T*

Public Functions

constexpr **Point**() = default

constexpr **Point**(*Point* const&) = default

Point &**operator=**(*Point* const&) = default

template<typename **U**>
inline explicit constexpr **Point**(*Point*<*U*> const &other) noexcept

template<typename **XType**, typename **YType**>
inline constexpr **Point**(*XType* &&x, *YType* &&y)

Public Members

X<*T*> **x**

Y<*T*> **y**

Friends

friend bool **operator==**(*Point* const &lhs, *Point* const &rhs) = default

friend bool **operator!=**(*Point* const &lhs, *Point* const &rhs) = default

Template Struct Rectangle

- Defined in file_include_core_mir_geometry_rectangle.h

Struct Documentation

template<typename **T**>

struct **Rectangle**

Public Functions

constexpr **Rectangle**() = default

inline constexpr **Rectangle**(*Point*<*T*> const &top_left, *Size*<*T*> const &size)

inline *Point*<*T*> **bottom_right**() const

The bottom right boundary point of the rectangle.

Note that the returned point is *not* included in the rectangle area, that is, the rectangle is represented as [top_left,bottom_right).

inline *Point*<*T*> **top_right**() const

inline *Point*<*T*> **bottom_left**() const

inline bool **contains**(*Point*<*T*> const &p) const

inline bool **contains**(*Rectangle*<*T*> const &r) const

Test if the rectangle contains another.

Note that an empty rectangle can still contain other empty rectangles, which are treated as points or lines of thickness zero.

inline bool **overlaps**(*Rectangle*<*T*> const &r) const

inline *X*<*T*> **left**() const

inline *X*<*T*> **right**() const

inline *Y*<*T*> **top**() const

inline *Y*<*T*> **bottom**() const

Public Members

Point<*T*> **top_left**

Size<*T*> **size**

Friends

friend bool **operator==**(*Rectangle* const &lhs, *Rectangle* const &rhs) = default

friend bool **operator!=**(*Rectangle* const &lhs, *Rectangle* const &rhs) = default

Template Struct Size

- Defined in file_include_core_mir_geometry_size.h

Struct Documentation

template<typename **T**>

struct **Size**

Public Types

using **ValueType** = *T*

Public Functions

inline constexpr **Size**() noexcept

constexpr **Size**(*Size* const&) noexcept = default

Size &**operator=**(*Size* const&) noexcept = default

template<typename U> inline requires constexpr is_exactly_representable< T, U > Size (Size< U > const &other) noexcept

template<typename **U**>
inline explicit constexpr **Size**(*Size*<*U*> const &other) noexcept

template<typename **WidthType**, typename **HeightType**>
inline constexpr **Size**(*WidthType* &&width, *HeightType* &&height) noexcept

Public Members

Width<*T*> **width**

Height<*T*> **height**

Friends

friend bool **operator==**(*Size* const &lhs, *Size* const &rhs) = default

friend bool **operator!=**(*Size* const &lhs, *Size* const &rhs) = default

Template Struct Value

- Defined in file_include_core_mir_geometry_dimensions.h

Struct Documentation

template<typename **T**, typename **Tag**>

struct **Value**

Wraps a geometry value and prevents it from being accidentally used for invalid operations (such as setting a width to a height or adding two x positions together). Of course, explicit casts are possible to get around these restrictions (see the as_*() functions).

Public Types

using **ValueType** = *T*

using **TagType** = *Tag*

Public Functions

template<typename **Q** = *T*>
inline constexpr std::enable_if<std::is_integral<*Q*>::value, int>::type **as_int**() const

template<typename **Q** = *T*>
inline constexpr std::enable_if<std::is_integral<*Q*>::value, uint32_t>::type **as_uint32_t**() const

inline constexpr *T* **as_value**() const noexcept

inline constexpr **Value**() noexcept

inline *Value* &**operator=**(*Value* const &that) noexcept

inline constexpr **Value**(*Value* const &that) noexcept

template<typename **U**>

inline explicit constexpr **Value**(*Value*<*U*, *Tag*> const &value) noexcept

template<typename **U**, typename std::enable_if<std::is_scalar<*U*>::value, bool>::type = true>
inline explicit constexpr **Value**(*U* const &value) noexcept

Protected Attributes

T **value**

Friends

friend auto **operator<=>**(*Value* const &lhs, *Value* const &rhs) = default

Struct HeightTag

- Defined in file_include_core_mir_geometry_forward.h

Struct Documentation

struct **HeightTag**

Struct StrideTag

- Defined in file_include_core_mir_geometry_forward.h

Struct Documentation

struct **StrideTag**

Struct WidthTag

- Defined in file_include_core_mir_geometry_forward.h

Struct Documentation

struct **WidthTag**

These tag types determine what type of dimension a value holds and what operations are possible with it. They are only used as template parameters, are never instantiated and should only require forward declarations, but some compiler versions seem to fail if they aren't given real declarations.

Struct XTag

- Defined in file_include_core_mir_geometry_forward.h

Struct Documentation

struct **XTag**

Struct YTag

- Defined in file_include_core_mir_geometry_forward.h

Struct Documentation

struct **YTag**

Struct IntOwnedFd

- Defined in file_include_core_mir_fd.h

Struct Documentation

struct **IntOwnedFd**

Public Members

int **int_owned_fd**

Struct ApplicationInfo

- Defined in file_include_miral_miral_application_info.h

Struct Documentation

struct **ApplicationInfo**

Public Functions

ApplicationInfo()

explicit **ApplicationInfo**(*Application* const &app)

~ApplicationInfo()

ApplicationInfo(*ApplicationInfo* const &that)

auto **operator=**(*ApplicationInfo* const &that) -> miral::*ApplicationInfo*&

auto **name**() const -> std::string

auto **application**() const -> *Application*

auto **windows**() const -> std::vector<*Window*>&

auto **userdata**() const -> std::shared_ptr<void>

This can be used by client code to store window manager specific information.

void **userdata**(std::shared_ptr<void> userdata)

Template Struct FunctionType

- Defined in file_include_miral_miral_lambda_as_function.h

Struct Documentation

template<class **F**>

struct **FunctionType**

Template Struct FunctionType< Return(Lambda::*)(Arg...) const >

- Defined in file_include_miral_miral_lambda_as_function.h

Struct Documentation

template<typename **Lambda**, typename **Return**, typename ...**Arg**>

struct **FunctionType**<*Return* (*Lambda*::*)(*Arg*...) const>

Public Types

using **type** = std::function<*Return*(*Arg*...)>

Template Struct FunctionType< Return(Lambda::*)(Arg...) >

- Defined in file_include_miral_miral_lambda_as_function.h

Struct Documentation

template<typename **Lambda**, typename **Return**, typename ...**Arg**>

struct **FunctionType**<*Return* (*Lambda*::*)(*Arg*...)>

Public Types

using **type** = std::function<*Return*(*Arg*...)>

Struct FdHandle

- Defined in file_include_miral_miral_runner.h

Struct Documentation

struct **FdHandle**

A handle which keeps a file descriptor registered to the main loop until it is dropped.

Public Functions

virtual **~FdHandle**()

Struct Output::PhysicalSizeMM

- Defined in file_include_miral_miral_output.h

Nested Relationships

This struct is a nested type of *Class Output*.

Struct Documentation

struct **PhysicalSizeMM**

Public Members

int **width**

int **height**

Struct WaylandExtensions::Builder

- Defined in file_include_miral_miral_wayland_extensions.h

Nested Relationships

This struct is a nested type of *Class WaylandExtensions*.

Struct Documentation

struct **Builder**

A *Builder* creates and registers an extension protocol.

> **Remark**
>
> Since MirAL 2.5

Public Members

std::string **name**

Name of the protocol extension's Wayland global.

std::function< std::shared_ptr< void >Context const *context)> build

Functor that creates and registers an extension protocol.

Param context

giving access to:

- the wl_display (so that, for example, the extension can be registered); and,
- allowing server initiated code to be executed on the Wayland mainloop.

Return
a shared pointer to the implementation. (Mir will manage the lifetime)

Struct WindowInfo

- Defined in file_include_miral_miral_window_info.h

Struct Documentation

struct **WindowInfo**

Unnamed Group

auto **min_width**() const -> mir::geometry::*Width*

These constrain the sizes a window may be resized to (both interactively and pragmatically). Clients can request a min/max size, but it can be overridden by the window management policy. By default, minimum values are 0 and maximum values are `std::numeric_limits<int>::max()`.

auto **min_height**() const -> mir::geometry::*Height*

auto **max_width**() const -> mir::geometry::*Width*

auto **max_height**() const -> mir::geometry::*Height*

Unnamed Group

auto **width_inc**() const -> mir::geometry::*DeltaX*

These control the size increments of the window. This is used in cases like a terminal that can only be resized character-by-character. Current Wayland protocols do not support this property, so it generally wont be requested by clients. By default, both are 1.

auto **height_inc**() const -> mir::geometry::*DeltaY*

Unnamed Group

auto **min_aspect**() const -> *AspectRatio*

These constrain the possible aspect ratio of the window. Current Wayland protocols to not support this property, so it generally wont be requested by clients. By default, min_aspect is `{0U, std::numeric_limits<unsigned>::max()}` and max_aspect is `{std::numeric_limits<unsigned>::max(), 0U}`.

auto **max_aspect**() const -> *AspectRatio*

Unnamed Group

auto **application_id**() const -> std::string

The D-bus service name and basename of the app's .desktop file See http://standards.freedesktop.org/desktop-entry-spec/.

> **Remark**
>
> Since MirAL 2.8

Unnamed Group

static bool **needs_titlebar**(*MirWindowType* type)

Deprecated:

Obsolete: *Window::size()* includes decorations

Public Types

using **AspectRatio** = *WindowSpecification*::*AspectRatio*

Public Functions

WindowInfo()

WindowInfo(*Window* const &window, *WindowSpecification* const &params)

~WindowInfo()

explicit **WindowInfo**(*WindowInfo* const &that)

WindowInfo &**operator=**(*WindowInfo* const &that)

bool **can_be_active**() const

bool **can_morph_to**(*MirWindowType* new_type) const

bool **must_have_parent**() const

bool **must_not_have_parent**() const

bool **is_visible**() const

void **constrain_resize**(mir::geometry::*Point* &requested_pos, mir::geometry::*Size* &requested_size) const

auto **window**() const -> *Window*&

auto **name**() const -> std::string

auto **type**() const -> *MirWindowType*

auto **state**() const -> *MirWindowState*

auto **restore_rect**() const -> mir::geometry::*Rectangle*

auto **parent**() const -> *Window*

auto **children**() const -> std::vector<*Window*> const&

bool **has_output_id**() const

auto **output_id**() const -> int

auto **preferred_orientation**() const -> *MirOrientationMode*

auto **confine_pointer**() const -> *MirPointerConfinementState*

auto **shell_chrome**() const -> *MirShellChrome*

auto **userdata**() const -> std::shared_ptr<void>

This can be used by client code to store window manager specific information.

void **userdata**(std::shared_ptr<void> userdata)

inline void **swap**(*WindowInfo* &rhs)

auto **depth_layer**() const -> *MirDepthLayer*

auto **attached_edges**() const -> *MirPlacementGravity*

Get the edges of the output that the window is attached to (only meaningful for windows in state mir_window_state_attached)

auto **exclusive_rect**() const -> mir::*optional_value*<mir::geometry::*Rectangle*>

Mir will try to avoid occluding the area covered by this rectangle (relative to the window) (only meaningful when the window is attached to an edge)

auto **ignore_exclusion_zones**() const -> bool

Mir will ignore the exclusive_rects of other windows when this is set to true. (only meaningful when the window is attached to an edge)

auto **clip_area**() const -> mir::*optional_value*<mir::geometry::*Rectangle*>

Mir will not render anything outside this rectangle.

void **clip_area**(mir::*optional_value*<mir::geometry::*Rectangle*> const &area)

auto **focus_mode**() const -> *MirFocusMode*

How the window should gain and lose focus.

> **Remark**
>
> Since MirAL 3.3

auto **visible_on_lock_screen**() const -> bool

If this surface should be shown while the compositor is locked.

Remark

Since MirAL 3.9

Struct WindowManagerOption

- Defined in file_include_miral_miral_window_management_options.h

Struct Documentation

struct **WindowManagerOption**

Public Members

std::string const **name**

WindowManagementPolicyBuilder const **build**

Struct WindowSpecification::AspectRatio

- Defined in file_include_miral_miral_window_specification.h

Nested Relationships

This struct is a nested type of *Class WindowSpecification*.

Struct Documentation

struct **AspectRatio**

Public Members

unsigned **width**

unsigned **height**

Struct MirBufferPackage

- Defined in file_include_core_mir_toolkit_mir_native_buffer.h

Struct Documentation

struct **MirBufferPackage**

Public Members

int **data_items**

int **fd_items**

int **data**[*mir_buffer_package_max*]

int **width**

int **height**

int **fd**[*mir_buffer_package_max*]

int **unused0**

unsigned int **flags**

int **stride**

int **age**

Number of frames submitted by the client since the client has rendered to this buffer.

This has the same semantics as the EGL_EXT_buffer_age extension

See also

http://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_buffer_age.txt

Struct DisplayConfigurationOptions

- Defined in file_include_miroil_miroil_display_configuration_storage.h

Nested Relationships

Nested Types

- *Struct DisplayConfigurationOptions::DisplayMode*

Struct Documentation

struct **DisplayConfigurationOptions**

Public Members

mir::*optional_value*<bool> **used**

mir::*optional_value*<uint> **clone_output_index**

mir::*optional_value*<*DisplayMode*> **mode**

mir::*optional_value*<*MirOrientation*> **orientation**

mir::*optional_value*<*MirFormFactor*> **form_factor**

mir::*optional_value*<float> **scale**

struct **DisplayMode**

Public Members

mir::geometry::*Size* **size**

double **refresh_rate** = {-1}

Struct DisplayConfigurationOptions::DisplayMode

- Defined in file_include_miroil_miroil_display_configuration_storage.h

Nested Relationships

This struct is a nested type of *Struct DisplayConfigurationOptions*.

Struct Documentation

struct **DisplayMode**

Public Members

mir::geometry::*Size* **size**

double **refresh_rate** = {-1}

Struct DisplayId

- Defined in file_include_miroil_miroil_display_id.h

Struct Documentation

struct **DisplayId**

Public Members

Edid **edid**

OutputId **output_id**

Struct Edid

- Defined in file_include_miroil_miroil_edid.h

Nested Relationships

Nested Types

- *Struct Edid::Descriptor*
- *Union Descriptor::Value*
- *Struct Edid::PhysicalSizeMM*

Struct Documentation

struct **Edid**

Public Functions

Edid &**parse_data**(std::vector<uint8_t> const&)

Public Members

std::string **vendor**

uint16_t **product_code** = {0}

uint32_t **serial_number** = {0}

PhysicalSizeMM **size** = {0, 0}

Descriptor **descriptors**[4]

struct **Descriptor**

Public Types

enum class **Type** : uint8_t

Values:

enumerator **timing_identifiers**

enumerator **white_point_data**

enumerator **monitor_name**

enumerator **monitor_limits**

enumerator **unspecified_text**

enumerator **serial_number**

enumerator **undefined**

Public Functions

std::string **string_value**() const

Public Members

Type **type** = {*Type*::*undefined*}

Value **value** = {{0}}

union **Value**

Public Members

char **monitor_name**[13]

char **unspecified_text**[13]

char **serial_number**[13]

struct **PhysicalSizeMM**

Public Members

int **width**

int **height**

Struct Edid::Descriptor

- Defined in file_include_miroil_miroil_edid.h

Nested Relationships

This struct is a nested type of *Struct Edid*.

Nested Types

- *Union Descriptor::Value*

Struct Documentation

struct **Descriptor**

Public Types

enum class **Type** : uint8_t

Values:

enumerator **timing_identifiers**

enumerator **white_point_data**

enumerator **monitor_name**

enumerator **monitor_limits**

enumerator **unspecified_text**

enumerator **serial_number**

enumerator **undefined**

Public Functions

std::string **string_value**() const

Public Members

Type **type** = {*Type*::*undefined*}

Value **value** = {{0}}

union **Value**

Public Members

char **monitor_name**[13]

char **unspecified_text**[13]

char **serial_number**[13]

Struct Edid::PhysicalSizeMM

- Defined in file_include_miroil_miroil_edid.h

Nested Relationships

This struct is a nested type of *Struct Edid*.

Struct Documentation

struct **PhysicalSizeMM**

Public Members

int **width**

int **height**

Template Struct hash< ::mir::IntWrapper< Tag, ValueType > >

- Defined in file_include_core_mir_int_wrapper.h

Struct Documentation

template<typename **Tag**, typename **ValueType**>

struct **hash**<::mir::*IntWrapper*<*Tag*, *ValueType*>>

Public Functions

inline constexpr std::size_t **operator()**(::mir::*IntWrapper*<*Tag*, *ValueType*> const &id) const

Public Members

std::hash<int> **self**

Class DecorationProvider

- Defined in file_examples_example-server-lib_decoration_provider.h

Class Documentation

class **DecorationProvider**

Public Functions

DecorationProvider()

~DecorationProvider()

void **operator()**(struct wl_display *display)

void **operator()**(std::weak_ptr<mir::scene::Session> const &session)

auto **session**() const -> std::shared_ptr<mir::scene::Session>

void **stop**()

bool **is_decoration**(miral::*Window* const &window) const

Class FloatingWindowManagerPolicy

- Defined in file_examples_example-server-lib_floating_window_manager.h

Inheritance Relationships

Base Type

- `public miral::MinimalWindowManager` (*Class MinimalWindowManager*)

Class Documentation

class **FloatingWindowManagerPolicy** : public miral::*MinimalWindowManager*

example event handling:

o Switch apps: Alt+Tab, tap or click on the corresponding window o Switch window: Alt+`, tap or click on the corresponding window o Move window: Alt-leftmousebutton drag (three finger drag) o Resize window: Alt-middle_button drag (three finger pinch) o Maximize/restore current window (to display size): Alt-F11 o Maximize/restore current window (to display height): Shift-F11 o Maximize/restore current window (to display width): Ctrl-F11 o Switch workspace .

… … … : Meta-Alt-[F1|F2|F3|F4] o Switch workspace taking active window: Meta-Ctrl-[F1|F2|F3|F4]

virtual bool **handle_pointer_event**(MirPointerEvent const *event) override

pointer event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_touch_event**(MirTouchEvent const *event) override

touch event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_keyboard_event**(MirKeyboardEvent const *event) override

keyboard event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

track events that affect titlebar

virtual void **advise_new_window**(miral::*WindowInfo* const &window_info) override

Notification that a window has been created.

Parameters

window_info – the window

virtual void **handle_window_ready**(miral::*WindowInfo* &window_info) override

notification that the first buffer has been posted

Parameters

window_info – the window

virtual void **advise_focus_gained**(miral::*WindowInfo* const &info) override

Notification that a window has gained focus.

Parameters

window_info – the window

virtual void **handle_modify_window**(miral::*WindowInfo* &window_info, miral::*WindowSpecification* const &modifications) override

request from client to modify the window specification.

Parameters

- **window_info** – the window
- **modifications** – the requested changes

> **Note**
>
> the request has already been validated against the type definition

Public Functions

FloatingWindowManagerPolicy(miral::*WindowManagerTools* const &tools, std::shared_ptr<*SplashSession*> const &spinner, miral::*InternalClientLauncher* const &launcher, std::function<void()> &shutdown_hook)

~FloatingWindowManagerPolicy()

virtual miral::*WindowSpecification* **place_new_window**(miral::*ApplicationInfo* const &app_info, miral::*WindowSpecification* const &request_parameters) override

Customize initial window placement.

Parameters

- **app_info** – the application requesting a new window
- **requested_specification** – the requested specification (updated with default placement)

Returns

the customized specification

Protected Static Attributes

static const int **modifier_mask** = mir_input_event_modifier_alt | mir_input_event_modifier_shift | mir_input_event_modifier_sym | mir_input_event_modifier_ctrl | mir_input_event_modifier_meta

Class KioskWindowManagerPolicy

- Defined in file_examples_miral-kiosk_kiosk_window_manager.h

Inheritance Relationships

Base Type

- `public miral::CanonicalWindowManagerPolicy` (*Class CanonicalWindowManagerPolicy*)

Class Documentation

class **KioskWindowManagerPolicy** : public miral::*CanonicalWindowManagerPolicy*

Public Functions

KioskWindowManagerPolicy(miral::*WindowManagerTools* const &tools, std::shared_ptr<*SplashSession*> const&)

virtual auto **place_new_window**(miral::*ApplicationInfo* const &app_info, miral::*WindowSpecification* const &request) -> miral::*WindowSpecification* override

Customize initial window placement.

Parameters

- **app_info** – the application requesting a new window
- **requested_specification** – the requested specification (updated with default placement)

Returns

the customized specification

virtual void **advise_focus_gained**(miral::*WindowInfo* const &info) override

Notification that a window has gained focus.

Parameters

window_info – the window

virtual bool **handle_keyboard_event**(MirKeyboardEvent const *event) override

keyboard event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_touch_event**(MirTouchEvent const *event) override

touch event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_pointer_event**(MirPointerEvent const *event) override

pointer event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual void **handle_modify_window**(miral::*WindowInfo* &window_info, miral::*WindowSpecification* const &modifications) override

request from client to modify the window specification.

Parameters

- **window_info** – the window
- **modifications** – the requested changes

Note

the request has already been validated against the type definition

virtual void **handle_request_move**(miral::*WindowInfo* &window_info, MirInputEvent const *input_event) override

request from client to initiate move

Parameters

- **window_info** – the window
- **input_event** – the requesting event

Note

the request has already been validated against the requesting event

virtual void **handle_request_resize**(miral::*WindowInfo* &window_info, MirInputEvent const *input_event, *MirResizeEdge* edge) override

request from client to initiate resize

Parameters

- **window_info** – the window
- **input_event** – the requesting event
- **edge** – the edge(s) being dragged

Note

the request has already been validated against the requesting event

virtual Rectangle **confirm_placement_on_display**(const miral::*WindowInfo* &window_info, *MirWindowState* new_state, Rectangle const &new_placement) override

Confirm (and optionally adjust) the placement of a window on the display.

Called when (re)placing fullscreen, maximized, horizontally maximised and vertically maximized windows to allow adjustment for decorations.

Parameters

- **window_info** – the window
- **new_state** – the new state
- **new_placement** – the suggested placement

Returns

the confirmed placement of the window

Class AbnormalExit

- Defined in file_include_core_mir_abnormal_exit.h

Inheritance Relationships

Base Type

- `public std::runtime_error`

Derived Type

- `public mir::ExitWithOutput` (*Class ExitWithOutput*)

Class Documentation

class **AbnormalExit** : public std::runtime_error

Indicates Mir should exit with an error message printed to stderr.

Subclassed by *mir::ExitWithOutput*

Public Functions

inline **AbnormalExit**(std::string const &what)

Class AnonymousShmFile

- Defined in file_include_core_mir_anonymous_shm_file.h

Inheritance Relationships

Base Type

- `public mir::ShmFile` (*Class ShmFile*)

Class Documentation

class **AnonymousShmFile** : public mir::*ShmFile*

Public Functions

AnonymousShmFile(size_t size)

~AnonymousShmFile() noexcept

virtual void ***base_ptr**() const override

virtual int **fd**() const override

Class ExitWithOutput

- Defined in file_include_core_mir_abnormal_exit.h

Inheritance Relationships

Base Type

- `public mir::AbnormalExit` (*Class AbnormalExit*)

Class Documentation

class **ExitWithOutput** : public mir::*AbnormalExit*

Indicates Mir should exit with the given output (such as help text) printed to stdout.

Public Functions

inline **ExitWithOutput**(std::string const &what)

Class FatalErrorStrategy

- Defined in file_include_core_mir_fatal.h

Class Documentation

class **FatalErrorStrategy**

Public Functions

inline explicit **FatalErrorStrategy**(void (*fatal_error_handler)(char const *reason, ...))

inline **~FatalErrorStrategy**()

Class Fd

- Defined in file_include_core_mir_fd.h

Class Documentation

class **Fd**

Public Functions

explicit **Fd**(int fd)

explicit **Fd**(*IntOwnedFd*)

Fd()

Fd(*Fd*&&)

Fd(*Fd* const&) = default

Fd &**operator=**(*Fd*)

operator int() const

Public Static Attributes

static int const **invalid** = {-1}

Friends

friend auto **close**(*Fd* const &fd) -> int = delete
Prevent accidental calling of ::close()

Class Rectangles

- Defined in file_include_core_mir_geometry_rectangles.h

Class Documentation

class **Rectangles**
A collection of rectangles (with possible duplicates).

Public Types

typedef std::vector<*Rectangle*>::const_iterator **const_iterator**

typedef std::vector<*Rectangle*>::size_type **size_type**

Public Functions

Rectangles()

Rectangles(std::initializer_list<*Rectangle*> const &rects)

void **add**(*Rectangle* const &rect)

void **remove**(*Rectangle* const &rect)
removes at most one matching rectangle

void **clear**()

Rectangle **bounding_rectangle**() const

void **confine**(*Point* &point) const

const_iterator **begin**() const

const_iterator **end**() const

size_type **size**() const

bool **operator==**(*Rectangles* const &rect) const

bool **operator!=**(*Rectangles* const &rect) const

Template Class IntWrapper

- Defined in file_include_core_mir_int_wrapper.h

Class Documentation

template<typename **Tag**, typename **ValueType** = int>

class **IntWrapper**

Public Functions

inline constexpr **IntWrapper**()

inline explicit constexpr **IntWrapper**(*ValueType* value)

inline constexpr *ValueType* **as_value**() const

Friends

friend auto **operator<=>**(*IntWrapper* const &lhs, *IntWrapper* const &rhs) = default

Template Class optional_value

- Defined in file_include_core_mir_optional_value.h

Class Documentation

template<typename **T**>

class **optional_value**

Public Functions

optional_value() = default

inline **optional_value**(*T* const &value)

inline *optional_value* &**operator=**(*T* const &value)

inline bool **is_set**() const

inline *T* const &**value**() const

inline *T* &**value**()

inline *T* **value_or**(*T* default_) const

inline *T* &&**consume**()

inline **operator bool**() const

Class ProofOfMutexLock

- Defined in file_include_core_mir_proof_of_mutex_lock.h

Class Documentation

class **ProofOfMutexLock**

A method can take an instance of this class by reference to require callers to hold a mutex lock, without requiring a specific type of lock.

