Dart Beyond Flutter

Journeys in Embedding Dart

DATADOG

Jeff Ward

Senior Software Engineer, Mobile SDKs

DATADOG

What is Embedding Dart?

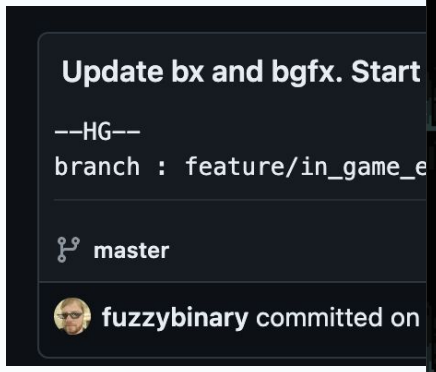
- Run Dart from an executable other than `dart` / `dart.exe`
- Best example is Flutter…

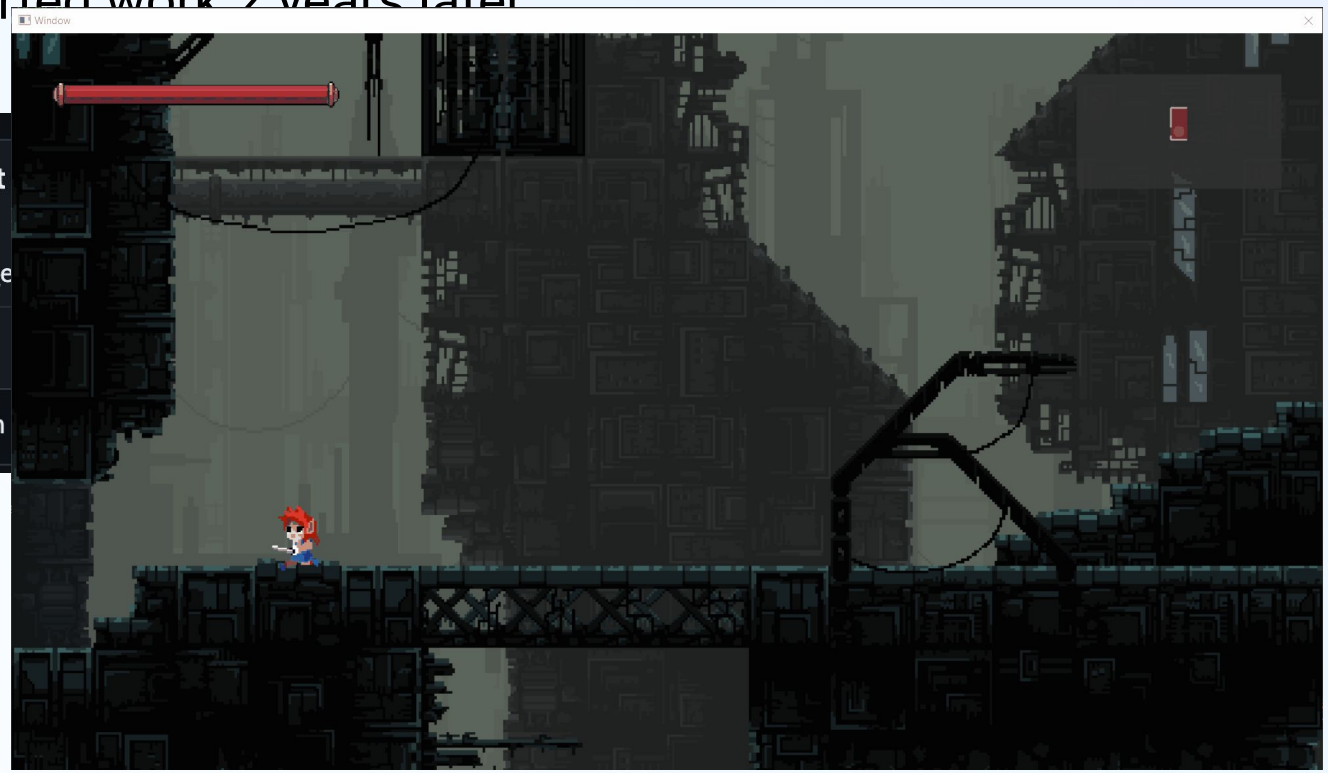
Why Do Dart Embedding?

- You like Dart and want to use it in other places
- Use Dart as a scripting language in a custom executable
  - Native Code for performance and system interactions.
  - Dart for everything else
- Lean more about Dart’s inner workings.

