Mojo Core Embedder API

This document is a subset of the Mojo documentation.

Contents

Overview

The Mojo Core Embedder API enables process to initialize and use Mojo for IPC, using an implementation of Mojo Core that is statically linked into the application. See the note about dynamic linking here for more information about an alternative approach to Mojo Core initialization.

NOTE: Unless you are introducing a new binary entry point into the system (*e.g.,* a new executable with a new `main()` definition), you probably don't need to know anything about the Embedder API. Most processes defined in the Chrome repo today already fully initialize Mojo Core so that all other public Mojo APIs just work out of the box.

Basic Initialization

As an embedder, initializing Mojo Core requires a single call to `mojo::core::Init`:

```
#include "mojo/core/embedder/embedder.h"

int main(int argc, char** argv) {
  mojo::core::Init();

  // Now you can create message pipes, write messages, etc

  return 0;
}
```

This enables local API calls to work, so message pipes *etc* can be created and used. In some cases (particularly many unit testing scenarios) this is sufficient, but to support any actual multiprocess communication (e.g. sending or accepting Mojo invitations), a second IPC initialization step is required.

IPC Initialization

Internal Mojo IPC implementation requires a background `TaskRunner` on which it can watch for inbound I/O from other processes. This is configured using a `ScopedIPCSupport` object, which keeps IPC support alive through the extent of its lifetime.

Typically an application will create a dedicated background thread and give its `TaskRunner` to Mojo. Note that in Chromium, we use the existing "IO thread" in the browser process and content child processes. In general, any thread used for Mojo IPC support must be running a `base::MessageLoop::TYPE_IO` loop.

```
#include "base/threading/thread.h"
#include "mojo/core/embedder/embedder.h"
#include "mojo/core/embedder/scoped_ipc_support.h"

int main(int argc, char** argv) {
  mojo::core::Init();

  base::Thread ipc_thread("ipc!");
  ipc_thread.StartWithOptions(
      base::Thread::Options(base::MessageLoop::TYPE_IO, 0));

  // As long as this object is alive, all Mojo API surface relevant to IPC
  // connections is usable, and message pipes which span a process boundary will
  // continue to function.
  mojo::core::ScopedIPCSupport ipc_support(
      ipc_thread.task_runner(),
      mojo::core::ScopedIPCSupport::ShutdownPolicy::CLEAN);

  return 0;
}
```

This process is now fully prepared to use Mojo IPC!

Note that all existing process types in Chromium already perform this setup very early during startup.

Connecting Two Processes

Once IPC is initialized, you can bootstrap connections to other processes by using the public Invitations API.