Public Functions

inline **ProofOfMutexLock**(std::lock_guard<std::mutex> const&)

inline **ProofOfMutexLock**(std::unique_lock<std::mutex> const &lock)

ProofOfMutexLock(*ProofOfMutexLock* const&) = delete

ProofOfMutexLock **operator=**(*ProofOfMutexLock* const&) = delete

Class ShmFile

- Defined in file_include_core_mir_shm_file.h

Inheritance Relationships

Derived Type

- `public mir::AnonymousShmFile` (*Class AnonymousShmFile*)

Class Documentation

class **ShmFile**

Subclassed by *mir::AnonymousShmFile*

Public Functions

virtual **~ShmFile**() = default

virtual void ***base_ptr**() const = 0

virtual int **fd**() const = 0

Protected Functions

ShmFile() = default

ShmFile(*ShmFile* const&) = delete

ShmFile &**operator=**(*ShmFile* const&) = delete

Template Class Synchronised

- Defined in file_include_core_mir_synchronised.h

Nested Relationships

Nested Types

- *Template Class Synchronised::LockedImpl*

Class Documentation

template<typename **T**>

class **Synchronised**

An object that enforces unique access to the data it contains.

This behaves like a mutex which owns the data it guards, and can give out a smart-pointer-esque handle to lock and access it.

Template Parameters

T – The type of data contained

Public Types

using **Locked** = *LockedImpl*<*T*>

Smart-pointer-esque accessor for the protected data.

Ensures exclusive access to the referenced data.

> **Note**
>
> Instances of Locked must not outlive the *Synchronised* they are derived from.

using **LockedView** = *LockedImpl*<*T* const>

Smart-pointer-esque accessor for the protected data.

Provides const access to the protected data, with the guarantee that no changes to the data can be made while this handle is live.

> **Note**
>
> Instances of Locked must not outlive the *Synchronised* they are derived from.

Public Functions

Synchronised() = default

inline **Synchronised**(*T* &&initial_value)

Synchronised(*Synchronised* const&) = delete

Synchronised &**operator=**(*Synchronised* const&) = delete

inline auto **lock**() -> *Locked*

Lock the data and return an accessor.

Returns

A smart-pointer-esque accessor for the contained data. While code has access to a live Locked instance it is guaranteed to have unique access to the contained data.

inline auto **lock**() const -> *LockedView*

Lock the data and return an accessor.

Returns

A smart-pointer-esque accessor for the contained data. While code has access to a live Locked instance it is guaranteed to have unique access to the contained data.

template<typename **U**>

class **LockedImpl**

Smart-pointer-esque accessor for the protected data.

Ensures exclusive access to the referenced data.

> **Note**
>
> Instances of Locked must not outlive the *Synchronised* they are derived from.

Public Functions

inline **LockedImpl**(*LockedImpl* &&from) noexcept

~LockedImpl() = default

inline auto **operator***() const -> *U*&

inline auto **operator->**() const -> *U**

inline void **drop**()

Relinquish access to the data.

This prevents further access to the contained data through this handle, and allows other code to acquire access.

template<typename **Cv**, typename **Predicate**>

inline void **wait**(*Cv* &cv, *Predicate* stop_waiting)

Allows waiting for a condition variable.

The protected data may be accessed both in the predicate and after this method completes.

Template Class Synchronised::LockedImpl

- Defined in file_include_core_mir_synchronised.h

Nested Relationships

This class is a nested type of *Template Class Synchronised*.

Class Documentation

template<typename **U**>

class **LockedImpl**

Smart-pointer-esque accessor for the protected data.

Ensures exclusive access to the referenced data.

> **Note**
>
> Instances of Locked must not outlive the *Synchronised* they are derived from.

Public Functions

inline **LockedImpl**(*LockedImpl* &&from) noexcept

~LockedImpl() = default

inline auto **operator***() const -> *U*&

inline auto **operator->**() const -> *U**

inline void **drop**()

Relinquish access to the data.

This prevents further access to the contained data through this handle, and allows other code to acquire access.

template<typename **Cv**, typename **Predicate**>
inline void **wait**(*Cv* &cv, *Predicate* stop_waiting)

Allows waiting for a condition variable.

The protected data may be accessed both in the predicate and after this method completes.

Class AddInitCallback

- Defined in file_include_miral_miral_add_init_callback.h

Class Documentation

class **AddInitCallback**

Add a callback to be invoked when the server has been initialized, but before it starts. If multiple callbacks are added they will be invoked in the sequence added.

Public Types

using **Callback** = std::function<void()>

Public Functions

explicit **AddInitCallback**(*Callback* const &callback)

~AddInitCallback()

void **operator()**(mir::Server &server) const

Class AppendEventFilter

- Defined in file_include_miral_miral_append_event_filter.h

Class Documentation

class **AppendEventFilter**

Public Functions

explicit **AppendEventFilter**(std::function<bool(*MirEvent* const *event)> const &filter)

Append an event filter (after any existing filters, including the window manager). The supplied filter should return true if and only if it handles the event as filters later in the list will not be called.

> **Remark**
>
> the filter return type changed to bool in MirAL 3.6

void **operator()**(mir::Server &server)

Class ApplicationAuthorizer

- Defined in file_include_miral_miral_application_authorizer.h

Class Documentation

class **ApplicationAuthorizer**

Public Functions

ApplicationAuthorizer() = default

virtual **~ApplicationAuthorizer**() = default

ApplicationAuthorizer(*ApplicationAuthorizer* const&) = delete

ApplicationAuthorizer &**operator=**(*ApplicationAuthorizer* const&) = delete

virtual bool **connection_is_allowed**(*ApplicationCredentials* const &creds) = 0

virtual bool **configure_display_is_allowed**(*ApplicationCredentials* const &creds) = 0

virtual bool **set_base_display_configuration_is_allowed**(*ApplicationCredentials* const &creds) = 0

virtual bool **screencast_is_allowed**(*ApplicationCredentials* const &creds) = 0

virtual bool **prompt_session_is_allowed**(*ApplicationCredentials* const &creds) = 0

virtual bool **configure_input_is_allowed**(*ApplicationCredentials* const &creds) = 0

virtual bool **set_base_input_configuration_is_allowed**(*ApplicationCredentials* const &creds) = 0

Class ApplicationCredentials

- Defined in file_include_miral_miral_application_authorizer.h

Class Documentation

class **ApplicationCredentials**

Public Functions

ApplicationCredentials(mir::frontend::SessionCredentials const &creds)

pid_t **pid**() const

uid_t **uid**() const

gid_t **gid**() const

Class BasicSetApplicationAuthorizer

- Defined in file_include_miral_miral_application_authorizer.h

Inheritance Relationships

Derived Type

- `public miral::SetApplicationAuthorizer< Policy >` (*Template Class SetApplicationAuthorizer*)

Class Documentation

class **BasicSetApplicationAuthorizer**

Subclassed by *miral::SetApplicationAuthorizer< Policy >*

Public Functions

explicit **BasicSetApplicationAuthorizer**(std::function<std::shared_ptr<*ApplicationAuthorizer*>()> const &builder)

~BasicSetApplicationAuthorizer()

void **operator()**(mir::Server &server)

auto **the_application_authorizer**() const -> std::shared_ptr<*ApplicationAuthorizer*>

Class CanonicalWindowManagerPolicy

- Defined in file_include_miral_miral_canonical_window_manager.h

Inheritance Relationships

Base Type

- `public miral::WindowManagementPolicy` (*Class WindowManagementPolicy*)

Derived Type

- `public KioskWindowManagerPolicy` (*Class KioskWindowManagerPolicy*)

Class Documentation

class **CanonicalWindowManagerPolicy** : public miral::*WindowManagementPolicy*

Widely accepted defaults for window management.

Subclassed by *KioskWindowManagerPolicy*

Public Functions

explicit **CanonicalWindowManagerPolicy**(*WindowManagerTools* const &tools)

virtual auto **place_new_window**(*ApplicationInfo* const &app_info, *WindowSpecification* const &request_parameters) -> *WindowSpecification* override

Customize initial window placement.

Parameters

- **app_info** – the application requesting a new window
- **requested_specification** – the requested specification (updated with default placement)

Returns

the customized specification

virtual void **handle_window_ready**(*WindowInfo* &window_info) override

Tries to focus on the newly ready window.

virtual void **handle_modify_window**(*WindowInfo* &window_info, *WindowSpecification* const &modifications) override

Applies the requested modifications.

virtual void **handle_raise_window**(*WindowInfo* &window_info) override

Tries to focus on the newly ready window.

virtual void **advise_focus_gained**(*WindowInfo* const &info) override

Raises the window (and any children)

virtual auto **confirm_inherited_move**(*WindowInfo* const &window_info, Displacement movement) -> Rectangle override

Move the child window with the parent.

virtual auto **confirm_placement_on_display**(*WindowInfo* const &window_info, *MirWindowState* new_state, Rectangle const &new_placement) -> Rectangle override

Confirm (and optionally adjust) the placement of a window on the display.

Called when (re)placing fullscreen, maximized, horizontally maximised and vertically maximized windows to allow adjustment for decorations.

Parameters

- **window_info** – the window
- **new_state** – the new state
- **new_placement** – the suggested placement

Returns

the confirmed placement of the window

Protected Attributes

WindowManagerTools **tools**

Class ConfigFile

- Defined in file_include_miral_miral_config_file.h

Class Documentation

class **ConfigFile**

Utility to locate and monitor a configuration file via the XDG Base Directory Specification.

Vis: ($XDG_CONFIG_HOME or $HOME/.config followed by $XDG_CONFIG_DIRS). If, instead of a filename, a path is given, then the base directories are not applied.

If mode is `no_reloading`, then the file is loaded on startup and not reloaded

If mode is `reload_on_change`, then the file is loaded on startup and either the user-specific configuration file base ($XDG_CONFIG_HOME or $HOME/.config), or the supplied path is monitored for changes.

Remark

MirAL 5.1

Public Types

enum class **Mode**

Mode of reloading.

Values:

enumerator **no_reloading**

enumerator **reload_on_change**

using **Loader** = std::function<void(std::istream &istream, std::filesystem::path const &path)>

Loader functor is passed both the open stream and the actual path (for use in reporting problems)

Public Functions

ConfigFile(*MirRunner* &runner, std::filesystem::path file, *Mode* mode, *Loader* load_config)

~ConfigFile()

Class ConfigurationOption

- Defined in file_include_miral_miral_configuration_option.h

Class Documentation

class **ConfigurationOption**

Add a user configuration option to Mir's option handling. By default the callback will be invoked following Mir initialisation but prior to the server starting. The value supplied to the callback will come from the command line, environment variable, config file or the default.

> **Note**
>
> Except for re-ordering implied by "pre_init()" the callbacks will be invoked in the order supplied. \Remark: Renamed (from CommandLineOption) in MirAL 3.6

Public Functions

ConfigurationOption(std::function<void(int value)> callback, std::string const &option, std::string const &description, int default_value)

ConfigurationOption(std::function<void(double value)> callback, std::string const &option, std::string const &description, double default_value)

ConfigurationOption(std::function<void(std::string const &value)> callback, std::string const &option, std::string const &description, std::string const &default_value)

ConfigurationOption(std::function<void(std::string const &value)> callback, std::string const &option, std::string const &description, char const *default_value)

ConfigurationOption(std::function<void(bool value)> callback, std::string const &option, std::string const &description, bool default_value)

ConfigurationOption(std::function<void(mir::*optional_value*<int> const &value)> callback, std::string const &option, std::string const &description)

ConfigurationOption(std::function<void(mir::*optional_value*<std::string> const &value)> callback, std::string const &option, std::string const &description)

ConfigurationOption(std::function<void(mir::*optional_value*<bool> const &value)> callback, std::string const &option, std::string const &description)

ConfigurationOption(std::function<void(bool is_set)> callback, std::string const &option, std::string const &description)

ConfigurationOption(std::function<void(std::vector<std::string> const &values)> callback, std::string const &option, std::string const &description)

template<typename **Lambda**>
inline **ConfigurationOption**(*Lambda* &&callback, std::string const &option, std::string const &description)

void **operator()**(mir::Server &server) const

~ConfigurationOption()

ConfigurationOption(*ConfigurationOption* const&)

auto **operator=**(*ConfigurationOption* const&) -> *ConfigurationOption*&

Friends

friend auto **pre_init**(*ConfigurationOption* const &clo) -> *ConfigurationOption*

Update the option to be called back *before* Mir initialization starts.

Parameters

clo – the option

Class CursorTheme

- Defined in file_include_miral_miral_cursor_theme.h

Class Documentation

class **CursorTheme**

Load an X-cursor theme, either the supplied default, or through the —cursor-theme config option.

Public Functions

explicit **CursorTheme**(std::string const &theme)

Specify a default theme.

~CursorTheme()

void **operator()**(mir::Server &server) const

Class CustomRenderer

- Defined in file_include_miral_miral_custom_renderer.h

Class Documentation

class **CustomRenderer**

Public Types

using **Builder** = std::function<std::unique_ptr<mir::renderer::Renderer>(std::unique_ptr<mir::graphics::gl::OutputSurface>, std::shared_ptr<mir::graphics::GLRenderingProvider>)>

Public Functions

explicit **CustomRenderer**(*Builder* &&renderer)

void **operator()**(mir::Server &server) const

Class Decorations

- Defined in file_include_miral_miral_decorations.h

Class Documentation

class **Decorations**

Configures the window decoration strategy.

> **Remark**
>
> Since MirAL 5.1

> **Note**
>
> The strategy can only be applied to clients that are able to negotiate the decoration style with the server.

Public Functions

Decorations() = delete

Decorations(*Decorations* const&) = default

auto **operator=**(*Decorations* const&) -> *Decorations*& = default

void **operator()**(mir::Server&) const

Public Static Functions

static auto **always_ssd**() -> *Decorations*

Always use server side decorations regardless of the client's choice.

static auto **always_csd**() -> *Decorations*

Always use client side decorations regardless of the client's choice.

static auto **prefer_ssd**() -> *Decorations*

Prefer server side decorations if the client does not set a specific mode. Otherwise use the mode specified by the client.

static auto **prefer_csd**() -> *Decorations*

Prefer client side decorations if the client does not set a specific mode. Otherwise use the mode specified by the client.

Class DisplayConfiguration

- Defined in file_include_miral_miral_display_configuration.h

Class Documentation

class **DisplayConfiguration**

Enable display configuration. The config file (*miral::MirRunner::display_config_file()*) is located via the XDG Base Directory Specification. Vis: ($XDG_CONFIG_HOME or $HOME/.config followed by $XDG_CONFIG_DIRS)

Remark

Since MirAL 2.4

Note

From MirAL 3.8 will monitor the configuration file or, if none found, for the creation of a file in $XDG_CONFIG_HOME or $HOME/.config. Changes to this file will be reloaded. In addition, the selected layout may be overridden using a corresponding file: display_config_file() + "-layout" which will also be monitored and changes reloaded

Public Functions

explicit **DisplayConfiguration**(*MirRunner* const &mir_runner)

auto **layout_option**() -> *ConfigurationOption*

Provide the default 'display-layout' configuration option.

void **select_layout**(std::string const &layout)

Select a layout from the configuration.

auto **list_layouts**() -> std::vector<std::string>

List all layouts found in the config file.

void **add_output_attribute**(std::string const &key)

Enable a custom output attribute in the .display YAML.

> **Remark**
>
> Since MirAL 3.8

void **operator()**(mir::Server &server) const

~DisplayConfiguration()

DisplayConfiguration(*DisplayConfiguration* const&)

auto **operator=**(*DisplayConfiguration* const&) -> *DisplayConfiguration*&

Class ExternalClientLauncher

- Defined in file_include_miral_miral_external_client.h

Class Documentation

class **ExternalClientLauncher**

Public Functions

ExternalClientLauncher()

~ExternalClientLauncher()

void **operator()**(mir::Server &server)

auto **launch**(std::vector<std::string> const &command_line) const -> pid_t

Launch with client environment configured for Wayland. If X11 is enabled, then DISPLAY will also be set accordingly.

Remark

Return type changed from void in MirAL 3.0

Returns
The pid of the process that was launched.

auto **launch_using_x11**(std::vector<std::string> const &command_line) const -> pid_t

If X11 is enabled, then launch with client environment configured for X11. For the occasions it is desired to coerce applications into using X11.

Remark

Return type changed from void in MirAL 3.0

Returns
The pid of the process that was launched (or -1 if X11 is not enabled)

void **snapcraft_launch**(std::string const &desktop_file) const

Use the proposed `desktop-entry` snap interface to launch another snap.

Remark

Since MirAL 3.0

auto **launch**(std::string const &command) const -> pid_t

Launch with client environment configured for Wayland. If X11 is enabled, then DISPLAY will also be set accordingly.

Remark

Since MirAL 3.6

Returns
The pid of the process that was launched.

Public Static Functions

static auto **split_command**(std::string const &command) -> std::vector<std::string>

Split out the tokens of a (possibly escaped) command.

Remark

Since MirAL 3.6

Class IdleListener

- Defined in file_include_miral_miral_idle_listener.h

Class Documentation

class **IdleListener**

Public Types

using **Callback** = std::function<void()>

Public Functions

IdleListener()

~IdleListener() = default

IdleListener &**on_dim**(*Callback* const&)

IdleListener &**on_off**(*Callback* const&)

IdleListener &**on_wake**(*Callback* const&)

void **operator()**(mir::Server &server) const

Class InputConfiguration

- Defined in file_include_miral_miral_input_configuration.h

Nested Relationships

Nested Types

- *Class InputConfiguration::Mouse*
- *Class InputConfiguration::Touchpad*

Class Documentation

class **InputConfiguration**

Input configuration.

Allow servers to make input configuration changes at runtime

Remark

Since MirAL 5.1

Public Functions

InputConfiguration()

~InputConfiguration()

void **operator()**(mir::Server &server)

auto **mouse**() -> *Mouse*

void **mouse**(*Mouse* const &val)

auto **touchpad**() -> *Touchpad*

void **touchpad**(*Touchpad* const &val)

class **Mouse**

Input configuration for mouse pointer devices.

Remark

Since MirAL 5.1

Public Functions

Mouse()

~Mouse()

Mouse(*Mouse* const &that)

auto **operator=**(*Mouse* that) -> *Mouse*&

auto **handedness**() const -> std::optional<*MirPointerHandedness*>

auto **acceleration**() const -> std::optional<*MirPointerAcceleration*>

auto **acceleration_bias**() const -> std::optional<double>

auto **vscroll_speed**() const -> std::optional<double>

auto **hscroll_speed**() const -> std::optional<double>

void **handedness**(std::optional<*MirPointerHandedness*> const &val)

void **acceleration**(std::optional<*MirPointerAcceleration*> const &val)

void **acceleration_bias**(std::optional<double> const &val)

> **Note**
>
> val will be clamped to the range [-1.0, 1.0]

void **vscroll_speed**(std::optional<double> const &val)

void **hscroll_speed**(std::optional<double> const &val)

Friends

friend class InputConfiguration::Self

class **Touchpad**

Input configuration for touchpad devices.

> **Remark**
>
> Since MirAL 5.1

Public Functions

Touchpad()

~Touchpad()

Touchpad(*Touchpad* const &that)

auto **operator=**(*Touchpad* that) -> *Touchpad*&

auto **disable_while_typing**() const -> std::optional<bool>

auto **disable_with_external_mouse**() const -> std::optional<bool>

auto **acceleration**() const -> std::optional<*MirPointerAcceleration*>

auto **acceleration_bias**() const -> std::optional<double>

auto **vscroll_speed**() const -> std::optional<double>

auto **hscroll_speed**() const -> std::optional<double>

auto **click_mode**() const -> std::optional<*MirTouchpadClickMode*>

auto **scroll_mode**() const -> std::optional<*MirTouchpadScrollMode*>

auto **tap_to_click**() const -> std::optional<bool>

void **disable_while_typing**(std::optional<bool> const &val)

void **disable_with_external_mouse**(std::optional<bool> const &val)

void **acceleration**(std::optional<*MirPointerAcceleration*> const &val)

void **acceleration_bias**(std::optional<double> const &val)

> **Note**
>
> val will be clamped to the range [-1.0, 1.0]

void **vscroll_speed**(std::optional<double> const &val)

void **hscroll_speed**(std::optional<double> const &val)

void **click_mode**(std::optional<*MirTouchpadClickMode*> const &val)

void **scroll_mode**(std::optional<*MirTouchpadScrollMode*> const &val)

void **tap_to_click**(std::optional<bool> const &val)

Friends

friend class InputConfiguration::Self

Class InputConfiguration::Mouse

- Defined in file_include_miral_miral_input_configuration.h

Nested Relationships

This class is a nested type of *Class InputConfiguration*.

Class Documentation

class **Mouse**

Input configuration for mouse pointer devices.

> **Remark**
>
> Since MirAL 5.1

Public Functions

Mouse()

~Mouse()

Mouse(*Mouse* const &that)

auto **operator=**(*Mouse* that) -> *Mouse*&

auto **handedness**() const -> std::optional<*MirPointerHandedness*>

auto **acceleration**() const -> std::optional<*MirPointerAcceleration*>

auto **acceleration_bias**() const -> std::optional<double>

auto **vscroll_speed**() const -> std::optional<double>

auto **hscroll_speed**() const -> std::optional<double>

void **handedness**(std::optional<*MirPointerHandedness*> const &val)

void **acceleration**(std::optional<*MirPointerAcceleration*> const &val)

void **acceleration_bias**(std::optional<double> const &val)

> **Note**
>
> val will be clamped to the range [-1.0, 1.0]

void **vscroll_speed**(std::optional<double> const &val)

void **hscroll_speed**(std::optional<double> const &val)

Friends

friend class InputConfiguration::Self

Class InputConfiguration::Touchpad

- Defined in file_include_miral_miral_input_configuration.h

Nested Relationships

This class is a nested type of *Class InputConfiguration*.

Class Documentation

class **Touchpad**

Input configuration for touchpad devices.

> **Remark**
>
> Since MirAL 5.1

Public Functions

Touchpad()

~Touchpad()

Touchpad(*Touchpad* const &that)

auto **operator=**(*Touchpad* that) -> *Touchpad*&

auto **disable_while_typing**() const -> std::optional<bool>

auto **disable_with_external_mouse**() const -> std::optional<bool>

auto **acceleration**() const -> std::optional<*MirPointerAcceleration*>

auto **acceleration_bias**() const -> std::optional<double>

auto **vscroll_speed**() const -> std::optional<double>

auto **hscroll_speed**() const -> std::optional<double>

auto **click_mode**() const -> std::optional<*MirTouchpadClickMode*>

auto **scroll_mode**() const -> std::optional<*MirTouchpadScrollMode*>

auto **tap_to_click**() const -> std::optional<bool>

void **disable_while_typing**(std::optional<bool> const &val)

void **disable_with_external_mouse**(std::optional<bool> const &val)

void **acceleration**(std::optional<*MirPointerAcceleration*> const &val)

void **acceleration_bias**(std::optional<double> const &val)

> **Note**
>
> val will be clamped to the range [-1.0, 1.0]

void **vscroll_speed**(std::optional<double> const &val)

void **hscroll_speed**(std::optional<double> const &val)

void **click_mode**(std::optional<*MirTouchpadClickMode*> const &val)

void **scroll_mode**(std::optional<*MirTouchpadScrollMode*> const &val)

void **tap_to_click**(std::optional<bool> const &val)

Friends

friend class InputConfiguration::Self

Class InternalClientLauncher

- Defined in file_include_miral_miral_internal_client.h

Class Documentation

class **InternalClientLauncher**

Public Functions

InternalClientLauncher()

~InternalClientLauncher()

void **operator()**(mir::Server &server)

void **launch**(std::function<void(struct ::wl_display *display)> const &wayland_fd,
std::function<void(std::weak_ptr<mir::scene::Session> const session)> const &connect_notification) const

template<typename **ClientObject**>
inline void **launch**(*ClientObject* &client_object) const

Class Keymap

- Defined in file_include_miral_miral_keymap.h

Class Documentation

class **Keymap**

Load a keymap.

Public Functions

Keymap()

Apply keymap from the config.

explicit **Keymap**(std::string const &keymap)

Specify a keymap. Format is:

```
<language>[+<variant>[+<options>]]
```

Options is a comma separated list. e.g. “gb” or “us+dvorak”

~Keymap()

Keymap(*Keymap* const &that)

auto **operator=**(*Keymap* const &rhs) -> *Keymap*&

void **operator()**(mir::Server &server) const

void **set_keymap**(std::string const &keymap)

Specify a new keymap.

Class MinimalWindowManager

- Defined in file_include_miral_miral_minimal_window_manager.h

Inheritance Relationships

Base Type

- `public miral::WindowManagementPolicy` (*Class WindowManagementPolicy*)

Derived Type

- `public FloatingWindowManagerPolicy` (*Class FloatingWindowManagerPolicy*)

Class Documentation

class **MinimalWindowManager** : public miral::*WindowManagementPolicy*

Minimal implementation of a floating window management policy.

Remark

Since MirAL 2.5

Subclassed by *FloatingWindowManagerPolicy*

Public Functions

explicit **MinimalWindowManager**(*WindowManagerTools* const &tools)

MinimalWindowManager(*WindowManagerTools* const &tools, *MirInputEventModifier* pointer_drag_modifier)

Allows shells to change the modifer used to identify a window drag gesture The default is mir_input_event_modifier_alt.

> **Remark**
>
> Since MirAL 3.7

~MinimalWindowManager()

virtual auto **place_new_window**(*ApplicationInfo* const &app_info, *WindowSpecification* const &requested_specification) -> *WindowSpecification* override

Honours the requested specification.

virtual void **handle_window_ready**(*WindowInfo* &window_info) override

If the window can have focus it is given focus.

virtual void **handle_modify_window**(*WindowInfo* &window_info, *WindowSpecification* const &modifications) override

Honours the requested modifications.

virtual void **handle_raise_window**(*WindowInfo* &window_info) override

Gives focus to the requesting window (tree)

virtual auto **confirm_placement_on_display**(*WindowInfo* const &window_info, *MirWindowState* new_state, Rectangle const &new_placement) -> Rectangle override

Honours the requested placement.

virtual bool **handle_keyboard_event**(MirKeyboardEvent const *event) override

Handles Alt-Tab, Alt-Grave and Alt-F4.

virtual bool **handle_touch_event**(MirTouchEvent const *event) override

Handles touch to focus.

virtual bool **handle_pointer_event**(MirPointerEvent const *event) override

Handles pre-existing move & resize gestures, plus click to focus.

virtual void **handle_request_move**(*WindowInfo* &window_info, MirInputEvent const *input_event) override

Initiates a move gesture (only implemented for pointers)

virtual void **handle_request_resize**(*WindowInfo* &window_info, MirInputEvent const *input_event, *MirResizeEdge* edge) override

Initiates a resize gesture (only implemented for pointers)

virtual auto **confirm_inherited_move**(*WindowInfo* const &window_info, Displacement movement) -> Rectangle override

Honours the requested movement.

virtual void **advise_focus_gained**(*WindowInfo* const &window_info) override

Raises newly focused window.

virtual void **advise_focus_lost**(*WindowInfo* const &window_info) override

Notification that a window has lost focus.

Parameters

window_info – the window

virtual void **advise_new_app**(miral::*ApplicationInfo* &app_info) override

Notification that a new application has connected.

Parameters

application – the application

virtual void **advise_delete_app**(miral::*ApplicationInfo* const &app_info) override

Notification that an application has disconnected.

Parameters

application – the application

virtual void **advise_new_window**(*WindowInfo* const &app_info) override

> **Remark**
>
> Since MirAL 5.0

virtual void **advise_delete_window**(*WindowInfo* const &app_info) override

> **Remark**
>
> Since MirAL 5.0

Protected Functions

bool **begin_pointer_move**(*WindowInfo* const &window_info, MirInputEvent const *input_event)

bool **begin_pointer_resize**(*WindowInfo* const &window_info, MirInputEvent const *input_event, *MirResizeEdge* const &edge)

bool **begin_touch_move**(*WindowInfo* const &window_info, MirInputEvent const *input_event)

bool **begin_touch_resize**(*WindowInfo* const &window_info, MirInputEvent const *input_event, *MirResizeEdge* const &edge)

Protected Attributes

WindowManagerTools **tools**

Class MirRunner

- Defined in file_include_miral_miral_runner.h

Class Documentation

class **MirRunner**

Runner for applying initialization options to Mir.

Public Functions

MirRunner(int argc, char const *argv[])

MirRunner(int argc, char const *argv[], char const *config_file)

~MirRunner()

void **add_start_callback**(std::function<void()> const &start_callback)

Add a callback to be invoked when the server has started, If multiple callbacks are added they will be invoked in the sequence added.

void **add_stop_callback**(std::function<void()> const &stop_callback)

Add a callback to be invoked when the server is about to stop, If multiple callbacks are added they will be invoked in the reverse sequence added.

void **register_signal_handler**(std::initializer_list<int> signals, std::function<void(int)> const &handler)

Add signal handler to the server's main loop.

> **Remark**
>
> Since MirAL 3.7

auto **register_fd_handler**(mir::*Fd* fd, std::function<void(int)> const &handler) -> std::unique_ptr<miral::*FdHandle*>

Add a watch on a file descriptor. The handler will be triggered when there is data to read on the Fd.

> **Remark**
>
> Since MirAL 3.7

void **set_exception_handler**(std::function<void()> const &handler)

Set a handler for exceptions caught in *run_with()*. *run_with()* invokes handler() in catch (...) blocks before returning EXIT_FAILURE. Hence the exception can be re-thrown to retrieve type information. The default action is to call mir::report_exception(std::cerr)

auto **run_with**(std::initializer_list<std::function<void(::mir::Server&)>> options) -> int

Apply the supplied initialization options and run the Mir server.

Returns

EXIT_SUCCESS or EXIT_FAILURE according to whether the server ran successfully

> **Note**
>
> blocks until the Mir server exits

void **stop**()

Tell the Mir server to exit.

auto **config_file**() const -> std::string

Name of the .config file. The .config file is located via the XDG Base Directory Specification: $XDG_CONFIG_HOME or $HOME/.config followed by $XDG_CONFIG_DIRS Config file entries are long form (e.g. "x11-output=1200x720")

> **Remark**
>
> Since MirAL 2.4

auto **display_config_file**() const -> std::string

Name of the .display configuration file. The .display file is located via the XDG Base Directory Specification: $XDG_CONFIG_HOME or $HOME/.config followed by $XDG_CONFIG_DIRS Config file entries are long form (e.g. "x11-output=1200x720")

> **Remark**
>
> Since MirAL 2.4

auto **wayland_display**() const -> mir::*optional_value*<std::string>

Get the Wayland endpoint name (if any) usable as a $WAYLAND_DISPLAY value.

> **Remark**
>
> Since MirAL 2.8

auto **x11_display**() const -> mir::*optional_value*<std::string>

Get the X11 socket name (if any) usable as a $DISPLAY value.

> **Remark**
>
> Since MirAL 2.8

Class Output

- Defined in file_include_miral_miral_output.h

Nested Relationships

Nested Types

- *Struct Output::PhysicalSizeMM*

Class Documentation

class **Output**

Public Types

enum class **Type**

Values:

enumerator **unknown**

enumerator **vga**

enumerator **dvii**

enumerator **dvid**

enumerator **dvia**

enumerator **composite**

enumerator **svideo**

enumerator **lvds**

enumerator **component**

enumerator **ninepindin**

enumerator **displayport**

enumerator **hdmia**

enumerator **hdmib**

enumerator **tv**

enumerator **edp**

Public Functions

explicit **Output**(const mir::graphics::DisplayConfigurationOutput &output)

Output(*Output* const&)

Output &**operator=**(*Output* const&)

~Output()

auto **type**() const -> *Type*
The type of the output.

auto **physical_size_mm**() const -> *PhysicalSizeMM*
The physical size of the output.

auto **connected**() const -> bool
Whether the output is connected.

auto **used**() const -> bool
Whether the output is used in the configuration.

auto **pixel_format**() const -> *MirPixelFormat*
The current output pixel format.

auto **refresh_rate**() const -> double
refresh_rate in Hz

auto **power_mode**() const -> *MirPowerMode*
Current power mode.

auto **orientation**() const -> *MirOrientation*

auto **scale**() const -> float
Requested scale factor for this output, for HiDPI support.

auto **form_factor**() const -> *MirFormFactor*
Form factor of this output; phone display, tablet, monitor, TV, projector…

auto **extents**() const -> Rectangle
The logical rectangle occupied by the output, based on its position, current mode and orientation (rotation)

auto **id**() const -> int
Mir's internal output ID mostly useful for matching against a *miral::WindowInfo::output_id*.

auto **name**() const -> std::string

The output name. This matches that suppled to clients through wl_output.

> **Remark**
>
> Since MirAL 3.8

auto **attribute**(std::string const &key) const -> std::optional<std::string>

A custom attribute value.

> **Remark**
>
> Since MirAL 3.8

auto **attributes_map**() const -> std::map<std::string const, std::optional<std::string>>

A custom attribute map.

> **Remark**
>
> Since MirAL 3.8

auto **valid**() const -> bool

auto **is_same_output**(*Output* const &other) const -> bool

auto **logical_group_id**() const -> int

A positive number if this output is part of a logical output group (aka a display wall) A single display area will stretch across all outputs in a group Zero if this output is not part of a logical group.

struct **PhysicalSizeMM**

Public Members

int **width**

int **height**

Class PrependEventFilter

- Defined in file_include_miral_miral_prepend_event_filter.h

Class Documentation

class **PrependEventFilter**

Public Functions

explicit **PrependEventFilter**(std::function<bool(*MirEvent* const *event)> const &filter)

Prepend an event filter (before any existing filters, including the window manager). The supplied filter should return true if and only if it handles the event as filters later in the list will not be called.

> **Remark**
>
> Since MirAL 3.6

void **operator()**(mir::Server &server)

Class SessionLockListener

- Defined in file_include_miral_miral_session_lock_listener.h

Class Documentation

class **SessionLockListener**

Add callbacks notifying when the session has been locked and unlocked.

Public Types

using **Callback** = std::function<void()>

Public Functions

SessionLockListener(*Callback* const &on_lock, *Callback* const &on_unlock)

~SessionLockListener() = default

void **operator()**(mir::Server &server) const

Template Class SetApplicationAuthorizer

- Defined in file_include_miral_miral_application_authorizer.h

Inheritance Relationships

Base Type

- `public miral::BasicSetApplicationAuthorizer` (*Class BasicSetApplicationAuthorizer*)

Class Documentation

template<typename **Policy**>

class **SetApplicationAuthorizer** : public miral::*BasicSetApplicationAuthorizer*

Public Functions

template<typename ...**Args**>
inline explicit **SetApplicationAuthorizer**(*Args* const&... args)

inline auto **the_custom_application_authorizer**() const -> std::shared_ptr<*Policy*>

Class SetCommandLineHandler

- Defined in file_include_miral_miral_set_command_line_handler.h

Class Documentation

class **SetCommandLineHandler**

Set a handler for any command line options Mir/MirAL does not recognise. This will be invoked if any unrecognised options are found during initialisation. Any unrecognised arguments are passed to this function. The pointers remain valid for the duration of the call only. If set_command_line_handler is not called the default action is to exit by throwing *mir::AbnormalExit* (which will be handled by the exception handler prior to exiting run().

Public Types

using **Handler** = std::function<void(int argc, char const *const *argv)>

Public Functions

explicit **SetCommandLineHandler**(*Handler* const &handler)

~SetCommandLineHandler()

void **operator()**(mir::Server &server) const

Class SetTerminator

- Defined in file_include_miral_miral_set_terminator.h

Class Documentation

class **SetTerminator**

Set handler for termination requests. terminator will be called following receipt of SIGTERM or SIGINT. The default terminator stop()s the server, replacements should probably do the same in addition to any additional shutdown logic.

Public Types

using **Terminator** = std::function<void(int signal)>

Public Functions

explicit **SetTerminator**(*Terminator* const &terminator)

~SetTerminator()

void **operator()**(mir::Server &server) const

Class SetWindowManagementPolicy

- Defined in file_include_miral_miral_set_window_management_policy.h

Class Documentation

class **SetWindowManagementPolicy**

Public Functions

SetWindowManagementPolicy(std::function<std::unique_ptr<*WindowManagementPolicy*>(*WindowManagerTools* const &tools)> const &builder)

~SetWindowManagementPolicy()

void **operator()**(mir::Server &server) const

Class StartupInternalClient

- Defined in file_include_miral_miral_internal_client.h

Class Documentation

class **StartupInternalClient**

Wrapper for running an internal Mir client at startup.

Param client_code
code implementing the internal client

Param connection_notification
handler for registering the server-side application

> **Note**
>
> client_code will be executed on its own thread, this must exit

> **Note**
>
> connection_notification will be called on a worker thread and must not block

Public Functions

explicit **StartupInternalClient**(std::function<void(struct ::wl_display *display)> client_code, std::function<void(std::weak_ptr<mir::scene::Session> const session)> connect_notification)

template<typename **ClientObject**>
inline explicit **StartupInternalClient**(*ClientObject* const &client_object)

~StartupInternalClient()

void **operator()**(mir::Server &server)

Class WaylandExtensions

- Defined in file_include_miral_miral_wayland_extensions.h

Nested Relationships

Nested Types

- *Struct WaylandExtensions::Builder*
- *Class WaylandExtensions::Context*
- *Class WaylandExtensions::EnableInfo*

Class Documentation

class **WaylandExtensions**

Enable configuration of the Wayland extensions enabled at runtime.

This adds the command line options '—wayland-extensions', '—add-wayland-extensions', '—drop-wayland-extensions' and the corresponding MIR_SERVER_* environment variables and config file options.

- The server can add support for additional extensions
- The server can specify the configuration defaults
- Mir's option handling allows the defaults to be overridden

> **Remark**
>
> Since MirAL 2.4

Unnamed Group

static char const *const **zwlr_layer_shell_v1**

Supported wayland extensions that are not enabled by default.

These can be passed into *WaylandExtensions::enable()* to turn them on. Enables shell components such as panels, notifications and lock screens. It is recommended to use this in conjunction with *conditionally_enable()* as malicious clients could potentially use this protocol to steal input focus or otherwise bother the user.

> **Remark**
>
> Since MirAL 2.6

static char const *const **zxdg_output_manager_v1**

Allows clients to retrieve additional information about outputs.

> **Remark**
>
> Since MirAL 2.6

static char const *const **zwlr_foreign_toplevel_manager_v1**

Allows a client to get information and gain control over all toplevels of all clients Useful for taskbars and app switchers Could allow a client to extract information about other programs the user is running.

> **Remark**
>
> Since MirAL 3.1

static char const *const **zwp_virtual_keyboard_manager_v1**

Allows clients to act as a virtual keyboard, useful for on-screen keyboards. Clients are not required to display anything to send keyboard events using this extension, so malicious clients could use it to take actions without user input.

> **Remark**
>
> Since MirAL 3.4

static const char *const **zwp_input_method_v1**

Allows clients (such as on-screen keyboards) to intercept physical key events and act as a source of text input for other clients. Input methods are not required to display anything to use this extension, so malicious clients could use it to intercept keys events or take actions without user input.

> **Remark**
>
> Since MirAL 3.10

static const char *const **zwp_input_panel_v1**

Allows clients to display a surface as an input panel surface. The input panel surface is shown when a text input is active and hidden otherwise. The panel itself can either be attached to the edge of the screen or set to float near the active input. This is often used in conjunction with zwp_input_method_v1.

> **Remark**
>
> Since MirAL 3.10

static char const *const **zwp_input_method_manager_v2**

Allows clients (such as on-screen keyboards) to intercept physical key events and act as a source of text input for other clients. Input methods are not required to display anything to use this extension, so malicious clients could use it to intercept keys events or take actions without user input.

> **Remark**
>
> Since MirAL 3.4

static char const *const **zwlr_screencopy_manager_v1**

Allows clients to take screenshots and record the screen. Only enable for clients that are trusted to view all displayed content, including windows of other apps.

> **Remark**
>
> Since MirAL 3.5

static char const *const **zwlr_virtual_pointer_manager_v1**

Allows clients to act as a virtual pointer, useful for remote control and automation. Clients are not required to display anything to send pointer events using this extension, so malicious clients could use it to take actions without user input.

> **Remark**
>
> Since MirAL 3.6

static char const *const **ext_session_lock_manager_v1**

Allows clients to act as a screen lock.

> **Remark**
>
> Since MirAL 3.6

void **add_extension**(*Builder* const &builder)

Add a bespoke Wayland extension both to “supported” and “enabled by default”.

Remark

Since MirAL 2.5

void **add_extension_disabled_by_default**(*Builder* const &builder)

Add a bespoke Wayland extension both to "supported" but not "enabled by default".

Remark

Since MirAL 2.5

auto **enable**(std::string name) -> *WaylandExtensions*&

Enable a Wayland extension by default. The user can still disable it with the drop-wayland-extensions or wayland-extensions options. The extension can be forced to be enabled regardless of user options with *conditionally_enable()*.

Remark

Since MirAL 2.6

auto **disable**(std::string name) -> *WaylandExtensions*&

Disable a Wayland extension by default. The user can still enable it with the add-wayland-extensions or wayland-extensions options. The extension can be forced to be disabled regardless of user options with *conditionally_enable()*.

Remark

Since MirAL 2.6

auto **conditionally_enable**(std::string name, *EnableCallback* const &callback) -> *WaylandExtensions*&

Enable a Wayland extension only when the callback returns true. The callback can use info.user_preference() to respect the extension options the user provided, it is not required. Unlike *enable()* and *disable()*, *conditionally_enable()* can override the user options. The callback may be called multiple times for a each client/extension pair (for example, once each time a client creates or destroys a wl_registry). All client processing will be blocked while the callback is being executed. To minimise the impact on client responsiveness users may want to cache the result of any expensive checks made in the callback.

Remark

Since MirAL 3.4

static auto **recommended**() -> std::set<std::string>

The set of Wayland extensions that Mir recommends. Also the set that is enabled by default upon construction of a *WaylandExtensions* object.

> **Remark**
>
> Since MirAL 2.6

static auto **supported**() -> std::set<std::string>

The set of Wayland extensions that core Mir supports. Does not include bespoke extensions A superset of *recommended()*

> **Remark**
>
> Since MirAL 2.6

Public Types

using **Filter** = std::function<bool(*Application* const &app, char const *protocol)>

> **Remark**
>
> Since MirAL 2.5

using **EnableCallback** = std::function<bool(*EnableInfo* const &info)>

> **Remark**
>
> Since MirAL 3.4

Public Functions

WaylandExtensions()

Default to enabling the extensions recommended by Mir.

void **operator()**(mir::Server &server) const

~WaylandExtensions()

WaylandExtensions(*WaylandExtensions* const&)

auto **operator=**(*WaylandExtensions* const&) -> *WaylandExtensions*&

auto **all_supported**() const -> std::set<std::string>

All Wayland extensions supported. This includes both the *supported()* provided by Mir and any extensions that have been added using *add_extension()*.

Remark

Since MirAL 3.0

struct **Builder**

A *Builder* creates and registers an extension protocol.

Remark

Since MirAL 2.5

Public Members

std::string **name**

Name of the protocol extension's Wayland global.

std::function< std::shared_ptr< void >Context const *context)> build

Functor that creates and registers an extension protocol.

Param context

giving access to:

- the wl_display (so that, for example, the extension can be registered); and,
- allowing server initiated code to be executed on the Wayland mainloop.

Return

a shared pointer to the implementation. (Mir will manage the lifetime)

class **Context**

Context information useful for implementing Wayland extensions.

Remark

Since MirAL 2.5

Public Functions

virtual auto **display**() const -> wl_display* = 0

virtual void **run_on_wayland_mainloop**(std::function<void()> &&work) const = 0

Protected Functions

Context() = default

virtual **~Context**() = default

Context(*Context* const&) = delete

Context &**operator=**(*Context* const&) = delete

class **EnableInfo**

Information that can be used to determine if to enable a conditionally enabled extension.

> **Remark**
>
> Since MirAL 3.4

Public Functions

auto **app**() const -> *Application* const&

The application that is being given access to this extension.

auto **name**() const -> const char*

The name of the extension/global, always the same as given to *conditionally_enable()*

auto **user_preference**() const -> std::optional<bool>

If the user has enabled or disabled this extension one of the wayland extension Mir options.

Class WaylandExtensions::Context

- Defined in file_include_miral_miral_wayland_extensions.h

Nested Relationships

This class is a nested type of *Class WaylandExtensions*.

Class Documentation

class **Context**

Context information useful for implementing Wayland extensions.

> **Remark**
>
> Since MirAL 2.5

Public Functions

virtual auto **display**() const -> wl_display* = 0

virtual void **run_on_wayland_mainloop**(std::function<void()> &&work) const = 0

Protected Functions

Context() = default

virtual **~Context**() = default

Context(*Context* const&) = delete

Context &**operator=**(*Context* const&) = delete

Class WaylandExtensions::EnableInfo

- Defined in file_include_miral_miral_wayland_extensions.h

Nested Relationships

This class is a nested type of *Class WaylandExtensions*.

Class Documentation

class **EnableInfo**

Information that can be used to determine if to enable a conditionally enabled extension.

> **Remark**
>
> Since MirAL 3.4

Public Functions

auto **app**() const -> *Application* const&

The application that is being given access to this extension.

auto **name**() const -> const char*

The name of the extension/global, always the same as given to *conditionally_enable()*

auto **user_preference**() const -> std::optional<bool>

If the user has enabled or disabled this extension one of the wayland extension Mir options.

Class Window

- Defined in file_include_miral_miral_window.h

Class Documentation

class **Window**

Handle class to manage a Mir surface. It may be null (e.g. default initialized)

Unnamed Group

void **resize**(mir::geometry::*Size* const &size)

Not for external use, use *WindowManagerTools::modify_window()* instead.

void **move_to**(mir::geometry::*Point* top_left)

Public Functions

Window()

Window(*Application* const &application, std::shared_ptr<mir::scene::Surface> const &surface)

~Window()

auto **top_left**() const -> mir::geometry::*Point*

The position of the top-left corner of the window frame.

auto **size**() const -> mir::geometry::*Size*

The size of the window frame. Units are logical screen coordinates (not necessarily device pixels). Any decorations are included in the size.

auto **application**() const -> *Application*

The application that created this window.

operator bool() const

Indicates that the *Window* isn't null.

Class WindowManagementPolicy

- Defined in file_include_miral_miral_window_management_policy.h

Inheritance Relationships

Derived Types

- `public TilingWindowManagerPolicy` (*Class TilingWindowManagerPolicy*)
- `public miral::CanonicalWindowManagerPolicy` (*Class CanonicalWindowManagerPolicy*)
- `public miral::MinimalWindowManager` (*Class MinimalWindowManager*)

Class Documentation

class **WindowManagementPolicy**

The interface through which the window management policy is determined.

Subclassed by *TilingWindowManagerPolicy*, *miral::CanonicalWindowManagerPolicy*, *miral::MinimalWindowManager*

handle events originating from the client

The policy is expected to update the model as appropriate

virtual void **handle_window_ready**(*WindowInfo* &window_info) = 0

notification that the first buffer has been posted

Parameters

window_info – the window

virtual void **handle_modify_window**(*WindowInfo* &window_info, *WindowSpecification* const &modifications) = 0

request from client to modify the window specification.

Parameters

- **window_info** – the window
- **modifications** – the requested changes

> **Note**
>
> the request has already been validated against the type definition

virtual void **handle_raise_window**(*WindowInfo* &window_info) = 0

request from client to raise the window

Parameters

window_info – the window

> **Note**
>
> the request has already been validated against the requesting event

virtual auto **confirm_placement_on_display**(*WindowInfo* const &window_info, *MirWindowState* new_state, Rectangle const &new_placement) -> Rectangle = 0

Confirm (and optionally adjust) the placement of a window on the display.

Called when (re)placing fullscreen, maximized, horizontally maximised and vertically maximized windows to allow adjustment for decorations.

Parameters

- **window_info** – the window
- **new_state** – the new state
- **new_placement** – the suggested placement

Returns

the confirmed placement of the window

handle events originating from user

The policy is expected to interpret (and optionally consume) the event

virtual bool **handle_keyboard_event**(MirKeyboardEvent const *event) = 0

keyboard event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_touch_event**(MirTouchEvent const *event) = 0

touch event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_pointer_event**(MirPointerEvent const *event) = 0

pointer event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

notification of WM events that the policy may need to track.

virtual void **advise_new_app**(*ApplicationInfo* &application)

Notification that a new application has connected.

Parameters

application – the application

virtual void **advise_delete_app**(*ApplicationInfo* const &application)

Notification that an application has disconnected.

Parameters

application – the application

virtual void **advise_new_window**(*WindowInfo* const &window_info)

Notification that a window has been created.

Parameters

window_info – the window

virtual void **advise_focus_lost**(*WindowInfo* const &window_info)

Notification that a window has lost focus.

Parameters

window_info – the window

virtual void **advise_focus_gained**(*WindowInfo* const &window_info)

Notification that a window has gained focus.

Parameters

window_info – the window

virtual void **advise_state_change**(*WindowInfo* const &window_info, *MirWindowState* state)

Notification that a window is about to change state.

Parameters

- **window_info** – the window
- **state** – the new state

virtual void **advise_move_to**(*WindowInfo* const &window_info, Point top_left)

Notification that a window is about to move.

Parameters

- **window_info** – the window
- **top_left** – the new position

virtual void **advise_resize**(*WindowInfo* const &window_info, Size const &new_size)

Notification that a window is about to resize.

Parameters

- **window_info** – the window
- **new_size** – the new size

virtual void **advise_delete_window**(*WindowInfo* const &window_info)

Notification that a window is about to be destroyed.

Parameters

window_info – the window

virtual void **advise_raise**(std::vector<*Window*> const &windows)

Notification that windows are being raised to the top.

These windows are ordered with parents before children, and form a single tree rooted at the first element.

Parameters

windows – the windows

> **Note**
>
> The relative Z-order of these windows will be maintained, they will be raised en bloc.

virtual void **advise_adding_to_workspace**(std::shared_ptr<Workspace> const &workspace,
std::vector<*Window*> const &windows)

Notification that windows are being added to a workspace.

These windows are ordered with parents before children, and form a single tree rooted at the first element.

Parameters

- **workspace** – the workspace
- **windows** – the windows

virtual void **advise_removing_from_workspace**(std::shared_ptr<Workspace> const &workspace, std::vector<*Window*> const &windows)

Notification that windows are being removed from a workspace.

These windows are ordered with parents before children, and form a single tree rooted at the first element.

Parameters

- **workspace** – the workspace
- **windows** – the windows

handle requests originating from the client

The policy is expected to update the model as appropriate

virtual void **handle_request_move**(*WindowInfo* &window_info, MirInputEvent const *input_event) = 0

request from client to initiate move

Parameters

- **window_info** – the window
- **input_event** – the requesting event

> **Note**
>
> the request has already been validated against the requesting event

virtual void **handle_request_resize**(*WindowInfo* &window_info, MirInputEvent const *input_event, *MirResizeEdge* edge) = 0

request from client to initiate resize

Parameters

- **window_info** – the window
- **input_event** – the requesting event
- **edge** – the edge(s) being dragged

> **Note**
>
> the request has already been validated against the requesting event

notification of changes to the (connected, active) outputs.

virtual void **advise_output_create**(*Output* const &output)

virtual void **advise_output_update**(*Output* const &updated, *Output* const &original)

virtual void **advise_output_delete**(*Output* const &output)

notification of changes to the current application zones

An application zone is the area a maximized application will fill.

There is often (but not necessarily) one zone per output. The areas normal applications windows should avoid (such as the areas covered by panels) will not be part of an application zone

virtual void **advise_application_zone_create**(*Zone* const &application_zone)

virtual void **advise_application_zone_update**(*Zone* const &updated, *Zone* const &original)

virtual void **advise_application_zone_delete**(*Zone* const &application_zone)

Public Functions

virtual void **advise_begin**()

before any related calls begin

virtual void **advise_end**()

after any related calls end

virtual auto **place_new_window**(*ApplicationInfo* const &app_info, *WindowSpecification* const &requested_specification) -> *WindowSpecification* = 0

Customize initial window placement.

Parameters

- **app_info** – the application requesting a new window
- **requested_specification** – the requested specification (updated with default placement)

Returns

the customized specification

virtual auto **confirm_inherited_move**(*WindowInfo* const &window_info, Displacement movement) -> Rectangle = 0

Confirm (and optionally adjust) the motion of a child window when the parent is moved.

Parameters

- **window_info** – the window
- **movement** – the movement of the parent

Returns

the confirmed placement of the window

virtual **~WindowManagementPolicy**() = default

WindowManagementPolicy() = default

WindowManagementPolicy(*WindowManagementPolicy* const&) = delete

WindowManagementPolicy &**operator=**(*WindowManagementPolicy* const&) = delete

Class WindowManagerOptions

- Defined in file_include_miral_miral_window_management_options.h

Class Documentation

class **WindowManagerOptions**

Public Functions

WindowManagerOptions() = delete

inline explicit **WindowManagerOptions**(std::initializer_list<*WindowManagerOption*> const &policies)

void **operator()**(mir::Server &server) const

Public Members

std::vector<*WindowManagerOption*> const **policies**

Class WindowManagerTools

- Defined in file_include_miral_miral_window_manager_tools.h

Class Documentation

class **WindowManagerTools**

Window management functions for querying and updating MirAL's model.

Query & Update Model

These functions assume that the BasicWindowManager data structures can be accessed freely.

I.e. they should only be used by a thread that has called the *WindowManagementPolicy* methods (where any necessary locks are held) or via a *invoke_under_lock()* callback.

auto **count_applications**() const -> unsigned int

count the applications

Returns

number of applications

void **for_each_application**(std::function<void(*ApplicationInfo* &info)> const &functor)

execute functor for each application

Parameters

functor – the functor

auto **find_application**(std::function<bool(*ApplicationInfo* const &info)> const &predicate) -> *Application*

find an application meeting the predicate

Parameters

predicate – the predicate

Returns

the application

auto **info_for**(std::weak_ptr<mir::scene::Session> const &session) const -> *ApplicationInfo*&

retrieve metadata for an application

Parameters

session – the application session

Returns

the metadata

auto **info_for**(std::weak_ptr<mir::scene::Surface> const &surface) const -> *WindowInfo*&

retrieve metadata for a window

Parameters

surface – the window surface

Returns

the metadata

auto **info_for**(*Window* const &window) const -> *WindowInfo*&

retrieve metadata for a window

Parameters

window – the window

Returns

the metadata

auto **info_for_window_id**(std::string const &id) const -> *WindowInfo*&

retrieve metadata for a persistent surface id

Parameters

id – the persistent surface id

Throws

`invalid_argument` – or runtime_error if the id is badly formatted/doesn't identify a current window

Returns

the metadata

auto **id_for_window**(*Window* const &window) const -> std::string

retrieve the persistent surface id for a window

Parameters

window – the window

Returns

the persistent surface id

void **ask_client_to_close**(*Window* const &window)

Send close request to the window.

auto **active_window**() const -> *Window*

retrieve the active window

auto **select_active_window**(*Window* const &hint) -> *Window*

select a new active window based on the hint

Parameters

hint – the hint

Returns

the new active window

void **drag_active_window**(mir::geometry::*Displacement* movement)

move the active window

void **drag_window**(*Window* const &window, mir::geometry::*Displacement* movement)

move the window

void **focus_next_application**()

make the next application active

void **focus_prev_application**()

make the previous application active

> **Remark**
>
> Since MirAL 2.5

void **focus_next_within_application**()

make the next surface active within the active application

void **focus_prev_within_application**()

make the prev surface active within the active application

auto **window_to_select_application**(const *Application*) const -> std::optional<*Window*>

If possible, returns the application window to select, otherwise `std::nullopt`

> **Remark**
>
> Since MirAL 3.10

auto **can_select_window**(*Window* const&) const -> bool

Check if the provided window can be selected.

> **Remark**
>
> Since MirAL 5.0

auto **window_at**(mir::geometry::*Point* cursor) const -> *Window*

Find the topmost window at the cursor.

auto **active_output**() -> mir::geometry::*Rectangle* const

Find the active output area.

auto **active_application_zone**() const -> *Zone*

Find the active zone area.

> **Remark**
>
> Since MirAL 3.0

void **raise_tree**(*Window* const &root)

Raise window and all its children.

void **swap_tree_order**(*Window* const &first, *Window* const &second)

Swaps the position of the windows in regards to Z order.

> **Remark**
>
> SinceMirAL 3.10

Parameters

- **first** –
- **second** –

void **send_tree_to_back**(*Window* const &root)

Moves the window to the bottom of the Z order remark Since MirAL 3.10.

void **modify_window**(*WindowInfo* &window_info, *WindowSpecification* const &modifications)

Apply modifications to a window.

void **modify_window**(*Window* const &window, *WindowSpecification* const &modifications)

Apply modifications to a window.

void **place_and_size_for_state**(*WindowSpecification* &modifications, *WindowInfo* const &window_info) const

Set a default size and position to reflect state change.

auto **create_workspace**() -> std::shared_ptr<Workspace>

Create a workspace.

> **Remark**
>
> the tools hold only a weak_ptr<> to the workspace - there is no need for an explicit “destroy”.

Returns

a shared_ptr owning the workspace

void **add_tree_to_workspace**(*Window* const &window, std::shared_ptr<Workspace> const &workspace)

Add the tree containing window to a workspace.

Parameters

- **window** – the window
- **workspace** – the workspace;

void **remove_tree_from_workspace**(*Window* const &window, std::shared_ptr<Workspace> const &workspace)

Remove the tree containing window from a workspace.

Parameters

- **window** – the window
- **workspace** – the workspace;

void **move_workspace_content_to_workspace**(std::shared_ptr<Workspace> const &to_workspace, std::shared_ptr<Workspace> const &from_workspace)

Moves all the content from one workspace to another.

Parameters

- **from_workspace** – the workspace to move the windows from;
- **to_workspace** – the workspace to move the windows to;

void **for_each_workspace_containing**(*Window* const &window, std::function<void(std::shared_ptr<Workspace> const &workspace)> const &callback)

invoke callback with each workspace containing window

Parameters

- **window** –
- **callback** –

Warning

it is unsafe to add or remove windows from workspaces from the callback during enumeration

void **for_each_window_in_workspace**(std::shared_ptr<Workspace> const &workspace, std::function<void(*Window* const &window)> const &callback)

invoke callback with each window contained in workspace

Parameters

- **workspace** –
- **callback** –

Warning

it is unsafe to add or remove windows from workspaces from the callback during enumeration

Public Functions

explicit **WindowManagerTools**(WindowManagerToolsImplementation *tools)

WindowManagerTools(*WindowManagerTools* const&)

WindowManagerTools &**operator=**(*WindowManagerTools* const&)

~WindowManagerTools()

void **invoke_under_lock**(std::function<void()> const &callback)

Multi-thread support Allows threads that don't hold a lock on the model to acquire one and call the "Update Model" member functions.

This should NOT be used by a thread that has called the *WindowManagementPolicy* methods (and already holds the lock).

void **move_cursor_to**(mir::geometry::*PointF* point)

Move the cursor to the provided point.

If the point is outside of the range of the outputs, the point is clamped.

Parameters

point –

Class WindowSpecification

- Defined in file_include_miral_miral_window_specification.h

Nested Relationships

Nested Types

- *Struct WindowSpecification::AspectRatio*

Class Documentation

class **WindowSpecification**

Unnamed Group

auto **min_width**() -> mir::*optional_value*<Width>&

Constrains how a window can be resized, see the corresponding properties on *WindowInfo* for details.

auto **min_height**() -> mir::*optional_value*<Height>&

auto **max_width**() -> mir::*optional_value*<Width>&

auto **max_height**() -> mir::*optional_value*<Height>&

auto **width_inc**() -> mir::*optional_value*<DeltaX>&

auto **height_inc**() -> mir::*optional_value*<DeltaY>&

auto **min_aspect**() -> mir::*optional_value*<*AspectRatio*>&

auto **max_aspect**() -> mir::*optional_value*<*AspectRatio*>&

Unnamed Group

auto **depth_layer**() const -> mir::*optional_value*<*MirDepthLayer*> const&

The depth layer of a child window is updated with the depth layer of its parent, but can be overridden.

auto **depth_layer**() -> mir::*optional_value*<*MirDepthLayer*>&

Unnamed Group

auto **attached_edges**() const -> mir::*optional_value*<*MirPlacementGravity*> const&

The set of window eges that are attched to edges of the output If attached to perpendicular edges, it is attached to the corner where the two edges intersect If attached to oposite edges (eg left and right), it is stretched across the output in that direction If all edges are specified, it takes up the entire output.

auto **attached_edges**() -> mir::*optional_value*<*MirPlacementGravity*>&

Unnamed Group

auto **exclusive_rect**() const -> mir::*optional_value*<mir::*optional_value*<mir::geometry::*Rectangle*>> const&

The area over which the window should not be occluded Only meaningful for windows attached to an edge If the outer optional is unset (the default), the window's exclusive rect is not changed by this spec If the outer optional is set but the inner is not, the window's exclusive rect is cleared.

auto **exclusive_rect**() -> mir::*optional_value*<mir::*optional_value*<mir::geometry::*Rectangle*>>&

Unnamed Group

auto **ignore_exclusion_zones**() const -> mir::*optional_value*<bool> const&

Decides whether or not this window should ignore the exclusion zones set by other windows. Only meaningful for windows attached to an edge.

auto **ignore_exclusion_zones**() -> mir::*optional_value*<bool>&

Unnamed Group

auto **application_id**() const -> mir::*optional_value*<std::string> const&

The D-bus service name and basename of the app's .desktop file See http://standards.freedesktop.org/desktop-entry-spec/.

auto **application_id**() -> mir::*optional_value*<std::string>&

Unnamed Group

auto **server_side_decorated**() const -> mir::*optional_value*<bool> const&

If this window should have server-side decorations provided by Mir Currently, Mir only respects this value during surface construction.

auto **server_side_decorated**() -> mir::*optional_value*<bool>&

Unnamed Group

auto **focus_mode**() const -> mir::*optional_value*<*MirFocusMode*> const&

How the window should gain and lose focus.

> **Remark**
>
> Since MirAL 3.3

auto **focus_mode**() -> mir::*optional_value*<*MirFocusMode*>&

Unnamed Group

auto **visible_on_lock_screen**() const -> mir::*optional_value*<bool> const&

If this surface should be shown while the compositor is locked.

> **Remark**
>
> Since MirAL 3.9

auto **visible_on_lock_screen**() -> mir::*optional_value*<bool>&

Public Types

enum class **InputReceptionMode**

Values:

enumerator **normal**

enumerator **receives_all_input**

Public Functions

WindowSpecification()

WindowSpecification(*WindowSpecification* const &that)

auto **operator=**(*WindowSpecification* const &that) -> *WindowSpecification*&

WindowSpecification(mir::shell::SurfaceSpecification const &spec)

~WindowSpecification()

auto **top_left**() const -> mir::*optional_value*<Point> const&

auto **size**() const -> mir::*optional_value*<Size> const&

auto **name**() const -> mir::*optional_value*<std::string> const&

auto **output_id**() const -> mir::*optional_value*<int> const&

auto **type**() const -> mir::*optional_value*<*MirWindowType*> const&

auto **state**() const -> mir::*optional_value*<*MirWindowState*> const&

auto **preferred_orientation**() const -> mir::*optional_value*<*MirOrientationMode*> const&

auto **aux_rect**() const -> mir::*optional_value*<Rectangle> const&

auto **placement_hints**() const -> mir::*optional_value*<*MirPlacementHints*> const&

auto **window_placement_gravity**() const -> mir::*optional_value*<*MirPlacementGravity*> const&

auto **aux_rect_placement_gravity**() const -> mir::*optional_value*<*MirPlacementGravity*> const&

auto **aux_rect_placement_offset**() const -> mir::*optional_value*<Displacement> const&

auto **min_width**() const -> mir::*optional_value*<Width> const&

auto **min_height**() const -> mir::*optional_value*<Height> const&

auto **max_width**() const -> mir::*optional_value*<Width> const&

auto **max_height**() const -> mir::*optional_value*<Height> const&

auto **width_inc**() const -> mir::*optional_value*<DeltaX> const&

auto **height_inc**() const -> mir::*optional_value*<DeltaY> const&

auto **min_aspect**() const -> mir::*optional_value*<*AspectRatio*> const&

auto **max_aspect**() const -> mir::*optional_value*<*AspectRatio*> const&

auto **parent**() const -> mir::*optional_value*<std::weak_ptr<mir::scene::Surface>> const&

auto **input_shape**() const -> mir::*optional_value*<std::vector<Rectangle>> const&

auto **input_mode**() const -> mir::*optional_value*<*InputReceptionMode*> const&

auto **shell_chrome**() const -> mir::*optional_value*<*MirShellChrome*> const&

auto **confine_pointer**() const -> mir::*optional_value*<*MirPointerConfinementState*> const&

auto **userdata**() const -> mir::*optional_value*<std::shared_ptr<void>> const&

auto **top_left**() -> mir::*optional_value*<Point>&

auto **size**() -> mir::*optional_value*<Size>&

The new size of the window frame (including any decorations). Will be adjusted based on *min_width()*, *WindowInfo::max_width()*, *WindowInfo::min_height()*, *WindowInfo::max_height()*, *WindowInfo::min_aspect()*, *WindowInfo::max_aspect()*, *WindowInfo::width_inc()* and *WindowInfo::height_inc()*. Set these to properties to their default values if they should be ignored.

auto **name**() -> mir::*optional_value*<std::string>&

auto **output_id**() -> mir::*optional_value*<int>&

auto **type**() -> mir::*optional_value*<*MirWindowType*>&

auto **state**() -> mir::*optional_value*<*MirWindowState*>&

auto **preferred_orientation**() -> mir::*optional_value*<*MirOrientationMode*>&

auto **aux_rect**() -> mir::*optional_value*<Rectangle>&

Relative to window's surface's CONTENT offset and size (not equal to the top_left and size exposed by this interface if server-side decorations are in use)

auto **placement_hints**() -> mir::*optional_value*<*MirPlacementHints*>&

auto **window_placement_gravity**() -> mir::*optional_value*<*MirPlacementGravity*>&

auto **aux_rect_placement_gravity**() -> mir::*optional_value*<*MirPlacementGravity*>&

auto **aux_rect_placement_offset**() -> mir::*optional_value*<Displacement>&

auto **parent**() -> mir::*optional_value*<std::weak_ptr<mir::scene::Surface>>&

auto **input_shape**() -> mir::*optional_value*<std::vector<Rectangle>>&

auto **input_mode**() -> mir::*optional_value*<*InputReceptionMode*>&

auto **shell_chrome**() -> mir::*optional_value*<*MirShellChrome*>&

auto **confine_pointer**() -> mir::*optional_value*<*MirPointerConfinementState*>&

auto **userdata**() -> mir::*optional_value*<std::shared_ptr<void>>&

struct **AspectRatio**

Public Members

unsigned **width**

unsigned **height**

Class X11Support

- Defined in file_include_miral_miral_x11_support.h

Class Documentation

class **X11Support**

Add user configuration options for X11 support.

> **Remark**
>
> Since MirAL 2.4

> **Note**
>
> From MirAL 4.1, the `enable-x11` option requires an argument (e.g., `enable-x11=true` to enable X11 support).

Public Functions

void **operator()**(mir::Server &server) const

X11Support()

~X11Support()

X11Support(*X11Support* const&)

auto **operator=**(*X11Support* const&) -> *X11Support*&

auto **default_to_enabled**() -> *X11Support*&

Set X11 support as enabled by default.

> **Remark**
>
> Since MirAL 4.1

Returns

A reference to the modified *miral::X11Support* object with the updated default configuration.

Class Zone

- Defined in file_include_miral_miral_zone.h

Class Documentation

class **Zone**

A rectangular area of the display. Not tied to a specific output.

Public Functions

Zone(Rectangle const &extents)

Create a new zone with the given extents.

Zone(*Zone* const &other)

Makes a copy of the underlying private data.

Zone &**operator=**(*Zone* const &other)

Copies private data by value.

~Zone()

auto **operator==**(*Zone* const &other) const -> bool

Returns true only if all properties including IDs match.

auto **is_same_zone**(*Zone* const &other) const -> bool

Returns if true if zone IDs match, even if extents are different.

auto **extents**() const -> Rectangle

The area of this zone in global display coordinates.

void **extents**(Rectangle const &extents)

Set the extents of this zone Does not make this a different zone.

auto **id**() const -> int

An arbitrary number that uniquely identifies this zone, reguardless of how it is resized and moved.

> **Remark**
>
> Since MirAL 3.6

Class MirEglSurface

- Defined in file_examples_miral-shell_spinner_miregl.h

Inheritance Relationships

Base Type

- `private WaylandSurface` (*Class WaylandSurface*)

Class Documentation

class **MirEglSurface** : private *WaylandSurface*

Public Functions

MirEglSurface(std::shared_ptr<MirEglApp> const &mir_egl_app, struct *wl_output* *wl_output)

~MirEglSurface()

template<typename **Painter**>
inline void **paint**(*Painter* const &functor)

Class Compositor

- Defined in file_include_miroil_miroil_compositor.h

Class Documentation

class **Compositor**

Public Functions

virtual **~Compositor**()

Compositor &**operator=**(*Compositor* const&) = delete

virtual void **start**() = 0

virtual void **stop**() = 0

Protected Functions

Compositor() = default

Compositor(*Compositor* const&) = delete

Class DisplayConfigurationControllerWrapper

- Defined in file_include_miroil_miroil_display_configuration_controller_wrapper.h

Class Documentation

class **DisplayConfigurationControllerWrapper**

Public Functions

DisplayConfigurationControllerWrapper(std::shared_ptr<mir::shell::DisplayConfigurationController> const &wrapped)

~DisplayConfigurationControllerWrapper() = default

void **set_base_configuration**(std::shared_ptr<mir::graphics::DisplayConfiguration> const &conf)

Set the base display configuration.

This is the display configuration that is used by default, but will be overridden by a client's requested configuration if that client is focused.

Parameters

conf – **[in]** The new display configuration to set

Class DisplayConfigurationPolicy

- Defined in file_include_miroil_miroil_display_configuration_policy.h

Class Documentation

class **DisplayConfigurationPolicy**

Public Functions

DisplayConfigurationPolicy()

virtual **~DisplayConfigurationPolicy**()

DisplayConfigurationPolicy(*DisplayConfigurationPolicy* const&) = delete

DisplayConfigurationPolicy &**operator=**(*DisplayConfigurationPolicy* const&) = delete

virtual void **apply_to**(mir::graphics::DisplayConfiguration &conf) = 0

Class DisplayConfigurationStorage

- Defined in file_include_miroil_miroil_display_configuration_storage.h

Class Documentation

class **DisplayConfigurationStorage**

Public Functions

virtual **~DisplayConfigurationStorage**() = default

virtual void **save**(const *DisplayId*&, const *DisplayConfigurationOptions*&) = 0

virtual bool **load**(const *DisplayId*&, *DisplayConfigurationOptions*&) const = 0

Class DisplayListenerWrapper

- Defined in file_include_miroil_miroil_display_listener_wrapper.h

Class Documentation

class **DisplayListenerWrapper**

Public Functions

DisplayListenerWrapper(std::shared_ptr<mir::compositor::DisplayListener> const &display_listener)

~DisplayListenerWrapper()

void **add_display**(mir::geometry::*Rectangle* const &area)

void **remove_display**(mir::geometry::*Rectangle* const &area)

Class EventBuilder

- Defined in file_include_miroil_miroil_event_builder.h

Nested Relationships

Nested Types

- *Class EventBuilder::EventInfo*

Class Documentation

class **EventBuilder**

Public Functions

EventBuilder()

virtual **~EventBuilder**()

void **add_touch**(*MirEvent* &event, *MirTouchId* touch_id, *MirTouchAction* action, *MirTouchTooltype* tooltype, float x_axis_value, float y_axis_value, float pressure_value, float touch_major_value, float touch_minor_value, float size_value)

mir::*EventUPtr* **make_key_event**(*MirInputDeviceId* device_id, std::chrono::nanoseconds timestamp, *MirKeyboardAction* action, xkb_keysym_t keysym, int scan_code, *MirInputEventModifiers* modifiers)

mir::*EventUPtr* **make_touch_event**(*MirInputDeviceId* device_id, std::chrono::nanoseconds timestamp, *MirInputEventModifiers* modifiers)

mir::*EventUPtr* **make_pointer_event**(*MirInputDeviceId* device_id, std::chrono::nanoseconds timestamp, *MirInputEventModifiers* modifiers, *MirPointerAction* action, *MirPointerButtons* buttons_pressed, float x_axis_value, float y_axis_value, float hscroll_value, float vscroll_value, float relative_x_value, float relative_y_value)

EventInfo ***find_info**(ulong qtTimestamp)

void **store**(const MirInputEvent *mirInputEvent, ulong qtTimestamp)

class **EventInfo**

Public Functions

void **store**(const MirInputEvent *mirInputEvent, ulong qtTimestamp)

Public Members

ulong **timestamp**

MirInputDeviceId **device_id**

float **relative_x** = {0}

float **relative_y** = {0}

Class EventBuilder::EventInfo

- Defined in file_include_miroil_miroil_event_builder.h

Nested Relationships

This class is a nested type of *Class EventBuilder*.

Class Documentation

class **EventInfo**

Public Functions

void **store**(const MirInputEvent *mirInputEvent, ulong qtTimestamp)

Public Members

ulong **timestamp**

MirInputDeviceId **device_id**

float **relative_x** = {0}

float **relative_y** = {0}

Class GLBuffer

- Defined in file_include_miroil_miroil_mirbuffer.h

Class Documentation

class **GLBuffer**

Public Functions

~GLBuffer()

explicit **GLBuffer**(std::shared_ptr<mir::graphics::Buffer> const &buffer)

bool **has_alpha_channel**() const

mir::geometry::*Size* **size**() const

void **bind**()

void **reset**(std::shared_ptr<mir::graphics::Buffer> const &buffer)

void **reset**()

bool **empty**()

Public Static Functions

static std::shared_ptr<*GLBuffer*> **from_mir_buffer**(std::shared_ptr<mir::graphics::Buffer> const &buffer)

Class InputDevice

- Defined in file_include_miroil_miroil_input_device.h

Class Documentation

class **InputDevice**

Public Functions

InputDevice(std::shared_ptr<mir::input::Device> const &device)

InputDevice(*InputDevice* const &src)

InputDevice(*InputDevice* &&src)

InputDevice()

~InputDevice()

auto operator=(*InputDevice* const &src) -> *InputDevice*&

auto operator=(*InputDevice* &&src) -> *InputDevice*&

bool operator==(*InputDevice* const &other)

void apply_keymap(std::string const &layout, std::string const &variant)

auto get_device_id() -> *MirInputDeviceId*

auto get_device_name() -> std::string

auto is_keyboard() -> bool

auto is_alpha_numeric() -> bool

Class InputDeviceObserver

- Defined in file_include_miroil_miroil_input_device_observer.h

Class Documentation

class InputDeviceObserver

Public Functions

InputDeviceObserver() = default

InputDeviceObserver(*InputDeviceObserver* const&) = delete

InputDeviceObserver &operator=(*InputDeviceObserver* const&) = delete

virtual ~InputDeviceObserver()

virtual void device_added(miroil::*InputDevice* device) = 0

virtual void device_removed(miroil::*InputDevice* device) = 0

Class MirPromptSession

- Defined in file_include_miroil_miroil_mir_prompt_session.h

Class Documentation

class **MirPromptSession**

Public Functions

MirPromptSession(::*MirPromptSession* *promptSession)

MirPromptSession(*MirPromptSession* const &src)

MirPromptSession(*MirPromptSession* &&src)

~MirPromptSession()

auto **operator=**(*MirPromptSession* const &src) -> *MirPromptSession*&

auto **operator=**(*MirPromptSession* &&src) -> *MirPromptSession*&

bool **operator==**(*MirPromptSession* const &other)

bool **new_fds_for_prompt_providers**(unsigned int no_of_fds, *MirClientFdCallback* callback, void *context)

Public Members

::*MirPromptSession* ***prompt_session**

Class MirServerHooks

- Defined in file_include_miroil_miroil_mir_server_hooks.h

Class Documentation

class **MirServerHooks**

Public Functions

MirServerHooks()

void **operator()**(mir::Server &server)

auto **the_prompt_session_listener**() const -> *PromptSessionListener**

auto **the_prompt_session_manager**() const -> std::shared_ptr<mir::scene::PromptSessionManager>

auto **the_mir_display**() const -> std::shared_ptr<mir::graphics::Display>

auto **the_display_configuration_controller**() const -> std::shared_ptr<mir::shell::DisplayConfigurationController>

void **create_named_cursor**(*CreateNamedCursor* func)

void **create_input_device_observer**(std::shared_ptr<*InputDeviceObserver*> &observer)

void **create_prompt_session_listener**(std::shared_ptr<*PromptSessionListener*> listener)

Class OpenGLContext

- Defined in file_include_miroil_miroil_open_gl_context.h

Class Documentation

class **OpenGLContext**

Public Functions

OpenGLContext(mir::graphics::GLConfig *gl_config)

void **operator()**(mir::Server &server)

auto **the_open_gl_config**() const -> std::shared_ptr<mir::graphics::GLConfig>

Class PersistDisplayConfig

- Defined in file_include_miroil_miroil_persist_display_config.h

Class Documentation

class **PersistDisplayConfig**

Restores the saved display configuration and saves changes to the base configuration.

Public Types

using **DisplayConfigurationPolicyWrapper** = std::function<std::shared_ptr<*DisplayConfigurationPolicy*>(std::shared_ptr<mir::graphics::DisplayConfigurationPolicy> const &wrapped)>

Public Functions

~PersistDisplayConfig()

PersistDisplayConfig(*PersistDisplayConfig* const&)

auto **operator=**(*PersistDisplayConfig* const&) -> *PersistDisplayConfig*&

PersistDisplayConfig(std::shared_ptr<*DisplayConfigurationStorage*> const &storage,
DisplayConfigurationPolicyWrapper const &custom_wrapper)

void **operator()**(mir::Server &server)

Class PromptSessionListener

- Defined in file_include_miroil_miroil_prompt_session_listener.h

Class Documentation

class **PromptSessionListener**

Public Functions

virtual **~PromptSessionListener**()

PromptSessionListener &**operator=**(*PromptSessionListener* const&) = delete

virtual void **starting**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) = 0

virtual void **stopping**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) = 0

virtual void **suspending**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) = 0

virtual void **resuming**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) = 0

virtual void **prompt_provider_added**(mir::scene::PromptSession const &prompt_session,
std::shared_ptr<mir::scene::Session> const &prompt_provider) = 0

virtual void **prompt_provider_removed**(mir::scene::PromptSession const &prompt_session,
std::shared_ptr<mir::scene::Session> const &prompt_provider) = 0

Protected Functions

PromptSessionListener() = default

PromptSessionListener(*PromptSessionListener* const&) = delete

Class PromptSessionManager

- Defined in file_include_miroil_miroil_prompt_session_manager.h

Class Documentation

class **PromptSessionManager**

Public Functions

PromptSessionManager(std::shared_ptr<mir::scene::PromptSessionManager> const &prompt_session_manager)

PromptSessionManager(*PromptSessionManager* const &src)

PromptSessionManager(*PromptSessionManager* &&src)

~PromptSessionManager()

auto **operator=**(*PromptSessionManager* const &src) -> *PromptSessionManager*&

auto **operator=**(*PromptSessionManager* &&src) -> *PromptSessionManager*&

bool **operator==**(*PromptSessionManager* const &other)

auto **application_for**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) const -> miral::*Application*

void **resume_prompt_session**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) const

void **stop_prompt_session**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) const

void **suspend_prompt_session**(std::shared_ptr<mir::scene::PromptSession> const &prompt_session) const

Class SetCompositor

- Defined in file_include_miroil_miroil_set_compositor.h

Class Documentation

class **SetCompositor**

Public Functions

SetCompositor(ConstructorFunction constructor, InitFunction init)

void **operator()**(mir::Server &server)

Class Surface

- Defined in file_include_miroil_miroil_surface.h

Class Documentation

class **Surface**

Public Functions

Surface(std::shared_ptr<mir::scene::Surface> wrapped)

~Surface() = default

mir::scene::Surface ***get_wrapped**() const

void **add_observer**(std::shared_ptr<miroil::*SurfaceObserver*> const &observer)

void **remove_observer**(std::shared_ptr<miroil::*SurfaceObserver*> const &observer)

mir::graphics::RenderableList **generate_renderables**(miroil::*CompositorID* id) const

bool **is_confined_to_window**()

void **set_orientation**(*MirOrientation* orientation)

void **set_confine_pointer_state**(*MirPointerConfinementState* state)

std::shared_ptr<mir::scene::Surface> **parent**() const

mir::geometry::*Point* **top_left**() const

Top-left corner (of the window frame if present)

bool **visible**() const

int **configure**(*MirWindowAttrib* attrib, int value)

int **query**(*MirWindowAttrib* attrib) const

void **set_keymap**(*MirInputDeviceId* id, std::string const &model, std::string const &layout, std::string const &variant, std::string const &options)

Class SurfaceObserver

- Defined in file_include_miroil_miroil_surface_observer.h

Class Documentation

class **SurfaceObserver**

Public Functions

SurfaceObserver() = default

SurfaceObserver(*SurfaceObserver* const&) = delete

SurfaceObserver &**operator=**(*SurfaceObserver* const&) = delete

virtual **~SurfaceObserver**() = default

virtual void **attrib_changed**(mir::scene::Surface const *surf, *MirWindowAttrib* attrib, int value) = 0

virtual void **window_resized_to**(mir::scene::Surface const *surf, mir::geometry::*Size* const &window_size) = 0

virtual void **content_resized_to**(mir::scene::Surface const *surf, mir::geometry::*Size* const &content_size) = 0

virtual void **moved_to**(mir::scene::Surface const *surf, mir::geometry::*Point* const &top_left) = 0

virtual void **hidden_set_to**(mir::scene::Surface const *surf, bool hide) = 0

virtual void **frame_posted**(mir::scene::Surface const *surf, int frames_available, mir::geometry::*Size* const &size) = 0

virtual void **alpha_set_to**(mir::scene::Surface const *surf, float alpha) = 0

virtual void **orientation_set_to**(mir::scene::Surface const *surf, *MirOrientation* orientation) = 0

virtual void **transformation_set_to**(mir::scene::Surface const *surf, glm::mat4 const &t) = 0

virtual void **cursor_image_set_to**(mir::scene::Surface const *surf, mir::graphics::CursorImage const &image) = 0

virtual void **client_surface_close_requested**(mir::scene::Surface const *surf) = 0

virtual void **keymap_changed**(mir::scene::Surface const *surf, *MirInputDeviceId* id, std::string const &model, std::string const &layout, std::string const &variant, std::string const &options) = 0

virtual void **renamed**(mir::scene::Surface const *surf, char const *name) = 0

virtual void **cursor_image_removed**(mir::scene::Surface const *surf) = 0

virtual void **placed_relative**(mir::scene::Surface const *surf, mir::geometry::*Rectangle* const &placement) = 0

virtual void **input_consumed**(mir::scene::Surface const *surf, *MirEvent* const *event) = 0

virtual void **start_drag_and_drop**(mir::scene::Surface const *surf, std::vector<uint8_t> const &handle) = 0

virtual void **depth_layer_set_to**(mir::scene::Surface const *surf, *MirDepthLayer* depth_layer) = 0

virtual void **application_id_set_to**(mir::scene::Surface const *surf, std::string const &application_id) = 0

Class SpinnerSplash

- Defined in file_examples_miral-shell_spinner_splash.h

Class Documentation

class **SpinnerSplash**

Public Functions

SpinnerSplash()

~SpinnerSplash()

void **operator()**(struct wl_display *display)

void **operator()**(std::weak_ptr<mir::scene::Session> const &session)

operator std::shared_ptr<*SplashSession*>() const

Class SplashSession

- Defined in file_examples_example-server-lib_splash_session.h

Class Documentation

class **SplashSession**

Public Functions

virtual auto **session**() const -> std::shared_ptr<mir::scene::Session> = 0

SplashSession() = default

virtual **~SplashSession**() = default

SplashSession(*SplashSession* const&) = delete

SplashSession &**operator=**(*SplashSession* const&) = delete

Class SwSplash

- Defined in file_examples_example-server-lib_sw_splash.h

Class Documentation

class **SwSplash**

Public Functions

SwSplash()

~SwSplash()

void **operator()**(struct wl_display *display)

void **operator()**(std::weak_ptr<mir::scene::Session> const &session)

void **enable**(bool show_splash_opt)

operator std::shared_ptr<*SplashSession*>() const

Class TilingWindowManagerPolicy

- Defined in file_examples_example-server-lib_tiling_window_manager.h

Nested Relationships

Nested Types

- *Class TilingWindowManagerPolicy::MRUTileList*

Inheritance Relationships

Base Type

- `public miral::WindowManagementPolicy` (*Class WindowManagementPolicy*)

Class Documentation

class **TilingWindowManagerPolicy** : public miral::*WindowManagementPolicy*

Public Functions

explicit **TilingWindowManagerPolicy**(miral::*WindowManagerTools* const &tools,
std::shared_ptr<*SplashSession*> const &spinner,
miral::*InternalClientLauncher* const &launcher)

virtual auto **place_new_window**(miral::*ApplicationInfo* const &app_info, miral::*WindowSpecification* const &request_parameters) -> miral::*WindowSpecification* override

Customize initial window placement.

Parameters

- **app_info** – the application requesting a new window
- **requested_specification** – the requested specification (updated with default placement)

Returns

the customized specification

virtual void **handle_window_ready**(miral::*WindowInfo* &window_info) override

notification that the first buffer has been posted

Parameters

window_info – the window

virtual void **handle_modify_window**(miral::*WindowInfo* &window_info, miral::*WindowSpecification* const &modifications) override

request from client to modify the window specification.

Parameters

- **window_info** – the window
- **modifications** – the requested changes

> **Note**
>
> the request has already been validated against the type definition

virtual bool **handle_keyboard_event**(MirKeyboardEvent const *event) override

keyboard event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_touch_event**(MirTouchEvent const *event) override

touch event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual bool **handle_pointer_event**(MirPointerEvent const *event) override

pointer event handler

Parameters

event – the event

Returns

whether the policy has consumed the event

virtual void **handle_raise_window**(miral::*WindowInfo* &window_info) override

request from client to raise the window

Parameters

window_info – the window

> **Note**
>
> the request has already been validated against the requesting event

virtual void **advise_end**() override

after any related calls end

virtual void **advise_new_window**(miral::*WindowInfo* const &window_info) override

Notification that a window has been created.

Parameters

window_info – the window

virtual void **advise_focus_gained**(miral::*WindowInfo* const &info) override

Notification that a window has gained focus.

Parameters

window_info – the window

virtual void **advise_new_app**(miral::*ApplicationInfo* &application) override

Notification that a new application has connected.

Parameters

application – the application

virtual void **advise_delete_app**(miral::*ApplicationInfo* const &application) override

Notification that an application has disconnected.

Parameters

application – the application

virtual void **handle_request_move**(miral::*WindowInfo* &window_info, MirInputEvent const *input_event) override

request from client to initiate move

Parameters

- **window_info** – the window
- **input_event** – the requesting event

> **Note**
>
> the request has already been validated against the requesting event

virtual void **handle_request_resize**(miral::*WindowInfo* &window_info, MirInputEvent const *input_event, *MirResizeEdge* edge) override

request from client to initiate resize

Parameters

- **window_info** – the window
- **input_event** – the requesting event
- **edge** – the edge(s) being dragged

> **Note**
>
> the request has already been validated against the requesting event

virtual auto **confirm_inherited_move**(miral::*WindowInfo* const &window_info, Displacement movement) -> Rectangle override

Confirm (and optionally adjust) the motion of a child window when the parent is moved.

Parameters

- **window_info** – the window
- **movement** – the movement of the parent

Returns

the confirmed placement of the window

virtual Rectangle **confirm_placement_on_display**(const miral::*WindowInfo* &window_info, *MirWindowState* new_state, Rectangle const &new_placement) override

Confirm (and optionally adjust) the placement of a window on the display.

Called when (re)placing fullscreen, maximized, horizontally maximised and vertically maximized windows to allow adjustment for decorations.

Parameters

- **window_info** – the window
- **new_state** – the new state
- **new_placement** – the suggested placement

Returns

the confirmed placement of the window

Class TilingWindowManagerPolicy::MRUTileList

- Defined in file_examples_example-server-lib_tiling_window_manager.h

Nested Relationships

This class is a nested type of *Class TilingWindowManagerPolicy*.

Class Documentation

class **MRUTileList**

Public Types

using **Enumerator** = std::function<void(std::shared_ptr<void> const &tile)>

Public Functions

void **push**(std::shared_ptr<void> const &tile)

void **erase**(std::shared_ptr<void> const &tile)

void **enumerate**(*Enumerator* const &enumerator) const

inline auto **count**() -> size_t

Class WaylandApp

- Defined in file_examples_example-server-lib_wayland_app.h

Class Documentation

class **WaylandApp**

Public Functions

WaylandApp()

WaylandApp(wl_display *display)

virtual **~WaylandApp**() = default

void **wayland_init**(wl_display *display)

Needs to be two-step initialized to virtual methods are called.

void **roundtrip**() const

inline auto **display**() const -> wl_display*

inline auto **compositor**() const -> wl_compositor*

inline auto **shm**() const -> wl_shm*

inline auto **seat**() const -> wl_seat*

inline auto **shell**() const -> wl_shell*

Protected Functions

inline virtual void **output_ready**(*WaylandOutput* const*)

inline virtual void **output_changed**(*WaylandOutput* const*)

inline virtual void **output_gone**(*WaylandOutput* const*)

Protected Attributes

friend WaylandOutput

Class WaylandCallback

- Defined in file_examples_example-server-lib_wayland_app.h

Class Documentation

class **WaylandCallback**

Public Static Functions

static void **create**(wl_callback *callback, std::function<void()> &&func)
Takes ownership of callback.

Template Class WaylandObject

- Defined in file_examples_example-server-lib_wayland_app.h

Class Documentation

template<typename **T**>

class **WaylandObject**

Public Functions

inline **WaylandObject**()

inline **WaylandObject**(*T* *proxy, void (*destroy)(*T**))

inline **operator** *T***() const

Class WaylandOutput

- Defined in file_examples_example-server-lib_wayland_app.h

Class Documentation

class **WaylandOutput**

Public Functions

WaylandOutput(*WaylandApp* *app, wl_output *output)

virtual **~WaylandOutput**() = default

inline auto **scale**() const -> int

inline auto **transform**() const -> int

inline auto **operator==**(wl_output *other) const -> bool

inline auto **wl**() const -> wl_output*

Class WaylandShm

- Defined in file_examples_example-server-lib_wayland_shm.h

Class Documentation

class **WaylandShm**

A single *WaylandShm* does not efficiently provision multiple buffers for multiple window sizes. Please use one *WaylandShm* per window (if windows may have distinct sizes)

Public Functions

WaylandShm(wl_shm *shm)

Does not take ownership of the wl_shm.

auto **get_buffer**(mir::geometry::*Size* size, mir::geometry::*Stride* stride) -> std::shared_ptr<*WaylandShmBuffer*>

Always returns a buffer of the correct size that is not in-use.

Class WaylandShmBuffer

- Defined in file_examples_example-server-lib_wayland_shm.h

Inheritance Relationships

Base Type

- `public std::enable_shared_from_this< WaylandShmBuffer >`

Class Documentation

class **WaylandShmBuffer** : public std::enable_shared_from_this<*WaylandShmBuffer*>

Public Functions

WaylandShmBuffer(std::shared_ptr<WaylandShmPool> pool, void *data, mir::geometry::*Size* size, mir::geometry::*Stride* stride, wl_buffer *buffer)

~WaylandShmBuffer()

inline auto **data**() const -> void*

inline auto **size**() const -> mir::geometry::*Size*

inline auto **stride**() const -> mir::geometry::*Stride*

inline auto **is_in_use**() const -> bool

Returns if this buffer is currently being used by the compositor. In-use buffers keep themselves alive.

auto **use**() -> wl_buffer*

Marks this buffer as in-use and assumes the resulting wl_buffer is sent to the compositor. Keeps this buffer alive until the compositor releases it (if it's not sent to the compositor or the compositor never releases it this buffer is leaked). Should only be called if the buffer is not already in-use.

Class WaylandSurface

- Defined in file_examples_example-server-lib_wayland_surface.h

Inheritance Relationships

Derived Type

- `private MirEglSurface` (*Class MirEglSurface*)

Class Documentation

class **WaylandSurface**

Subclassed by *MirEglSurface*

Public Functions

WaylandSurface(*WaylandApp* const *app)

virtual **~WaylandSurface**() = default

void **attach_buffer**(wl_buffer *buffer, int scale)

void **commit**() const

void **set_fullscreen**(wl_output *output)

output can be null, user needs to commit after

void **add_frame_callback**(std::function<void()> &&func)

inline auto **app**() const -> *WaylandApp* const*

inline auto **surface**() const -> wl_surface*

inline auto **configured_size**() const -> mir::geometry::*Size*

Protected Functions

inline virtual void **configured**()

Called when the compositor configures this shell surface.

Enums

Enum @0

- Defined in file_include_core_mir_toolkit_mir_native_buffer.h

Enum Documentation

enum **[anonymous]**

Values:

enumerator **mir_buffer_package_max**

Enum MirBufferFlag

- Defined in file_include_core_mir_toolkit_mir_native_buffer.h

Enum Documentation

enum **MirBufferFlag**

Values:

enumerator **mir_buffer_flag_can_scanout**

enumerator **mir_buffer_flag_fenced**

Enum MirDepthLayer

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirDepthLayer**

Depth layer controls Z ordering of surfaces.

A surface will always appear on top of surfaces with a lower depth layer, and below those with a higher one. A depth layer can be converted to a number with *mir::mir_depth_layer_get_index()*. This is useful for creating a list indexed by depth layer, or comparing the height of two layers.

Values:

enumerator **mir_depth_layer_background**

For desktop backgrounds and alike (lowest layer)

enumerator **mir_depth_layer_below**

For panels or other controls/decorations below normal windows.

enumerator **mir_depth_layer_application**

For normal application windows.

enumerator **mir_depth_layer_always_on_top**

For always-on-top application windows.

enumerator **mir_depth_layer_above**

For panels or notifications that want to be above normal windows.

enumerator **mir_depth_layer_overlay**

For overlays such as lock screens (heighest layer)

Enum MirEdgeAttachment

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirEdgeAttachment**

Values:

enumerator **mir_edge_attachment_vertical**

enumerator **mir_edge_attachment_horizontal**

enumerator **mir_edge_attachment_any**

Enum MirEventType

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirEventType**

Values:

enumerator **mir_event_type_key**

enumerator **mir_event_type_motion**

enumerator **mir_event_type_window**

enumerator **mir_event_type_resize**

enumerator **mir_event_type_prompt_session_state_change**

enumerator **mir_event_type_orientation**

enumerator **mir_event_type_close_window**

enumerator **mir_event_type_input**

enumerator **mir_event_type_input_configuration**

enumerator **mir_event_type_window_output**

enumerator **mir_event_type_input_device_state**

enumerator **mir_event_type_window_placement**

Enum MirFocusMode

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirFocusMode**

Focus mode controls how a surface gains and loses focus.

Values:

enumerator **mir_focus_mode_focusable**

The surface can gain and lose focus normally.

enumerator **mir_focus_mode_disabled**

The surface will never be given focus.

enumerator **mir_focus_mode_grabbing**

This mode causes the surface to take focus if possible, and prevents focus from leaving it as long as it has this mode.

Enum MirFormFactor

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirFormFactor**

Form factor associated with a physical output.

Values:

enumerator **mir_form_factor_unknown**

enumerator **mir_form_factor_phone**

enumerator **mir_form_factor_tablet**

enumerator **mir_form_factor_monitor**

enumerator **mir_form_factor_tv**

enumerator **mir_form_factor_projector**

Enum MirInputEventModifier

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirInputEventModifier**

Description of key modifier state.

Values:

enumerator **mir_input_event_modifier_none**

enumerator **mir_input_event_modifier_alt**

enumerator **mir_input_event_modifier_alt_left**

enumerator **mir_input_event_modifier_alt_right**

enumerator **mir_input_event_modifier_shift**

enumerator **mir_input_event_modifier_shift_left**

enumerator **mir_input_event_modifier_shift_right**

enumerator **mir_input_event_modifier_sym**

enumerator **mir_input_event_modifier_function**

enumerator **mir_input_event_modifier_ctrl**

enumerator **mir_input_event_modifier_ctrl_left**

enumerator **mir_input_event_modifier_ctrl_right**

enumerator **mir_input_event_modifier_meta**

enumerator **mir_input_event_modifier_meta_left**

enumerator **mir_input_event_modifier_meta_right**

enumerator **mir_input_event_modifier_caps_lock**

enumerator **mir_input_event_modifier_num_lock**

enumerator **mir_input_event_modifier_scroll_lock**

Enum MirInputEventType

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirInputEventType**

Values:

enumerator **mir_input_event_type_key**

enumerator **mir_input_event_type_touch**

enumerator **mir_input_event_type_pointer**

enumerator **mir_input_event_type_keyboard_resync**

enumerator **mir_input_event_types**

Enum MirKeyboardAction

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirKeyboardAction**

Possible actions for changing key state.

Values:

enumerator **mir_keyboard_action_up**

enumerator **mir_keyboard_action_down**

enumerator **mir_keyboard_action_repeat**

enumerator **mir_keyboard_action_modifiers**

enumerator **mir_keyboard_actions**

Enum MirLifecycleState

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirLifecycleState**

Values:

enumerator **mir_lifecycle_state_will_suspend**

enumerator **mir_lifecycle_state_resumed**

enumerator **mir_lifecycle_connection_lost**

Enum MirMirrorMode

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirMirrorMode**

Mirroring axis relative to the "natural" orientation of the display.

Values:

enumerator **mir_mirror_mode_none**

enumerator **mir_mirror_mode_vertical**

enumerator **mir_mirror_mode_horizontal**

Enum MirOrientation

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirOrientation**

Direction relative to the "natural" orientation of the display.

Values:

enumerator **mir_orientation_normal**

enumerator **mir_orientation_left**

enumerator **mir_orientation_inverted**

enumerator **mir_orientation_right**

Enum MirOrientationMode

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirOrientationMode**

Values:

enumerator **mir_orientation_mode_portrait**

enumerator **mir_orientation_mode_landscape**

enumerator **mir_orientation_mode_portrait_inverted**

enumerator **mir_orientation_mode_landscape_inverted**

enumerator **mir_orientation_mode_portrait_any**

enumerator **mir_orientation_mode_landscape_any**

enumerator **mir_orientation_mode_any**

Enum MirOutputGammaSupported

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirOutputGammaSupported**

Supports gamma correction.

Values:

enumerator **mir_output_gamma_unsupported**

enumerator **mir_output_gamma_supported**

Enum MirOutputType

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirOutputType**

Values:

enumerator **mir_output_type_unknown**

enumerator **mir_output_type_vga**

enumerator **mir_output_type_dvii**

enumerator **mir_output_type_dvid**

enumerator **mir_output_type_dvia**

enumerator **mir_output_type_composite**

enumerator **mir_output_type_svideo**

enumerator **mir_output_type_lvds**

enumerator **mir_output_type_component**

enumerator **mir_output_type_ninepindin**

enumerator **mir_output_type_displayport**

enumerator **mir_output_type_hdmia**

enumerator **mir_output_type_hdmib**

enumerator **mir_output_type_tv**

enumerator **mir_output_type_edp**

enumerator **mir_output_type_virtual**

enumerator **mir_output_type_dsi**

enumerator **mir_output_type_dpi**

Enum MirPixelFormat

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirPixelFormat**

32-bit pixel formats (8888): The order of components in the enum matches the order of the components as they would be written in an integer representing a pixel value of that format.

For example; abgr_8888 should be coded as 0xAABBGGRR, which will end up as R,G,B,A in memory on a little endian system, and as A,B,G,R on a big endian system.

24-bit pixel formats (888): These are in literal byte order, regardless of CPU architecture it's always the same. Writing these 3-byte pixels is typically slower than other formats but uses less memory than 32-bit and is endian-independent.

16-bit pixel formats (565/5551/4444): Always interpreted as one 16-bit integer per pixel with components in high-to-low bit order following the format name. These are the fastest formats, however colour quality is visibly lower.

Values:

enumerator **mir_pixel_format_invalid**

enumerator **mir_pixel_format_abgr_8888**

enumerator **mir_pixel_format_xbgr_8888**

enumerator **mir_pixel_format_argb_8888**

enumerator **mir_pixel_format_xrgb_8888**

enumerator **mir_pixel_format_bgr_888**

enumerator **mir_pixel_format_rgb_888**

enumerator **mir_pixel_format_rgb_565**

enumerator **mir_pixel_format_rgba_5551**

enumerator **mir_pixel_format_rgba_4444**

enumerator **mir_pixel_formats**

Enum MirPlacementGravity

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirPlacementGravity**

Reference point for aligning a surface relative to a rectangle.

Each element (surface and rectangle) has a MirPlacementGravity assigned.

Values:

enumerator **mir_placement_gravity_center**

the reference point is at the center.

enumerator **mir_placement_gravity_west**

the reference point is at the middle of the left edge.

enumerator **mir_placement_gravity_east**

the reference point is at the middle of the right edge.

enumerator **mir_placement_gravity_north**

the reference point is in the middle of the top edge.

enumerator **mir_placement_gravity_south**

the reference point is at the middle of the lower edge.

enumerator **mir_placement_gravity_northwest**

the reference point is at the top left corner.

enumerator **mir_placement_gravity_northeast**

the reference point is at the top right corner.

enumerator **mir_placement_gravity_southwest**

the reference point is at the lower left corner.

enumerator **mir_placement_gravity_southeast**

the reference point is at the lower right corner.

Enum MirPlacementHints

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirPlacementHints**

Positioning hints for aligning a window relative to a rectangle.

These hints determine how the window should be positioned in the case that the surface would fall off-screen if placed in its ideal position.

For example, `mir_placement_hints_flip_x` will invert the x component of `aux_rect_placement_offset` and replace `mir_placement_gravity_northwest` with `mir_placement_gravity_northeast` and vice versa if the window extends beyond the left or right edges of the monitor.

If `mir_placement_hints_slide_x` is set, the window can be shifted horizontally to fit on-screen.

If `mir_placement_hints_resize_x` is set, the window can be shrunken horizontally to fit.

If `mir_placement_hints_antipodes` is set then the rect gravity may be substituted with the opposite corner (e.g. `mir_placement_gravity_northeast` to `mir_placement_gravity_southwest`) in combination with other options.

When multiple flags are set, flipping should take precedence over sliding, which should take precedence over resizing.

Values:

enumerator **mir_placement_hints_flip_x**

allow flipping anchors horizontally

enumerator **mir_placement_hints_flip_y**

allow flipping anchors vertically

enumerator **mir_placement_hints_slide_x**

allow sliding window horizontally

enumerator **mir_placement_hints_slide_y**
allow sliding window vertically

enumerator **mir_placement_hints_resize_x**
allow resizing window horizontally

enumerator **mir_placement_hints_resize_y**
allow resizing window vertically

enumerator **mir_placement_hints_antipodes**
allow flipping aux_anchor to opposite corner

enumerator **mir_placement_hints_flip_any**
allow flipping anchors on both axes

enumerator **mir_placement_hints_slide_any**
allow sliding window on both axes

enumerator **mir_placement_hints_resize_any**
allow resizing window on both axes

Enum MirPointerAcceleration

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Enum Documentation

enum **MirPointerAcceleration**
MirPointerAcceleration describes the way pointer movement is filtered:

- mir_pointer_acceleration_none: (acceleration bias + 1.0) is applied as a factor to the current velocity of the pointer. So a bias of 0 to results to no change of velocity.
- mir_pointer_acceleration_adaptive: acceleration bias selects an acceleration function based on the current velocity that usually consists of two linear inclines separated by a plateau.

Values:

enumerator **mir_pointer_acceleration_none**

enumerator **mir_pointer_acceleration_adaptive**

Enum MirPointerAction

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirPointerAction**

Possible pointer actions.

Values:

enumerator **mir_pointer_action_button_up**

enumerator **mir_pointer_action_button_down**

enumerator **mir_pointer_action_enter**

enumerator **mir_pointer_action_leave**

enumerator **mir_pointer_action_motion**

enumerator **mir_pointer_actions**

Enum MirPointerAxis

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirPointerAxis**

Identifiers for pointer axis.

Values:

enumerator **mir_pointer_axis_x**

enumerator **mir_pointer_axis_y**

enumerator **mir_pointer_axis_vscroll**

enumerator **mir_pointer_axis_hscroll**

enumerator **mir_pointer_axis_relative_x**

enumerator **mir_pointer_axis_relative_y**

enumerator **mir_pointer_axis_vscroll_discrete**

enumerator **mir_pointer_axis_hscroll_discrete**

enumerator **mir_pointer_axis_vscroll_value120**

enumerator **mir_pointer_axis_hscroll_value120**

enumerator **mir_pointer_axes**

Enum MirPointerAxisSource

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirPointerAxisSource**

Identifiers for pointer event source.

Values:

enumerator **mir_pointer_axis_source_none**

enumerator **mir_pointer_axis_source_wheel**

enumerator **mir_pointer_axis_source_finger**

enumerator **mir_pointer_axis_source_continuous**

enumerator **mir_pointer_axis_source_wheel_tilt**

Enum MirPointerButton

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirPointerButton**

Values:

enumerator **mir_pointer_button_primary**

enumerator **mir_pointer_button_secondary**

enumerator **mir_pointer_button_tertiary**

enumerator **mir_pointer_button_back**

enumerator **mir_pointer_button_forward**

enumerator **mir_pointer_button_side**

enumerator **mir_pointer_button_extra**

enumerator **mir_pointer_button_task**

Enum MirPointerConfinementState

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirPointerConfinementState**

Pointer Confinement.

Values:

enumerator **mir_pointer_unconfined**

enumerator **mir_pointer_confined_oneshot**

enumerator **mir_pointer_confined_persistent**

enumerator **mir_pointer_locked_oneshot**

enumerator **mir_pointer_locked_persistent**

Enum MirPointerHandedness

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Enum Documentation

enum **MirPointerHandedness**

Values:

enumerator **mir_pointer_handedness_right**

enumerator **mir_pointer_handedness_left**

Enum MirPowerMode

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirPowerMode**

Values:

enumerator **mir_power_mode_on**

enumerator **mir_power_mode_standby**

enumerator **mir_power_mode_suspend**

enumerator **mir_power_mode_off**

Enum MirPromptSessionState

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirPromptSessionState**

Values:

enumerator **mir_prompt_session_state_stopped**

enumerator **mir_prompt_session_state_started**

enumerator **mir_prompt_session_state_suspended**

Enum MirResizeEdge

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirResizeEdge**

Hints for resizing a window.

These values are used to indicate which edge(s) of a surface is being dragged in a resize operation.

Values:

enumerator **mir_resize_edge_none**

enumerator **mir_resize_edge_west**

enumerator **mir_resize_edge_east**

enumerator **mir_resize_edge_north**

enumerator **mir_resize_edge_south**

enumerator **mir_resize_edge_northwest**

enumerator **mir_resize_edge_northeast**

enumerator **mir_resize_edge_southwest**

enumerator **mir_resize_edge_southeast**

Enum MirShellChrome

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirShellChrome**

Shell chrome.

Values:

enumerator **mir_shell_chrome_normal**

enumerator **mir_shell_chrome_low**

Enum MirSubpixelArrangement

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirSubpixelArrangement**

Physical arrangement of subpixels on the physical output.

This is always relative to the "natural" orientation of the display - mir_orientation_normal.

Values:

enumerator **mir_subpixel_arrangement_unknown**

Arrangement of subpixels cannot be determined.

enumerator **mir_subpixel_arrangement_horizontal_rgb**

Subpixels are arranged horizontally, R, G, B from left to right.

enumerator **mir_subpixel_arrangement_horizontal_bgr**

Subpixels are arranged horizontally, B, G, R from left to right.

enumerator **mir_subpixel_arrangement_vertical_rgb**

Subpixels are arranged vertically, R, G, B from top to bottom.

enumerator **mir_subpixel_arrangement_vertical_bgr**

Subpixels are arranged vertically, B, G, R from top to bottom.

enumerator **mir_subpixel_arrangement_none**

Device does not have regular subpixels.

Enum MirTouchAction

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirTouchAction**

Possible per touch actions for state changing.

Values:

enumerator **mir_touch_action_up**

enumerator **mir_touch_action_down**

enumerator **mir_touch_action_change**

enumerator **mir_touch_actions**

Enum MirTouchAxis

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirTouchAxis**

Identifiers for touch axis.

Values:

enumerator **mir_touch_axis_x**

enumerator **mir_touch_axis_y**

enumerator **mir_touch_axis_pressure**

enumerator **mir_touch_axis_touch_major**

enumerator **mir_touch_axis_touch_minor**

enumerator **mir_touch_axis_size**

enumerator **mir_touch_axes**

Enum MirTouchpadClickMode

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Enum Documentation

enum **MirTouchpadClickMode**

MirTouchpadClickMode configures how the touchpad itself should generate pointer button events.

The available click modes may be active simultaneously.
- mir_touchpad_click_mode_none: no active click mode
- mir_touchpad_click_mode_area_to_click: simulate pointer buttons using click areas on the touchpad
- mir_touchpad_click_mode_finger_count: simulate pointer buttons using the number of fingers down

Values:

enumerator **mir_touchpad_click_mode_none**

enumerator **mir_touchpad_click_mode_area_to_click**

enumerator **mir_touchpad_click_mode_finger_count**

Enum MirTouchpadScrollMode

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Enum Documentation

enum **MirTouchpadScrollMode**

MirTouchpadScrollMode configures how the touchpad should generate scroll events.

- mir_touchpad_scroll_mode_none: no scroll
- mir_touchpad_scroll_mode_two_finger_scroll: two finger movement generates generates vertical and horizontal scroll events
- mir_touchpad_scroll_mode_edge_scroll: touch movement at the edge of the touchpad genertes scroll events
- mir_touchpad_scroll_mode_button_down_scroll: movement on the touchpad generates scroll events when a button is held down simultaneously

Values:

enumerator **mir_touchpad_scroll_mode_none**

enumerator **mir_touchpad_scroll_mode_two_finger_scroll**

enumerator **mir_touchpad_scroll_mode_edge_scroll**

enumerator **mir_touchpad_scroll_mode_button_down_scroll**

Enum MirTouchscreenMappingMode

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Enum Documentation

enum **MirTouchscreenMappingMode**

Mapping modes for touchscreen devices.

The mode defines how coordinates from the touchscreen frequently referred to as device coordinates are translated into scene coordinates.

This configuration mode is relevant for different classes of input devices, i.e handheld devices with builtin touchscreens or external graphic tablets or external monitors with touchscreen capabilities.

Values:

enumerator **mir_touchscreen_mapping_mode_to_output**

Map the device coordinates onto specific output.

enumerator **mir_touchscreen_mapping_mode_to_display_wall**

Map the device coordinates onto the whole wall of outputs.

Enum MirTouchTooltype

- Defined in file_include_core_mir_toolkit_events_enums.h

Enum Documentation

enum **MirTouchTooltype**

Identifiers for per-touch tool types.

Values:

enumerator **mir_touch_tooltype_unknown**

enumerator **mir_touch_tooltype_finger**

enumerator **mir_touch_tooltype_stylus**

enumerator **mir_touch_tooltypes**

Enum MirWindowAttrib

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirWindowAttrib**

Attributes of a window that the client and server/shell may wish to get or set over the wire.

Values:

enumerator **mir_window_attrib_type**

enumerator **mir_window_attrib_state**

enumerator **mir_window_attrib_focus**

enumerator **mir_window_attrib_dpi**

enumerator **mir_window_attrib_visibility**

enumerator **mir_window_attrib_preferred_orientation**

enumerator **mir_window_attribs**

Enum MirWindowFocusState

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirWindowFocusState**

Values:

enumerator **mir_window_focus_state_unfocused**

Inactive and does not have focus

enumerator **mir_window_focus_state_focused**

Active and has keybaord focus

enumerator **mir_window_focus_state_active**

Active but does not have keyboard focus

Enum MirWindowState

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirWindowState**

Values:

enumerator **mir_window_state_unknown**

enumerator **mir_window_state_restored**

enumerator **mir_window_state_minimized**

enumerator **mir_window_state_maximized**

enumerator **mir_window_state_vertmaximized**

enumerator **mir_window_state_fullscreen**

enumerator **mir_window_state_horizmaximized**

enumerator **mir_window_state_hidden**

enumerator **mir_window_state_attached**

Used for panels, notifications and other windows attached to output edges.

enumerator **mir_window_states**

Enum MirWindowType

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirWindowType**

Values:

enumerator **mir_window_type_normal**

AKA “regular”

enumerator **mir_window_type_utility**
AKA “floating”

enumerator **mir_window_type_dialog**

enumerator **mir_window_type_gloss**

enumerator **mir_window_type_freestyle**

enumerator **mir_window_type_menu**

enumerator **mir_window_type_inputmethod**
AKA “OSK” or handwriting etc.

enumerator **mir_window_type_satellite**
AKA “toolbox”/”toolbar”

enumerator **mir_window_type_tip**
AKA “tooltip”

enumerator **mir_window_type_decoration**

enumerator **mir_window_types**

Enum MirWindowVisibility

- Defined in file_include_core_mir_toolkit_common.h

Enum Documentation

enum **MirWindowVisibility**
Values:

enumerator **mir_window_visibility_occluded**

enumerator **mir_window_visibility_exposed**

Unions

Union Descriptor::Value

- Defined in file_include_miroil_miroil_edid.h

Nested Relationships

This union is a nested type of *Struct Edid::Descriptor*.

Union Documentation

union **Value**

#include <edid.h>

Public Members

char **monitor_name**[13]

char **unspecified_text**[13]

char **serial_number**[13]

Functions

Function make_mir_eglapp

- Defined in file_examples_miral-shell_spinner_miregl.h

Function Documentation

std::shared_ptr<MirEglApp> **make_mir_eglapp**(struct wl_display *display)

Function mir::fatal_error_abort

- Defined in file_include_core_mir_fatal.h

Function Documentation

void mir::**fatal_error_abort**(char const *reason, ...)

An alternative to *fatal_error_except()* that kills the program and dump core as cleanly as possible.

Parameters

reason – **[in]** A printf-style format string.

Function mir::fatal_error_except

- Defined in file_include_core_mir_fatal.h

Function Documentation

void mir::**fatal_error_except**(char const *reason, ...)

Throws an exception that will typically kill the Mir server and propagate from mir::run_mir.

Parameters

reason – **[in]** A printf-style format string.

Template Function mir::geometry::as_delta(generic::X<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*DeltaX*<*T*> mir::geometry::**as_delta**(generic::*X*<*T*> const &x)

Template Function mir::geometry::as_delta(generic::Y<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*DeltaY*<*T*> mir::geometry::**as_delta**(generic::*Y*<*T*> const &y)

Template Function mir::geometry::as_delta(generic::Width<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*DeltaX*<*T*> `mir::geometry::`**`as_delta`**(generic::*Width*<*T*> const &w)

Template Function mir::geometry::as_delta(generic::Height<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*DeltaY*<*T*> `mir::geometry::`**`as_delta`**(generic::*Height*<*T*> const &h)

Template Function mir::geometry::as_height(generic::DeltaY<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*Height*<*T*> `mir::geometry::`**`as_height`**(generic::*DeltaY*<*T*> const &dy)

Template Function mir::geometry::as_height(generic::Y<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*Height*<*T*> `mir::geometry::`**`as_height`**(generic::*Y*<*T*> const &y)

Template Function mir::geometry::as_width(generic::DeltaX<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*Width*<*T*> `mir::geometry::`**`as_width`**(generic::*DeltaX*<*T*> const &dx)

Template Function mir::geometry::as_width(generic::X<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*Width*<*T*> `mir::geometry::`**`as_width`**(generic::*X*<*T*> const &x)

Template Function mir::geometry::as_x(generic::DeltaX<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*X*<*T*> `mir::geometry::`**`as_x`**(generic::*DeltaX*<*T*> const &dx)

Template Function mir::geometry::as_x(generic::Width<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*X*<*T*> `mir::geometry::`**`as_x`**(generic::*Width*<*T*> const &w)

Template Function mir::geometry::as_y(generic::DeltaY<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*Y*<*T*> `mir::geometry::`**`as_y`**(generic::*DeltaY*<*T*> const &dy)

Template Function mir::geometry::as_y(generic::Height<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr generic::*Y*<*T*> `mir::geometry::`**`as_y`**(generic::*Height*<*T*> const &h)

Template Function mir::geometry::generic::as_displacement(Size<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::`**`as_displacement`**(*Size*<*T*> const &size)

Template Function mir::geometry::generic::as_displacement(Point<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::`**`as_displacement`**(*Point*<*T*> const &point)

Template Function mir::geometry::generic::as_point(Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::`**`as_point`**(*Displacement*<*T*> const &disp)

Template Function mir::geometry::generic::as_point(Size<T> const&)

- Defined in file_include_core_mir_geometry_size.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::`**`as_point`**(*Size*<*T*> const &size)

Template Function mir::geometry::generic::as_size(Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Size*<*T*> `mir::geometry::generic::`**`as_size`**(*Displacement*<*T*> const &disp)

Template Function mir::geometry::generic::as_size(Point<T> const&)

- Defined in file_include_core_mir_geometry_size.h

Function Documentation

template<typename **T**>
inline constexpr *Size*<*T*> `mir::geometry::generic::`**`as_size`**(*Point*<*T*> const &point)

Template Function mir::geometry::generic::intersection_of

- Defined in file_include_core_mir_geometry_rectangle.h

Function Documentation

template<typename **T**>
Rectangle<*T*> mir::geometry::generic::**intersection_of**(*Rectangle*<*T*> const &a, *Rectangle*<*T*> const &b)

Template Function mir::geometry::generic::operator*(Scalar, Width<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Width*<*T*> mir::geometry::generic::**operator***(*Scalar* scale, *Width*<*T*> const &w)

Template Function mir::geometry::generic::operator*(Scalar, Height<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Height*<*T*> mir::geometry::generic::**operator***(*Scalar* scale, *Height*<*T*> const &h)

Template Function mir::geometry::generic::operator*(Scalar, DeltaX<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *DeltaX*<*T*> mir::geometry::generic::**operator***(*Scalar* scale, *DeltaX*<*T*> const &dx)

Template Function mir::geometry::generic::operator*(Scalar, DeltaY<T> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *DeltaY*<*T*> `mir::geometry::generic::`**`operator*`**(*Scalar* scale, *DeltaY*<*T*> const &dy)

Template Function mir::geometry::generic::operator*(Width<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Width*<*T*> `mir::geometry::generic::`**`operator*`**(*Width*<*T*> const &w, *Scalar* scale)

Template Function mir::geometry::generic::operator*(Height<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Height*<*T*> `mir::geometry::generic::`**`operator*`**(*Height*<*T*> const &h, *Scalar* scale)

Template Function mir::geometry::generic::operator*(DeltaX<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *DeltaX*<*T*> `mir::geometry::generic::`**`operator*`**(*DeltaX*<*T*> const &dx, *Scalar* scale)

Template Function mir::geometry::generic::operator*(DeltaY<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *DeltaY*<*T*> `mir::geometry::generic::`**`operator*`**(*DeltaY*<*T*> const &dy, *Scalar* scale)

Template Function mir::geometry::generic::operator*(Scalar, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::`**`operator*`**(*Scalar* scale, *Displacement*<*T*> const &disp)

Template Function mir::geometry::generic::operator*(Displacement<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::`**`operator*`**(*Displacement*<*T*> const &disp, *Scalar* scale)

Template Function mir::geometry::generic::operator*(Scalar, Size<T> const&)

- Defined in file_include_core_mir_geometry_size.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Size*<*T*> `mir::geometry::generic::`**`operator*`**(*Scalar* scale, *Size*<*T*> const &size)

Template Function mir::geometry::generic::operator*(Size<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_size.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Size*<*T*> `mir::geometry::generic::`**operator***(*Size*<*T*> const &size, *Scalar* scale)

Template Function mir::geometry::generic::operator+(DeltaX<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaX*<*T*> `mir::geometry::generic::`**operator+**(*DeltaX*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+(DeltaY<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaY*<*T*> `mir::geometry::generic::`**operator+**(*DeltaY*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator+(X<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *X*<*T*> `mir::geometry::generic::`**operator+**(*X*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+(Y<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Y*<*T*> `mir::geometry::generic::`**operator+**(*Y*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator+(Width<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Width*<*T*> `mir::geometry::generic::`**operator+**(*Width*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+(Height<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Height*<*T*> `mir::geometry::generic::`**operator+**(*Height*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator+(Width<T>, Width<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Width*<*T*> `mir::geometry::generic::`**operator+**(*Width*<*T*> lhs, *Width*<*T*> rhs)

Template Function mir::geometry::generic::operator+(Height<T>, Height<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Height*<*T*> `mir::geometry::generic::`**operator+**(*Height*<*T*> lhs, *Height*<*T*> rhs)

Template Function mir::geometry::generic::operator+(Displacement<T> const&, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::`**operator+**(*Displacement*<*T*> const &lhs,
Displacement<*T*> const &rhs)

Template Function mir::geometry::generic::operator+(Point<T> const&, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::`**operator+**(*Point*<*T*> const &lhs, *Displacement*<*T*>
const &rhs)

Template Function mir::geometry::generic::operator+(Displacement<T> const&, Point<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::`**operator+**(*Displacement*<*T*> const &lhs, *Point*<*T*>
const &rhs)

Template Function mir::geometry::generic::operator+(Point<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::`**operator+**(*Point*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+(Point<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::`**operator+**(*Point*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(DeltaX<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *DeltaX*<*T*> &`mir::geometry::generic::`**operator+=**(*DeltaX*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(DeltaY<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *DeltaY*<*T*> &`mir::geometry::generic::`**operator+=**(*DeltaY*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(X<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *X*<*T*> &mir::geometry::generic::**operator+=**(*X*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(Y<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Y*<*T*> &mir::geometry::generic::**operator+=**(*Y*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(Width<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Width*<*T*> &mir::geometry::generic::**operator+=**(*Width*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(Height<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Height*<*T*> &mir::geometry::generic::**operator+=**(*Height*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(Width<T>&, Width<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Width*<*T*> &**mir::geometry::generic::operator+=**(*Width*<*T*> &lhs, *Width*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(Height<T>&, Height<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Height*<*T*> &**mir::geometry::generic::operator+=**(*Height*<*T*> &lhs, *Height*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(Point<T>&, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> &**mir::geometry::generic::operator+=**(*Point*<*T*> &lhs, *Displacement*<*T*> const &rhs)

Template Function mir::geometry::generic::operator+=(Point<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline *Point*<*T*> &**mir::geometry::generic::operator+=**(*Point*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator+=(Point<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline *Point*<*T*> &**mir::geometry::generic::operator+=**(*Point*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-(DeltaX<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaX*<*T*> **mir::geometry::generic::operator-**(*DeltaX*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-(DeltaY<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaY*<*T*> **mir::geometry::generic::operator-**(*DeltaY*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-(DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaX*<*T*> **mir::geometry::generic::operator-**(*DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-(DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaY*<*T*> `mir::geometry::generic::`**operator-**(*DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-(X<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *X*<*T*> `mir::geometry::generic::`**operator-**(*X*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Y<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Y*<*T*> `mir::geometry::generic::`**operator-**(*Y*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Width<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Width*<*T*> `mir::geometry::generic::`**operator-**(*Width*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Height<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *Height*<*T*> `mir::geometry::generic::`**`operator-`**(*Height*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-(X<T>, X<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaX*<*T*> `mir::geometry::generic::`**`operator-`**(*X*<*T*> lhs, *X*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Y<T>, Y<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaY*<*T*> `mir::geometry::generic::`**`operator-`**(*Y*<*T*> lhs, *Y*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Width<T>, Width<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaX*<*T*> `mir::geometry::generic::`**`operator-`**(*Width*<*T*> lhs, *Width*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Height<T>, Height<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline constexpr *DeltaY*<*T*> `mir::geometry::generic::`**`operator-`**(*Height*<*T*> lhs, *Height*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Displacement<T> const&, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::`**`operator-`**(*Displacement*<*T*> const &lhs,
Displacement<*T*> const &rhs)

Template Function mir::geometry::generic::operator-(Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::`**`operator-`**(*Displacement*<*T*> const &rhs)

Template Function mir::geometry::generic::operator-(Point<T> const&, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::`**`operator-`**(*Point*<*T*> const &lhs, *Displacement*<*T*>
const &rhs)

Template Function mir::geometry::generic::operator-(Point<T> const&, Point<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Displacement*<*T*> `mir::geometry::generic::operator-`(*Point*<*T*> const &lhs, *Point*<*T*> const &rhs)

Template Function mir::geometry::generic::operator-(Point<T>, DeltaX<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::operator-`(*Point*<*T*> lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-(Point<T>, DeltaY<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> `mir::geometry::generic::operator-`(*Point*<*T*> lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(DeltaX<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *DeltaX*<*T*> &`mir::geometry::generic::operator-=`(*DeltaX*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(DeltaY<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *DeltaY*<*T*> &mir::geometry::generic::**operator-=**(*DeltaY*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(X<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *X*<*T*> &mir::geometry::generic::**operator-=**(*X*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(Y<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Y*<*T*> &mir::geometry::generic::**operator-=**(*Y*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(Width<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Width*<*T*> &mir::geometry::generic::**operator-=**(*Width*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(Height<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**>
inline *Height*<*T*> &**mir::geometry::generic::operator-=**(*Height*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(Point<T>&, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline constexpr *Point*<*T*> &**mir::geometry::generic::operator-=**(*Point*<*T*> &lhs, *Displacement*<*T*> const &rhs)

Template Function mir::geometry::generic::operator-=(Point<T>&, DeltaX<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline *Point*<*T*> &**mir::geometry::generic::operator-=**(*Point*<*T*> &lhs, *DeltaX*<*T*> rhs)

Template Function mir::geometry::generic::operator-=(Point<T>&, DeltaY<T>)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
inline *Point*<*T*> &**mir::geometry::generic::operator-=**(*Point*<*T*> &lhs, *DeltaY*<*T*> rhs)

Template Function mir::geometry::generic::operator/(Width<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Width*<*T*> `mir::geometry::generic::`**operator/**(*Width*<*T*> const &w, *Scalar* scale)

Template Function mir::geometry::generic::operator/(Height<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Height*<*T*> `mir::geometry::generic::`**operator/**(*Height*<*T*> const &h, *Scalar* scale)

Template Function mir::geometry::generic::operator/(DeltaX<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *DeltaX*<*T*> `mir::geometry::generic::`**operator/**(*DeltaX*<*T*> const &dx, *Scalar* scale)

Template Function mir::geometry::generic::operator/(DeltaY<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *DeltaY*<*T*> `mir::geometry::generic::`**operator/**(*DeltaY*<*T*> const &dy, *Scalar* scale)

Template Function mir::geometry::generic::operator/(X<T> const&, Width<U> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **U**>
inline constexpr auto `mir::geometry::generic::operator/`(*X*<*T*> const &x, *Width*<*U*> const &width)

Template Function mir::geometry::generic::operator/(Y<T> const&, Height<U> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **U**>
inline constexpr auto `mir::geometry::generic::operator/`(*Y*<*T*> const &y, *Height*<*U*> const &height)

Template Function mir::geometry::generic::operator/(Size<T> const&, Scalar)

- Defined in file_include_core_mir_geometry_size.h

Function Documentation

template<typename **T**, typename **Scalar**>
inline constexpr *Size*<*T*> `mir::geometry::generic::operator/`(*Size*<*T*> const &size, *Scalar* scale)

Template Function mir::geometry::generic::operator<

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
inline bool `mir::geometry::generic::operator<`(*Displacement*<*T*> const &lhs, *Displacement*<*T*> const &rhs)

Template Function mir::geometry::generic::operator<<(std::ostream&, Value<T, Tag> const&)

- Defined in file_include_core_mir_geometry_dimensions.h

Function Documentation

template<typename **T**, typename **Tag**>
std::ostream &**mir::geometry::generic::operator<<**(std::ostream &out, *Value*<*T*, *Tag*> const &value)

Template Function mir::geometry::generic::operator<<(std::ostream&, Displacement<T> const&)

- Defined in file_include_core_mir_geometry_displacement.h

Function Documentation

template<typename **T**>
std::ostream &**mir::geometry::generic::operator<<**(std::ostream &out, *Displacement*<*T*> const &value)

Template Function mir::geometry::generic::operator<<(std::ostream&, Point<T> const&)

- Defined in file_include_core_mir_geometry_point.h

Function Documentation

template<typename **T**>
std::ostream &**mir::geometry::generic::operator<<**(std::ostream &out, *Point*<*T*> const &value)

Template Function mir::geometry::generic::operator<<(std::ostream&, Rectangle<T> const&)

- Defined in file_include_core_mir_geometry_rectangle.h

Function Documentation

template<typename **T**>
std::ostream &**mir::geometry::generic::operator<<**(std::ostream &out, *Rectangle*<*T*> const &value)

Template Function mir::geometry::generic::operator<<(std::ostream&, Size<T> const&)

- Defined in file_include_core_mir_geometry_size.h

Function Documentation

template<typename **T**>
std::ostream &**mir::geometry::generic::operator<<**(std::ostream &out, *Size*<*T*> const &value)

Function mir::geometry::operator<<

- Defined in file_include_core_mir_geometry_rectangles.h

Function Documentation

std::ostream &**mir::geometry::operator<<**(std::ostream &out, *Rectangles* const &value)

Function mir::mir_depth_layer_get_index

- Defined in file_include_core_mir_depth_layer.h

Function Documentation

auto **mir::mir_depth_layer_get_index**(*MirDepthLayer* depth_layer) -> unsigned int

Returns the height of a MirDepthLayer.

As the name implies, the returned value is usable as an array index (0 is returned for the bottommost layer and there are no gaps between layers). The values returned for each layer are in no way stable across Mir versions, and are only meaningful relative to each other.

Template Function mir::operator!=(optional_value<T> const&, optional_value<T> const&)

- Defined in file_include_core_mir_optional_value.h

Function Documentation

template<typename **T**>
inline bool **mir::operator!=**(*optional_value*<*T*> const &lhs, *optional_value*<*T*> const &rhs)

Template Function mir::operator!=(optional_value<T> const&, T const&)

- Defined in file_include_core_mir_optional_value.h

Function Documentation

template<typename **T**>
inline bool **mir::operator!=**(*optional_value*<*T*> const &lhs, *T* const &rhs)

Template Function mir::operator!=(T const&, optional_value<T> const&)

- Defined in file_include_core_mir_optional_value.h

Function Documentation

template<typename **T**>
inline bool **mir::operator!=**(*T* const &lhs, *optional_value*<*T*> const &rhs)

Template Function mir::operator<<

- Defined in file_include_core_mir_int_wrapper.h

Function Documentation

template<typename **Tag**, typename **ValueType**>
std::ostream &**mir::operator<<**(std::ostream &out, *IntWrapper*<*Tag*, *ValueType*> const &value)

Template Function mir::operator==(optional_value<T> const&, optional_value<T> const&)

- Defined in file_include_core_mir_optional_value.h

Function Documentation

template<typename **T**>
inline bool **mir::operator==**(*optional_value*<*T*> const &lhs, *optional_value*<*T*> const &rhs)

Template Function mir::operator==(optional_value<T> const&, T const&)

- Defined in file_include_core_mir_optional_value.h

Function Documentation

template<typename **T**>
inline bool `mir::operator==`(*optional_value*<*T*> const &lhs, *T* const &rhs)

Template Function mir::operator==(T const&, optional_value<T> const&)

- Defined in file_include_core_mir_optional_value.h

Function Documentation

template<typename **T**>
inline bool `mir::operator==`(*T* const &lhs, *optional_value*<*T*> const &rhs)

Function mir_eglapp_init

- Defined in file_examples_miral-shell_spinner_eglapp.h

Function Documentation

std::vector<std::shared_ptr<*MirEglSurface*>> **mir_eglapp_init**(struct wl_display *display)

Function mir_surface_init

- Defined in file_examples_miral-shell_spinner_miregl.h

Function Documentation

std::vector<std::shared_ptr<*MirEglSurface*>> **mir_surface_init**(std::shared_ptr<MirEglApp> const &app)

Template Function miral::add_window_manager_policy

- Defined in file_include_miral_miral_window_management_options.h

Function Documentation

template<typename **Policy**, typename ...**Args**>
inline auto `miral::add_window_manager_policy`(std::string const &name, *Args*&... args) ->
WindowManagerOption

Function miral::application_for(wl_client *)

- Defined in file_include_miral_miral_wayland_extensions.h

Function Documentation

auto `miral::application_for`(wl_client *client) -> *Application*

Get the MirAL application for a wl_client.

> **Remark**
>
> Since MirAL 2.5

Returns

The application (null if no application is found)

Function miral::application_for(wl_resource *)

- Defined in file_include_miral_miral_wayland_extensions.h

Function Documentation

auto `miral::application_for`(wl_resource *resource) -> *Application*

Get the MirAL application for a wl_resource.

> **Remark**
>
> Since MirAL 2.5

Returns

The application (null if no application is found)

Function miral::apply_lifecycle_state_to

- Defined in file_include_miral_miral_application.h

Function Documentation

void `miral::apply_lifecycle_state_to`(*Application* const &application, *MirLifecycleState* state)

Function miral::display_configuration_options

- Defined in file_include_miral_miral_display_configuration_option.h

Function Documentation

void `miral::display_configuration_options`(mir::Server &server)

Enable the display configuration options.

- display-config {clone,sidebyside,single,static=<filename>}
- translucent {on,off}

Function miral::equivalent_display_area

- Defined in file_include_miral_miral_output.h

Function Documentation

auto `miral::equivalent_display_area`(*Output* const &lhs, *Output* const &rhs) -> bool

Function miral::kill

- Defined in file_include_miral_miral_application.h

Function Documentation

void `miral::kill`(*Application* const &application, int sig)

Template Function miral::lambda_as_function

- Defined in file_include_miral_miral_lambda_as_function.h

Function Documentation

template<typename **Lambda**>
auto `miral::lambda_as_function`(*Lambda* &&lambda) -> typename detail::*FunctionType*<decltype(&std::remove_reference<*Lambda*>::type::operator**()**)>::type

Function miral::name_of

- Defined in file_include_miral_miral_application.h

Function Documentation

auto `miral::name_of`(*Application* const &application) -> std::string

Function miral::operator!=(Output::PhysicalSizeMM const&, Output::PhysicalSizeMM const&)

- Defined in file_include_miral_miral_output.h

Function Documentation

inline bool `miral::operator!=`(*Output*::*PhysicalSizeMM* const &lhs, *Output*::*PhysicalSizeMM* const &rhs)

Function miral::operator!=(Window const&, Window const&)

- Defined in file_include_miral_miral_window.h

Function Documentation

inline bool `miral::operator!=`(*Window* const &lhs, *Window* const &rhs)

Function miral::operator!=(std::shared_ptr<mir::scene::Surface> const&, Window const&)

- Defined in file_include_miral_miral_window.h

Function Documentation

inline bool `miral::operator!=`(std::shared_ptr<mir::scene::Surface> const &lhs, *Window* const &rhs)

Function miral::operator!=(Window const&, std::shared_ptr<mir::scene::Surface> const&)

- Defined in file_include_miral_miral_window.h

Function Documentation

inline bool `miral::operator!=`(*Window* const &lhs, std::shared_ptr<mir::scene::Surface> const &rhs)

Function miral::operator<

- Defined in file_include_miral_miral_window.h

Function Documentation

bool `miral::operator<`(*Window* const &lhs, *Window* const &rhs)

Function miral::operator<=

- Defined in file_include_miral_miral_window.h

Function Documentation

inline bool `miral::operator<=`(*Window* const &lhs, *Window* const &rhs)

Function miral::operator==(Output::PhysicalSizeMM const&, Output::PhysicalSizeMM const&)

- Defined in file_include_miral_miral_output.h

Function Documentation

bool `miral::operator==`(*Output*::*PhysicalSizeMM* const &lhs, *Output*::*PhysicalSizeMM* const &rhs)

Function miral::operator==(Window const&, Window const&)

- Defined in file_include_miral_miral_window.h

Function Documentation

bool `miral::operator==`(*Window* const &lhs, *Window* const &rhs)

Function miral::operator==(std::shared_ptr<mir::scene::Surface> const&, Window const&)

- Defined in file_include_miral_miral_window.h

Function Documentation

bool `miral::operator==`(std::shared_ptr<mir::scene::Surface> const &lhs, *Window* const &rhs)

Function miral::operator==(Window const&, std::shared_ptr<mir::scene::Surface> const&)

- Defined in file_include_miral_miral_window.h

Function Documentation

bool `miral::operator==`(*Window* const &lhs, std::shared_ptr<mir::scene::Surface> const &rhs)

Function miral::operator>

- Defined in file_include_miral_miral_window.h

Function Documentation

inline bool `miral::operator>`(*Window* const &lhs, *Window* const &rhs)

Function miral::operator>=

- Defined in file_include_miral_miral_window.h

Function Documentation

inline bool `miral::operator>=`(*Window* const &lhs, *Window* const &rhs)

Function miral::pid_of

- Defined in file_include_miral_miral_application.h

Function Documentation

auto `miral::pid_of`(*Application* const &application) -> pid_t

Function miral::pre_init

- Defined in file_include_miral_miral_configuration_option.h

Function Documentation

auto `miral::pre_init`(*ConfigurationOption* const &clo) -> *ConfigurationOption*

Update the option to be called back *before* Mir initialization starts.

Parameters

`clo` – the option

Function miral::PrintTo

- Defined in file_include_miral_miral_window.h

Function Documentation

void `miral::PrintTo`(*Window* const &bar, std::ostream *os)

Customization for Google test (to print surface name in errors)

See also

https://github.com/google/googletest/blob/main/docs/advanced.md#teaching-googletest-how-to-print-your-values

Remark

Since MirAL 3.3

Template Function miral::set_window_management_policy

- Defined in file_include_miral_miral_set_window_management_policy.h

Function Documentation

template<typename **Policy**, typename ...**Args**>
auto `miral::`**`set_window_management_policy`**`(`*Args*`&... args) -> `*SetWindowManagementPolicy*

Function miral::socket_fd_of

- Defined in file_include_miral_miral_application.h

Function Documentation

auto `miral::`**`socket_fd_of`**`(`*Application* const &application`) -> int`

Returns the file descriptor of the client's socket connection, or -1 if there is no client socket. May be used for authentication with apparmor. X11 apps always return -1, since they do not connect directly to the Mir process.

Remark

Since MirAL 3.4

Function miral::toolkit::mir_event_get_input_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEvent const *`miral::toolkit::`**`mir_event_get_input_event`**`(`*MirEvent* const *event`)`

Retrieve the MirInputEvent associated with a MirEvent of type mir_event_type_input.

See <mir_toolkit/events/input/input_event.h> for accessors.

Parameters

event – **[in]** The event

Returns

The associated MirInputEvent

Function miral::toolkit::mir_event_get_type

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirEventType miral::toolkit::**mir_event_get_type**(*MirEvent* const *event)

Retrieves the type of a MirEvent.

Now preferred over direct access to ev->type. In particular ev->type will never be mir_event_type_input and mir_event_get_type is the only way to ensure mir_event_get_input_event will succeed.

Parameters

event – **[in]** The event

Returns

The event type

Function miral::toolkit::mir_input_event_get_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirEvent const *miral::toolkit::**mir_input_event_get_event**(MirInputEvent const *event)

Retrieve the MirEvent associated with a given input event.

Parameters

event – **[in]** The input event

Returns

The MirEvent

Function miral::toolkit::mir_input_event_get_event_time

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

int64_t miral::toolkit::**mir_input_event_get_event_time**(MirInputEvent const *event)

**

Retrieve the time at which an input event occurred.

Parameters

event – **[in]** The input event

Returns

A timestamp in nanoseconds-since-epoch

Function miral::toolkit::mir_input_event_get_keyboard_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirKeyboardEvent const *miral::toolkit::**mir_input_event_get_keyboard_event**(MirInputEvent const *event)

Retrieve the MirKeyboardEvent associated with a given input event.

Parameters

event – **[in]** The input event

Returns

The MirKeyboardEvent or NULL if event type is not mir_input_event_type_key

Function miral::toolkit::mir_input_event_get_pointer_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirPointerEvent const *miral::toolkit::**mir_input_event_get_pointer_event**(MirInputEvent const *event)

Retrieve the MirPointerEvent associated with a given input event.

Parameters

event – **[in]** The input event

Returns

The MirPointerEvent or NULL if event type is not mir_input_event_type_pointer

Function miral::toolkit::mir_input_event_get_touch_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirTouchEvent const *miral::toolkit::**mir_input_event_get_touch_event**(MirInputEvent const *event)

Retrieve the MirTouchEvent associated with a given input event.

Parameters

event – **[in]** The input event

Returns

The MirTouchEvent or NULL if event type is not mir_input_event_type_touch

Function miral::toolkit::mir_input_event_get_type

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEventType `miral::toolkit::`**`mir_input_event_get_type`**(MirInputEvent const *event)

Retrieve the type of an input event.

E.g. key, touch...

Parameters

event – **[in]** The input event

Returns

The input event type

Function miral::toolkit::mir_keyboard_event_action

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirKeyboardAction `miral::toolkit::`**`mir_keyboard_event_action`**(MirKeyboardEvent const *event)

Retrieve the action which triggered a given key event.

Parameters

event – **[in]** The key event

Returns

The associated action

Function miral::toolkit::mir_keyboard_event_input_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEvent const *`miral::toolkit::`**`mir_keyboard_event_input_event`**(MirKeyboardEvent const *event)

Retrieve the corresponding input event.

Parameters

event – **[in]** The keyboard event

Returns

The input event

Function miral::toolkit::mir_keyboard_event_key_text

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

char const *`miral::toolkit::`**`mir_keyboard_event_key_text`**(MirKeyboardEvent const *event)

Retrieve the text the key press would emit as null terminated utf8 string.

The text will only be available to key down and key repeat events. For mir_keyboard_action_up or key presses that do produce text an empty string will be returned.

Parameters

`event` – **[in]** The key event

Returns

The text

Function miral::toolkit::mir_keyboard_event_keysym

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

xkb_keysym_t `miral::toolkit::`**`mir_keyboard_event_keysym`**(MirKeyboardEvent const *event)

Retrieve the xkb mapped keysym associated with the key acted on.

. May be interpreted as per <xkbcommon/xkbcommon-keysyms.h>

Remark

Since MirAL 3.3

Parameters

`event` – **[in]** The key event

Returns

The xkb_keysym

Function miral::toolkit::mir_keyboard_event_modifiers

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEventModifiers miral::toolkit::**mir_keyboard_event_modifiers**(MirKeyboardEvent const *event)

Retrieve the modifier keys pressed when the key action occured.

Parameters

event – **[in]** The key event

Returns

The modifier mask

Function miral::toolkit::mir_keyboard_event_scan_code

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

int miral::toolkit::**mir_keyboard_event_scan_code**(MirKeyboardEvent const *event)

Retrieve the raw hardware scan code associated with the key acted on.

May be interpreted as per <linux/input.h>

Parameters

event – **[in]** The key event

Returns

The scancode

Function miral::toolkit::mir_pointer_event_action

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirPointerAction miral::toolkit::**mir_pointer_event_action**(MirPointerEvent const *event)

Retrieve the action which occured to generate a given pointer event.

Parameters

event – **[in]** The pointer event

Returns

Action performed by the pointer

Function miral::toolkit::mir_pointer_event_axis_value

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

float miral::toolkit::**mir_pointer_event_axis_value**(MirPointerEvent const *event, *MirPointerAxis* axis)

Retrieve the axis value reported by a given pointer event.

Parameters

- **event** – **[in]** The pointer event
- **axis** – **[in]** The axis to retreive a value from

Returns

The value of the given axis

Function miral::toolkit::mir_pointer_event_button_state

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

bool miral::toolkit::**mir_pointer_event_button_state**(MirPointerEvent const *event, *MirPointerButton* button)

Retrieve the state of a given pointer button when the action occurred.

Parameters

- **event** – **[in]** The pointer event
- **button** – **[in]** The button to check

Returns

Whether the given button is depressed

Function miral::toolkit::mir_pointer_event_buttons

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirPointerButtons miral::toolkit::**mir_pointer_event_buttons**(MirPointerEvent const *event)

Retreive the pointer button state as a masked set of values.

Parameters

event – **[in]** The pointer event

Returns

The button state

Function miral::toolkit::mir_pointer_event_input_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEvent const *miral::toolkit::**mir_pointer_event_input_event**(MirPointerEvent const *event)

Retrieve the corresponding input event.

Parameters

event – **[in]** The pointer event

Returns

The input event

Function miral::toolkit::mir_pointer_event_modifiers

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEventModifiers miral::toolkit::**mir_pointer_event_modifiers**(MirPointerEvent const *event)

Retrieve the modifier keys pressed when the pointer action occured.

Parameters

event – **[in]** The pointer event

Returns

The modifier mask

Function miral::toolkit::mir_touch_event_action

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirTouchAction miral::toolkit::**mir_touch_event_action**(MirTouchEvent const *event, unsigned int touch_index)

Retrieve the action which occured for a touch at given index.

Parameters

- **event** – **[in]** The touch event
- **touch_index** – **[in]** The touch index. Must be less than (touch_count - 1).

Returns

Action performed for the touch at index.

Function miral::toolkit::mir_touch_event_axis_value

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

float miral::toolkit::**mir_touch_event_axis_value**(MirTouchEvent const *event, unsigned int touch_index, *MirTouchAxis* axis)

Retrieve the axis value for a given axis on an indexed touch.

Parameters

- **event** – **[in]** The touch event
- **touch_index** – **[in]** The touch index. Must be less than (touch_count - 1).
- **axis** – **[in]** The axis to retreive a value from

Returns

The value of the given axis

Function miral::toolkit::mir_touch_event_id

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirTouchId miral::toolkit::**mir_touch_event_id**(MirTouchEvent const *event, unsigned int touch_index)

Retrieve the TouchID for a touch at given index.

Parameters

- **event** – **[in]** The touch event
- **touch_index** – **[in]** The touch index. Must be less than (touch_count - 1).

Returns

ID of the touch at index

Function miral::toolkit::mir_touch_event_input_event

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEvent const *miral::toolkit::**mir_touch_event_input_event**(MirTouchEvent const *event)

Retrieve the corresponding input event.

Parameters

event – **[in]** The touch event

Returns

The input event

Function miral::toolkit::mir_touch_event_modifiers

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirInputEventModifiers `miral::toolkit::`**`mir_touch_event_modifiers`**(MirTouchEvent const *event)

Retrieve the modifier keys pressed when the touch action occured.

Parameters

`event` – **[in]** The key event

Returns

The modifier mask

Function miral::toolkit::mir_touch_event_point_count

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

unsigned int `miral::toolkit::`**`mir_touch_event_point_count`**(MirTouchEvent const *event)

Retrieve the number of touches reported for a given touch event.

Each touch is said to be index in the event and may be accessed by index 0, 1, … , (touch_count - 1)

Parameters

`event` – **[in]** The touch event

Returns

The number of touches

Function miral::toolkit::mir_touch_event_tooltype

- Defined in file_include_miral_miral_toolkit_event.h

Function Documentation

MirTouchTooltype `miral::toolkit::`**`mir_touch_event_tooltype`**(MirTouchEvent const *event, unsigned int touch_index)

Retrieve the tooltype for touch at given index.

Parameters

- **`event`** – **[in]** The touch event
- **`touch_index`** – **[in]** The touch index. Must be less than (touch_count - 1).

Returns

Tooltype used for the touch at index

Function miral::window_for

- Defined in file_include_miral_miral_wayland_extensions.h

Function Documentation

auto `miral::window_for`(wl_resource *surface) -> *Window*

Get the MirAL *Window* for a Wayland Surface, XdgSurface, etc. Note that there may not be a corresponding *miral::Window* (e.g. the surface is created and assigned properties before 'commit' creates the *miral::Window*).

Remark

Since MirAL 2.5

Returns

The window (null if no window is found)

Function miroil::dispatch_input_event

- Defined in file_include_miroil_miroil_eventdispatch.h

Function Documentation

void `miroil::dispatch_input_event`(const miral::*Window* &window, const MirInputEvent *event)

Specialized Template Function std::swap

- Defined in file_include_miral_miral_window_info.h

Function Documentation

template<>
inline void `std::swap`(miral::*WindowInfo* &lhs, miral::*WindowInfo* &rhs)

Function wallpaper::font_file(std::string const&)

- Defined in file_examples_example-server-lib_wallpaper_config.h

Function Documentation

void `wallpaper::font_file`(std::string const &font_file)

Function wallpaper::font_file()

- Defined in file_examples_example-server-lib_wallpaper_config.h

Function Documentation

auto `wallpaper::font_file`() -> std::string

Variables

Variable mir::fatal_error

- Defined in file_include_core_mir_fatal.h

Variable Documentation

void (*`mir::fatal_error`)(char const *reason, ...)

fatal_error() is strictly for "this should never happen" situations that you cannot recover from.

By default it points at *fatal_error_abort()*. Note the reason parameter is a simple char* so its value is clearly visible in stack trace output.

Remark

There is no attempt to make this thread-safe, if it needs to be changed that should be done before spinning up the Mir server.

Param reason
[in] A printf-style format string.

Variable mir::geometry::generic::is_exactly_representable

- Defined in file_include_core_mir_geometry_size.h

Variable Documentation

```
template<typename T,
typename U> concept mir::geometry::generic::is_exactly_representable  = requires{typename std::enable_i
U>::type;requires std::floating_point<T>;}
```

Variable mir_eglapp_background_opacity

- Defined in file_examples_miral-shell_spinner_eglapp.h

Variable Documentation

float **mir_eglapp_background_opacity**

Defines

Define __has_extension

- Defined in file_include_core_mir_toolkit_common.h

Define Documentation

__has_extension

Define __has_feature

- Defined in file_include_core_mir_toolkit_common.h

Define Documentation

__has_feature(x)

Define MIR_BYTES_PER_PIXEL

- Defined in file_include_core_mir_toolkit_common.h

Define Documentation

MIR_BYTES_PER_PIXEL(f)

Define MIR_VERSION_NUMBER

- Defined in file_include_core_mir_toolkit_mir_version_number.h

Define Documentation

MIR_VERSION_NUMBER(major, minor, micro)

MIR_VERSION_NUMBER.

Returns the combined version information as a single 32-bit value for logical comparisons. For example: #if MIR_CLIENT_VERSION >= *MIR_VERSION_NUMBER(2,3,4)*

This can be useful to conditionally build code depending on new features or specific bugfixes in the Mir client library.

Parameters

- **major** – [in] The major version (eg: 3 for version 3.2.33)
- **minor** – [in] The minor version (eg: 2 for version 3.2.33)
- **micro** – [in] The micro version (eg: 33 for version 3.2.33)

Define MIRAL_MAJOR_VERSION

- Defined in file_include_miral_miral_version.h

Define Documentation

MIRAL_MAJOR_VERSION

MIRAL_MAJOR_VERSION.

The major MirAL API version. This will increase once per API incompatible release.

See also: http://semver.org/

Define MIRAL_MICRO_VERSION

- Defined in file_include_miral_miral_version.h

Define Documentation

MIRAL_MICRO_VERSION

MIRAL_MICRO_VERSION.

The micro miral API version. This will increment each release. This is usually uninteresting for server code, but may be of use in selecting whether to use a feature that has previously been buggy.

This corresponds to the PATCH field of http://semver.org/

Define MIRAL_MINOR_VERSION

- Defined in file_include_miral_miral_version.h

Define Documentation

MIRAL_MINOR_VERSION

MIRAL_MINOR_VERSION.

The minor MirAL API version. This will increase each time new backwards-compatible API is added, and will reset to 0 each time MIRAL_MAJOR_VERSION is incremented.

See also: http://semver.org/

Define MIRAL_VERSION

- Defined in file_include_miral_miral_version.h

Define Documentation

MIRAL_VERSION

MIRAL_VERSION.

The current version of the MirAL headers in use.

Typedefs

Typedef mir::EventUPtr

- Defined in file_include_miroil_miroil_event_builder.h

Typedef Documentation

typedef std::unique_ptr<*MirEvent*, void (*)(*MirEvent**)> `mir::EventUPtr`

Typedef mir::geometry::DeltaX

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::DeltaX` = generic::*DeltaX*<int>

Typedef mir::geometry::DeltaXF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::DeltaXF` = generic::*DeltaX*<float>

Typedef mir::geometry::DeltaY

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::DeltaY` = generic::*DeltaY*<int>

Typedef mir::geometry::DeltaYF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::DeltaYF` = generic::*DeltaY*<float>

Typedef mir::geometry::Displacement

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Displacement` = generic::*Displacement*<int>

Typedef mir::geometry::DisplacementD

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::DisplacementD` = generic::*Displacement*<double>

Typedef mir::geometry::DisplacementF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::DisplacementF` = generic::*Displacement*<float>

Typedef mir::geometry::generic::DeltaX

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

template<typename **T**>

using `mir::geometry::generic::DeltaX` = *Value*<*T*, *DeltaXTag*>

Typedef mir::geometry::generic::DeltaY

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

template<typename **T**>

using `mir::geometry::generic::`**`DeltaY`** = *Value*<*T*, *DeltaYTag*>

Typedef mir::geometry::generic::Height

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

template<typename **T**>

using `mir::geometry::generic::`**`Height`** = *Value*<*T*, *HeightTag*>

Typedef mir::geometry::generic::Width

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

template<typename **T**>

using `mir::geometry::generic::`**`Width`** = *Value*<*T*, *WidthTag*>

Typedef mir::geometry::generic::X

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

template<typename **T**>

using `mir::geometry::generic::`**`X`** = *Value*<*T*, *XTag*>

Typedef mir::geometry::generic::Y

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

template<typename **T**>

using `mir::geometry::generic::Y` = *Value*<*T*, *YTag*>

Typedef mir::geometry::Height

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Height` = generic::*Height*<int>

Typedef mir::geometry::HeightF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::HeightF` = generic::*Height*<float>

Typedef mir::geometry::Point

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Point` = generic::*Point*<int>

Typedef mir::geometry::PointD

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::PointD` = generic::*Point*<double>

Typedef mir::geometry::PointF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::PointF` = generic::*Point*<float>

Typedef mir::geometry::Rectangle

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Rectangle` = generic::*Rectangle*<int>

Typedef mir::geometry::RectangleD

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::RectangleD` = generic::*Rectangle*<double>

Typedef mir::geometry::RectangleF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::RectangleF` = generic::*Rectangle*<float>

Typedef mir::geometry::Size

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Size` = generic::*Size*<int>

Typedef mir::geometry::SizeD

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::SizeD` = generic::*Size*<double>

Typedef mir::geometry::SizeF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::SizeF` = generic::*Size*<float>

Typedef mir::geometry::Stride

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Stride` = generic::*Value*<int, *StrideTag*>

Typedef mir::geometry::Width

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Width` = generic::*Width*<int>

Typedef mir::geometry::WidthF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::WidthF` = generic::*Width*<float>

Typedef mir::geometry::X

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::X` = generic::*X*<int>

Typedef mir::geometry::XF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::XF` = generic::*X*<float>

Typedef mir::geometry::Y

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::Y` = generic::*Y*<int>

Typedef mir::geometry::YF

- Defined in file_include_core_mir_geometry_forward.h

Typedef Documentation

using `mir::geometry::YF` = generic::*Y*<float>

Typedef miral::Application

- Defined in file_include_miral_miral_application.h

Typedef Documentation

using `miral::Application` = std::shared_ptr<mir::scene::Session>

Typedef miral::BufferStreamId

- Defined in file_include_miral_miral_window_specification.h

Typedef Documentation

typedef mir::*IntWrapper*<detail::SessionsBufferStreamIdTag> `miral::BufferStreamId`

Typedef miral::CommandLineOption

- Defined in file_include_miral_miral_command_line_option.h

Typedef Documentation

using `miral::CommandLineOption` = *ConfigurationOption*

Typedef miral::WindowManagementPolicyBuilder

- Defined in file_include_miral_miral_window_management_options.h

Typedef Documentation

using `miral::WindowManagementPolicyBuilder` = std::function<std::unique_ptr<miral::*WindowManagementPolicy*>(*WindowManagerTools* const &tools)>

Typedef MirBufferPackage

- Defined in file_include_core_mir_toolkit_mir_native_buffer.h

Typedef Documentation

typedef struct *MirBufferPackage* **MirBufferPackage**

Typedef MirClientFdCallback

- Defined in file_include_miroil_miroil_mir_prompt_session.h

Typedef Documentation

typedef void (***MirClientFdCallback**)(*MirPromptSession* *prompt_session, size_t count, int const *fds, void *context)

Typedef MirDepthLayer

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirDepthLayer* **MirDepthLayer**

Depth layer controls Z ordering of surfaces.

A surface will always appear on top of surfaces with a lower depth layer, and below those with a higher one. A depth layer can be converted to a number with *mir::mir_depth_layer_get_index()*. This is useful for creating a list indexed by depth layer, or comparing the height of two layers.

Typedef MirEdgeAttachment

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirEdgeAttachment* **MirEdgeAttachment**

Typedef MirEvent

- Defined in file_include_miral_miral_append_event_filter.h

Typedef Documentation

typedef struct *MirEvent* **MirEvent**

Typedef MirEvent

- Defined in file_include_miral_miral_prepend_event_filter.h

Typedef Documentation

typedef struct *MirEvent* **MirEvent**

Typedef MirFocusMode

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirFocusMode* **MirFocusMode**

Focus mode controls how a surface gains and loses focus.

Typedef MirFormFactor

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirFormFactor* **MirFormFactor**

Form factor associated with a physical output.

Typedef MirInputDeviceId

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef int64_t **MirInputDeviceId**

Typedef MirInputEventModifiers

- Defined in file_include_core_mir_toolkit_events_enums.h

Typedef Documentation

typedef unsigned int **MirInputEventModifiers**

Typedef MirLifecycleState

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirLifecycleState* **MirLifecycleState**

Typedef MirMirrorMode

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirMirrorMode* **MirMirrorMode**

Mirroring axis relative to the "natural" orientation of the display.

Typedef MirNativeBuffer

- Defined in file_include_core_mir_toolkit_mir_native_buffer.h

Typedef Documentation

typedef struct *MirBufferPackage* **MirNativeBuffer**

Typedef miroil::CompositorID

- Defined in file_include_miroil_miroil_surface.h

Typedef Documentation

using `miroil::CompositorID` = void const*

Typedef miroil::CreateNamedCursor

- Defined in file_include_miroil_miroil_mir_server_hooks.h

Typedef Documentation

using `miroil::CreateNamedCursor` = std::function<std::shared_ptr<mir::graphics::CursorImage>(std::string const &name)>

Typedef miroil::OutputId

- Defined in file_include_miroil_miroil_display_id.h

Typedef Documentation

using `miroil::OutputId` = mir::*IntWrapper*<mir::graphics::detail::GraphicsConfOutputIdTag>

Typedef MirOrientation

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirOrientation* `MirOrientation`

Direction relative to the "natural" orientation of the display.

Typedef MirOrientationMode

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirOrientationMode* `MirOrientationMode`

Typedef MirOutputGammaSupported

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirOutputGammaSupported* `MirOutputGammaSupported`

Supports gamma correction.

Typedef MirOutputType

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirOutputType* **MirOutputType**

Typedef MirPixelFormat

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirPixelFormat* **MirPixelFormat**

32-bit pixel formats (8888): The order of components in the enum matches the order of the components as they would be written in an integer representing a pixel value of that format.

For example; abgr_8888 should be coded as 0xAABBGGRR, which will end up as R,G,B,A in memory on a little endian system, and as A,B,G,R on a big endian system.

24-bit pixel formats (888): These are in literal byte order, regardless of CPU architecture it's always the same. Writing these 3-byte pixels is typically slower than other formats but uses less memory than 32-bit and is endian-independent.

16-bit pixel formats (565/5551/4444): Always interpreted as one 16-bit integer per pixel with components in high-to-low bit order following the format name. These are the fastest formats, however colour quality is visibly lower.

Typedef MirPlacementGravity

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirPlacementGravity* **MirPlacementGravity**

Reference point for aligning a surface relative to a rectangle.

Each element (surface and rectangle) has a MirPlacementGravity assigned.

Typedef MirPlacementHints

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirPlacementHints* **MirPlacementHints**

Positioning hints for aligning a window relative to a rectangle.

These hints determine how the window should be positioned in the case that the surface would fall off-screen if placed in its ideal position.

For example, `mir_placement_hints_flip_x` will invert the x component of `aux_rect_placement_offset` and replace `mir_placement_gravity_northwest` with `mir_placement_gravity_northeast` and vice versa if the window extends beyond the left or right edges of the monitor.

If `mir_placement_hints_slide_x` is set, the window can be shifted horizontally to fit on-screen.

If `mir_placement_hints_resize_x` is set, the window can be shrunken horizontally to fit.

If `mir_placement_hints_antipodes` is set then the rect gravity may be substituted with the opposite corner (e.g. `mir_placement_gravity_northeast` to `mir_placement_gravity_southwest`) in combination with other options.

When multiple flags are set, flipping should take precedence over sliding, which should take precedence over resizing.

Typedef MirPointerAcceleration

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef enum *MirPointerAcceleration* **MirPointerAcceleration**

MirPointerAcceleration describes the way pointer movement is filtered:

- mir_pointer_acceleration_none: (acceleration bias + 1.0) is applied as a factor to the current velocity of the pointer. So a bias of 0 to results to no change of velocity.
- mir_pointer_acceleration_adaptive: acceleration bias selects an acceleration function based on the current velocity that usually consists of two linear inclines separated by a plateau.

Typedef MirPointerButtons

- Defined in file_include_core_mir_toolkit_events_enums.h

Typedef Documentation

typedef unsigned int **MirPointerButtons**

Typedef MirPointerConfinementState

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirPointerConfinementState* **MirPointerConfinementState**
Pointer Confinement.

Typedef MirPointerHandedness

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef enum *MirPointerHandedness* **MirPointerHandedness**

Typedef MirPowerMode

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirPowerMode* **MirPowerMode**

Typedef MirPromptSession

- Defined in file_include_miroil_miroil_mir_prompt_session.h

Typedef Documentation

typedef struct *MirPromptSession* **MirPromptSession**

Typedef MirPromptSessionState

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirPromptSessionState* **MirPromptSessionState**

Typedef MirResizeEdge

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirResizeEdge* **MirResizeEdge**

Hints for resizing a window.

These values are used to indicate which edge(s) of a surface is being dragged in a resize operation.

Typedef MirShellChrome

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirShellChrome* **MirShellChrome**

Shell chrome.

Typedef MirSubpixelArrangement

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirSubpixelArrangement* **MirSubpixelArrangement**

Physical arrangement of subpixels on the physical output.

This is always relative to the "natural" orientation of the display - mir_orientation_normal.

Typedef MirTouchId

- Defined in file_include_miral_miral_toolkit_event.h

Typedef Documentation

typedef int32_t **MirTouchId**

An identifier for a touch-point.

TouchId's are unique per-gesture. That is to say, once a touch has gone down at time T, no other touch will use that touch's ID until all touches at time T have come up.

Typedef MirTouchpadClickMode

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef enum *MirTouchpadClickMode* **MirTouchpadClickMode**

MirTouchpadClickMode configures how the touchpad itself should generate pointer button events.

The available click modes may be active simultaneously.

- mir_touchpad_click_mode_none: no active click mode
- mir_touchpad_click_mode_area_to_click: simulate pointer buttons using click areas on the touchpad
- mir_touchpad_click_mode_finger_count: simulate pointer buttons using the number of fingers down

Typedef MirTouchpadClickModes

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef unsigned int **MirTouchpadClickModes**

Typedef MirTouchpadScrollMode

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef enum *MirTouchpadScrollMode* **MirTouchpadScrollMode**

MirTouchpadScrollMode configures how the touchpad should generate scroll events.

- mir_touchpad_scroll_mode_none: no scroll
- mir_touchpad_scroll_mode_two_finger_scroll: two finger movement generates generates vertical and horizontal scroll events
- mir_touchpad_scroll_mode_edge_scroll: touch movement at the edge of the touchpad genertes scroll events
- mir_touchpad_scroll_mode_button_down_scroll: movement on the touchpad generates scroll events when a button is held down simultaneously

Typedef MirTouchpadScrollModes

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef unsigned int **MirTouchpadScrollModes**

Typedef MirTouchscreenMappingMode

- Defined in file_include_core_mir_toolkit_mir_input_device_types.h

Typedef Documentation

typedef enum *MirTouchscreenMappingMode* **MirTouchscreenMappingMode**

Mapping modes for touchscreen devices.

The mode defines how coordinates from the touchscreen frequently referred to as device coordinates are translated into scene coordinates.

This configuration mode is relevant for different classes of input devices, i.e handheld devices with builtin touchscreens or external graphic tablets or external monitors with touchscreen capabilities.

Typedef MirWindowAttrib

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirWindowAttrib* **MirWindowAttrib**

Attributes of a window that the client and server/shell may wish to get or set over the wire.

Typedef MirWindowFocusState

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirWindowFocusState* **MirWindowFocusState**

Typedef MirWindowState

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirWindowState* **MirWindowState**

Typedef MirWindowType

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirWindowType* **MirWindowType**

Typedef MirWindowVisibility

- Defined in file_include_core_mir_toolkit_common.h

Typedef Documentation

typedef enum *MirWindowVisibility* **MirWindowVisibility**

2.4.2 Introducing the Miral API

The main() program

The main() program from miral-shell looks like this:

```
/*
 * Copyright © Canonical Ltd.
 *
 * This program is free software: you can redistribute it and/or modify
 * under the terms of the GNU General Public License version 2 or 3 as
 * published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include "tiling_window_manager.h"
#include "floating_window_manager.h"
#include "wallpaper_config.h"
#include "spinner/splash.h"

#include <miral/display_configuration_option.h>
#include <miral/external_client.h>
#include <miral/runner.h>
#include <miral/window_management_options.h>
```

(continues on next page)

(continued from previous page)

```
#include <miral/append_event_filter.h>
#include <miral/internal_client.h>
#include <miral/command_line_option.h>
#include <miral/cursor_theme.h>
#include <miral/decorations.h>
#include <miral/keymap.h>
#include <miral/toolkit_event.h>
#include <miral/x11_support.h>
#include <miral/wayland_extensions.h>

#include <xkbcommon/xkbcommon-keysyms.h>

#include <cstring>

namespace
{
struct ConfigureDecorations
{
    miral::Decorations const decorations{[]
        {
            if (auto const strategy = getenv("MIRAL_SHELL_DECORATIONS"))
            {
                if (strcmp(strategy, "always-ssd") == 0) return␣
↪miral::Decorations::always_ssd();
                if (strcmp(strategy, "prefer-ssd") == 0) return␣
↪miral::Decorations::prefer_ssd();
                if (strcmp(strategy, "always-csd") == 0) return␣
↪miral::Decorations::always_csd();
            }
            return miral::Decorations::prefer_csd();
        }()};

    void operator()(mir::Server& s) const
    {
        decorations(s);
    }
};
}

int main(int argc, char const* argv[])
{
    using namespace miral;
    using namespace miral::toolkit;

    std::function<void()> shutdown_hook{[]{}};

    SpinnerSplash spinner;
    InternalClientLauncher launcher;
    WindowManagerOptions window_managers
        {
            add_window_manager_policy<FloatingWindowManagerPolicy>("floating", spinner,␣
↪launcher, shutdown_hook),
```

(continues on next page)

(continued from previous page)

```
            add_window_manager_policy<TilingWindowManagerPolicy>("tiling", spinner,␣
↪launcher),
        };

    MirRunner runner{argc, argv};

    runner.add_stop_callback([&] { shutdown_hook(); });

    ExternalClientLauncher external_client_launcher;

    std::string terminal_cmd{"miral-terminal"};

    auto const quit_on_ctrl_alt_bksp = [&](MirEvent const* event)
        {
            if (mir_event_get_type(event) != mir_event_type_input)
                return false;

            MirInputEvent const* input_event = mir_event_get_input_event(event);
            if (mir_input_event_get_type(input_event) != mir_input_event_type_key)
                return false;

            MirKeyboardEvent const* kev = mir_input_event_get_keyboard_event(input_
↪event);
            if (mir_keyboard_event_action(kev) != mir_keyboard_action_down)
                return false;

            MirInputEventModifiers mods = mir_keyboard_event_modifiers(kev);
            if (!(mods & mir_input_event_modifier_alt) || !(mods & mir_input_event_
↪modifier_ctrl))
                return false;

            switch (mir_keyboard_event_keysym(kev))
            {
            case XKB_KEY_BackSpace:
                runner.stop();
                return true;

            case XKB_KEY_t:
            case XKB_KEY_T:
                external_client_launcher.launch({terminal_cmd});
                return false;

            case XKB_KEY_x:
            case XKB_KEY_X:
                external_client_launcher.launch_using_x11({"xterm"});
                return false;

            default:
                return false;
            };
        };
```

(continues on next page)

(continued from previous page)

```
    Keymap config_keymap;

    auto run_startup_apps = [&](std::string const& apps)
    {
      for (auto i = begin(apps); i != end(apps); )
      {
          auto const j = find(i, end(apps), ':');
          external_client_launcher.launch(std::vector<std::string>{std::string{i, j}});
          if ((i = j) != end(apps)) ++i;
      }
    };


    return runner.run_with(
        {
            CursorTheme{"default:DMZ-White"},
            WaylandExtensions{},
            X11Support{},
            ConfigureDecorations{},
            window_managers,
            display_configuration_options,
            external_client_launcher,
            launcher,
            config_keymap,
            AppendEventFilter{quit_on_ctrl_alt_bksp},
            StartupInternalClient{spinner},
            ConfigurationOption{run_startup_apps, "startup-apps", "Colon separated list␣
↪of startup apps", ""},
            pre_init(ConfigurationOption{[&](std::string const& typeface) {␣
↪::wallpaper::font_file(typeface); },
                                         "shell-wallpaper-font", "font file to use for␣
↪wallpaper", ::wallpaper::font_file()}),
            ConfigurationOption{[&](std::string const& cmd) { terminal_cmd = cmd; },
                                "shell-terminal-emulator", "terminal emulator to use",␣
↪terminal_cmd}
        });
}
```

This shell is providing *FloatingWindowManagerPolicy*, *TilingWindowManagerPolicy* and *SpinnerSplash*. The rest is from MirAL.

If you look for the corresponding code in lp:qtmir and lp:mir you'll find it less clear, more verbose and scattered over multiple files.

A shell has to provide a window management policy (miral-shell provides two: *FloatingWindowManagerPolicy* and *TilingWindowManagerPolicy*). A window management policy needs to implement the *miral::WindowManagementPolicy* interface for handling a set of window management events.

The way these events are handled determines the behaviour of the shell.

The *miral::WindowManagerTools* interface provides the principle methods for a window management policy to control Mir.

2.4.3 A brief guide for versioning symbols in the Mir DSOs

So, what do I have to do?

There are more detailed descriptions below, but as a general rule:

- If you add a new symbol, add it to a `*_NEXTSERIES` version stanza, like `MIR_CLIENT_0.22`, `MIR_PLATFORM_0.22`, etc representing the next future Mir series in which the new symbol will first be released.
- If you change the behaviour or signature of a symbol *and* wish to preserve backward compatibility, see "Change symbols without breaking ABI" below.

Can I have some details?

Sure.

Mir is a set of libraries, one C++ library for writing display- server/compositor/shells and one C library for writing clients (or, more usually, toolkits for clients) that use a Mir display-server for output. Mir also has internal dynamic libraries for platform support - drivers - and may in future allow the same with extensions to the core functionality. As such, the ABI of these interfaces is important to keep in mind.

Mir uses the ELF symbol versioning support. This provides three advantages:

- Consumers of the Mir libraries can know at load time rather than symbol resolution time whether the library exposes all the symbols they expect.
- We can drop or change the behaviour of symbols without breaking ABI by exposing multiple different implementations under different versions, and
- We can safely load multiple different versions of Mir libraries into the same process.

When should I bump SONAME?

There are varying standards for when to bump SONAME. In Mir we choose to bump the SONAME of a library whenever we make a change that could cause a binary linked to the library to fail *as long as* the binary is using only public interfaces and (where applicable) relying on documented behaviour. In general, changes that make an interface work as described by its documentation will not result in SONAME bumps.

With that explanation, you *should* bump SONAME when:

- You remove a public symbol from a library
- You change the signature of a public symbol *without* retaining the previous signature exposed under the old versioning.
- You change the behaviour of a public symbol *without* retaining the previous behaviour exposed with the old versioning.

If you are changing the behaviour of an interface, think about whether it's easy to maintain the old interface in parallel. If it is, you should consider providing both under different versions. This should become easier over time as the Mir ABI becomes more stable and also more valuable over time as the Mir libraries become more widely used.

Load-time version detection

When using versioned symbols the linker adds an extra, special symbol containing the version(s) exported from the library. Consumers of the library resolve this on library load. For example:

```
$ objdump -C -T lib/libmirclient.so
...
00000000002a2080  w   DO .data.rel.ro   0000000000000080  MIR_CLIENT_8 vtable for␣
↪mir::client::DefaultConnectionConfiguration
0000000000000000 g    DO *ABS*  0000000000000000  MIR_CLIENT_8 MIR_CLIENT_8
0000000000030ed2 g    DF .text  0000000000000098  MIR_CLIENT_8␣
↪mir::client::DefaultConnectionConfiguration::the_rpc_report()
...
```

This shows the special `MIR_CLIENT_8` symbol of the current libmirclient, along with a versioned symbol in the read-only data segment (the vtable for `mir::client::DefaultConnectionConfiguration`) and a versioned symbol in the text segment (the implementation of `mir::client::DefaultConnectionConfiguration::the_rpc_report()`). If a client needed a symbol versioned with `MIR_CLIENT_9`, it would try to resolve this at load time and fail, rather than failing when the symbol was first referenced - possibly much later, and more confusingly.

So what do I have to do to make this work?

When you add new symbols, add them to a new `version` block in the relevant `symbols.map` file, like so:

```
MIR_CLIENT_0.17 {
    global:
        mir_connect_sync;
        ...
        /* Other symbols go here */
};

MIR_CLIENT_0.18 {
    global:
        mir_connect_new_symbol;
    local:
        *;
} MIR_CLIENT_0.17;
```

Note that the script is read top to bottom; wildcards are greedily bound when first encountered, so to avoid surprises you should only have a wildcard in the final stanza.

Change symbols without breaking ABI

ELF DSOs can have multiple implementations for the same symbol with different versions. This means that you can change the signature or behaviour of a symbol without breaking dependants that use the old behaviour. While there can be as many different implementations with different versions as you want, there can only be one default implementation - this is what the linker will resolve to when building a dependant project.

Binding different implementations to the versioned symbol is done with `__asm__` directives in the relevant source file(s). The default implementation is specified with `symbol_name@@VERSION`; other versions are specified with `symbol_name@VERSION`.

Note that this does *not* require a change in SONAME. Binaries that have been linked against the old library will continue to work and resolve to the old implementation. Binaries linked against the new library will resolve to the new (default) implementation.

So, what do I have to do to make this work?

For example, if you wanted to change the signature of `mir_connection_create_surface` to take a new parameter:

`mir_connection_api.cpp`:

```
__asm__(".symver old_mir_connection_create_surface,mir_connection_create_surface@MIR_
↪CLIENT_0.17");

extern "C" MirWaitHandle* old_mir_connection_create_surface(...)
/* The old implementation */

/* The @@ specifies that this is the default version */
__asm__(".symver mir_connection_create_surface,mir_connection_create_surface@@@MIR_
↪CLIENT_0.18");
MirWaitHandle* mir_connection_create_surface(...)
/* The new implementation */
```

`symbols.map`:

```
MIR_CLIENT_0.17 {
    global:
        ...
        mir_connection_create_surface;
        ...
};

MIR_CLIENT_0.18 {
    global:
        ...
        mir_connection_create_surface;
        ...
    local:
        *;
} MIR_CLIENT_0.17;
```

Safely load multiple versions of a library into the same address space

This benefit is currently theoretical, as there seems to be a Protobuf singleton that aborts if we try this. But should that be resolved, it's theoretically possible and of some benefit...

This situation will come about - the Qtmir plugin links to libmirclient and also libEGL, and libEGL will link to libmirclient itself. There is no guarantee that Qtmir and libEGL will link to the same SONAME, and so a process can end up trying to load both `libmirclient.so.8` and `libmirclient.so.9` into its address space. Without symbol versioning this is potentially broken - there's no mechanism for libEGL to only resolve symbols from `libmirclient.so.8` and Qtmir to only resolve symbols from `libmirclient.so.9`, so in cases where symbols have changed use of those symbols will break.

By versioning the symbols we ensure that code always gets exactly the symbol implementation it expects, even when multiple library versions are loaded.

So, what do I have to do to make this work?

Ensure that different implementations of a symbol have different versions.

Additionally, there's the complication of passing objects between different versions. For the moment, we can not bother trying to make this work.

See also:

Binutils manual

Former glibc maintainer's DSO guide

2.4.4 Mir Continuous Integration

This document outlines the journey of a contribution to Mir through its continuous integration pipeline.

Overview

There are a number of components to this story, and a diagram might make for a good overview:

These are run at different stages in the pipeline, balancing the time it takes to run and the breadth of testing. We'll discuss those in more detail below.

Mir builds

When a pull request is opened, updated or merged into `main`, we validate that the contribution is correct by building the code and running our test suite. To facilitate this, we use Spread (or rather, our lightly patched version) to build Mir across a number of environments. `spread.yaml` holds environment definitions, while the actual build tasks are maintained under `spread/build`.

Our focus of development is the most recent Ubuntu LTS, and we maintain builds for any more recent, supported Ubuntu releases. We also build for stable releases of other Linux distributions that we have community interest on.

We also run builds using alternative toolchains (Clang) and development versions of Ubuntu and those other distributions.

We also maintain Mir branches for all previous Ubuntu LTS releases under support. These will only receive security updates and the occasional bug fix. These will always be `release/` branches on GitHub, and releases published on the GitHub release pages.

Opening a pull request, or merging into the `main` or `release/` branches, triggers a build on GitHub.

As part of the build, the following sets of tests are run:

Unit tests

Defined in `tests/unit-tests`, these assert the functionality of individual components in isolation, as is usual practice.

Acceptance tests

`tests/acceptance-tests` run more high-level tests, ensuring we maintain the contracts with external components as well as our own loadable modules' interfaces.

One major part of this is WLCS - asserting the behaviour of Mir's Wayland frontend is to specification of the protocol.

Integration tests

In `tests/integration-tests` lie other high level tests, ensuring we correctly integrate with some system components.

Performance and smoke tests

Lastly, `tests/performance-tests` holds a handful of performance-related tests, collecting some metrics about how the system performs end-to-end - and verifying that it works in the first place.

We also run these on a number of hardware platforms in our testing lab for every build of `main`.

Sanitizer runs

In addition to the above, for every merge to `main` we build and run the tests using the following sanitizers:

- **Undefined Behaviour** Building with UndefinedBehaviourSanitizer enabled reports no undefined behaviour, it would be a CI failure otherwise.
- **Address Sanitizer** As we have some fixing to do here, we run those builds - but don't enforce the problems reported.
- **Thread Sanitizer** Unfortunately due to incompatibility between GLib and TSan, we don't currently run the thread sanitizer, as that produces too many false positives.

ABI checks

For each pull request we check that the exported symbols are as expected

- and fail if the ABI changed in any way. It doesn't necessarily mean an ABI break - but new symbols need to be tracked.

Coverage measurement

We track test coverage for each pull request and `main` builds, and the results are visible on Codecov.io.

`.deb` package builds

Merges to `main`, `release/` branches as well as annotated tags are followed by `.deb` package builds in mir-team's Launchpad's Personal Package Archives (PPAs):

- `~mir-team/dev` for the latest development builds
- `~mir-team/rc` for release candidate and release builds

At release, those get copied to the `~mir-team/release`, general availability PPA.

Downstream snap builds

We build a subset of the downstream snaps on pull requests - these are then available in `edge/mir-pr<number>` Snap channels for testing, e.g.:

```
snap install miriway --channel edge/mir-pr1234
```

When `.deb` packages build in the PPAs above, all the affected downstream snaps get rebuilt through the `~mir-team` snap recipes on Launchpad and made available in the `edge` or `beta` channels, as appropriate.

End-to-end testing

When the snaps get published to the store, we have Jenkins dispatch test runs on a selection of hardware, ranging from Raspberry Pis through single-GPU systems all the way up to high performance multi-GPU (usually hybrid) ones. We maintain a matrix of test coverage in this spreadsheet.

These tests ultimately verify the full story end-to-end, installing the snaps on a range of Ubuntu versions (Core and classic alike) and verify behaviours and visuals through a number of scenarios.

It's a somewhat complex web of things happening, so again a diagram might help

checkbox-mir is the test orchestrator, collecting the different tests (smoke, performance and functional) and packaging them in a way that can be consumed by our lab setup. It's using our own mir-ci tests along with the mir-test-tools snap.

You can find more information about Checkbox itself in its documentation - it's a system used by our certification team, running thousands of tests on hundreds of systems every day.

The Jenkins job definitions go into a private repository, as they contain credentials. jenkins-job-builder is used to maintain the jobs, and at runtime, testflinger dispatches them to the device under test, while test-observer collects the results.

Summary

As shown above, Mir is being tested quite extensively on a large number of environments. We're extending the coverage every day, as well. There's a lot of moving pieces involved, but it does help us greatly in ensuring the quality level we hold ourselves to.

2.4.5 Linux Kernel Requirements for Mir

To run Mir with the default `gbm-kms` platform you need a linux kernel with at least:

Modules: i915, radeon and nouveau, to support the broadest range of common desktop/laptop hardware.

Version: Kernel version 3.11.0 or later, at least for radeon and nouveau to support fullscreen bypass correctly. Intel (i915) is known to work properly with even older kernels.

Additional patches: None at this time.

INDEX

Symbols

D

F

K

M

S

T

W