My Dart Embedding

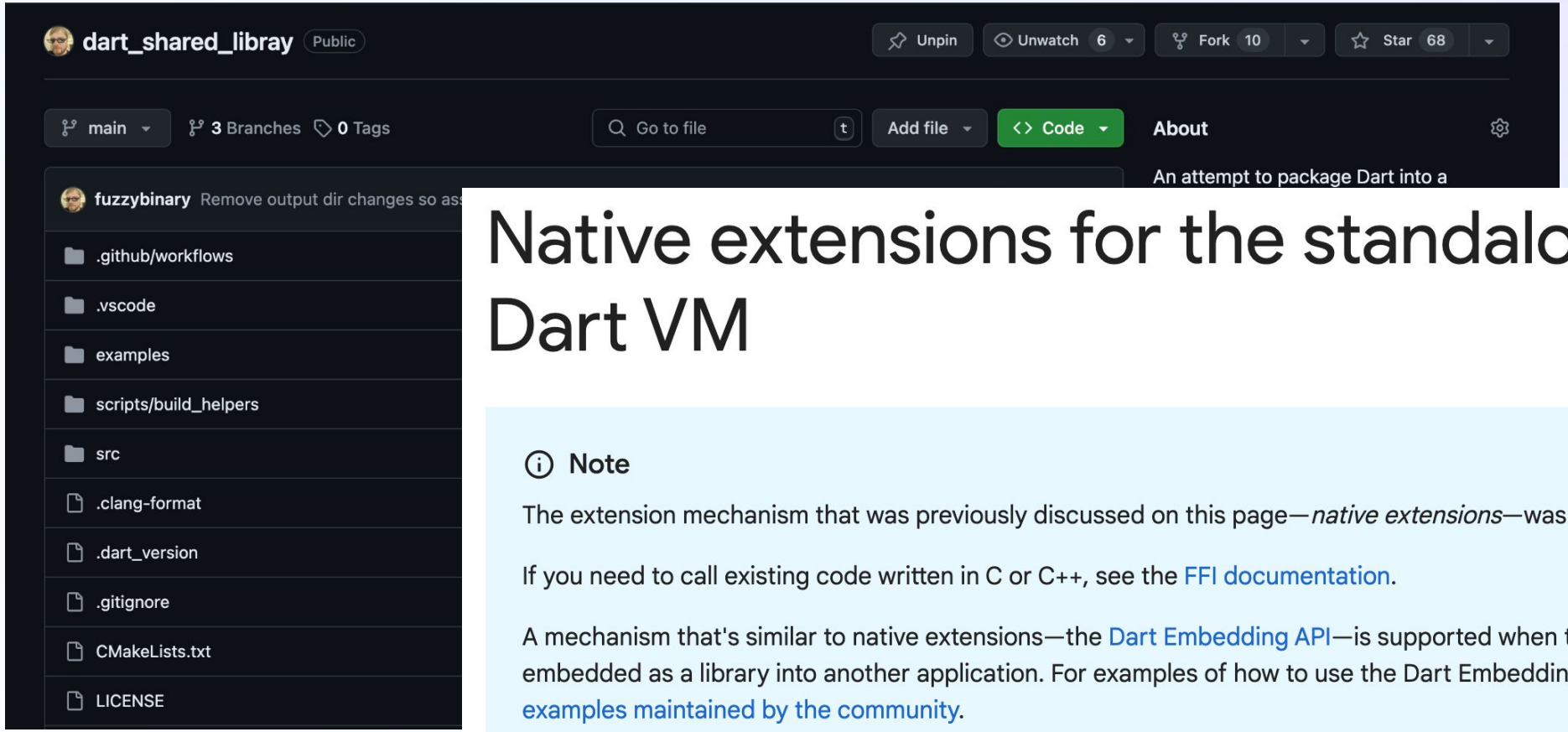
- I was working on a hobby game engine with C# scripting
- I liked Dart.
- 2015 - Eric Seidel's Sky Talk
- I wanted hot reload.
- Actually started work 2 years later





Current State

- https://github.com/fuzzybinary/dart_shared_library



Native extensions for the standalone Dart VM

Note

The extension mechanism that was previously discussed on this page—*native extensions*—was removed in Dart 2.15.

If you need to call existing code written in C or C++, see the FFI documentation.

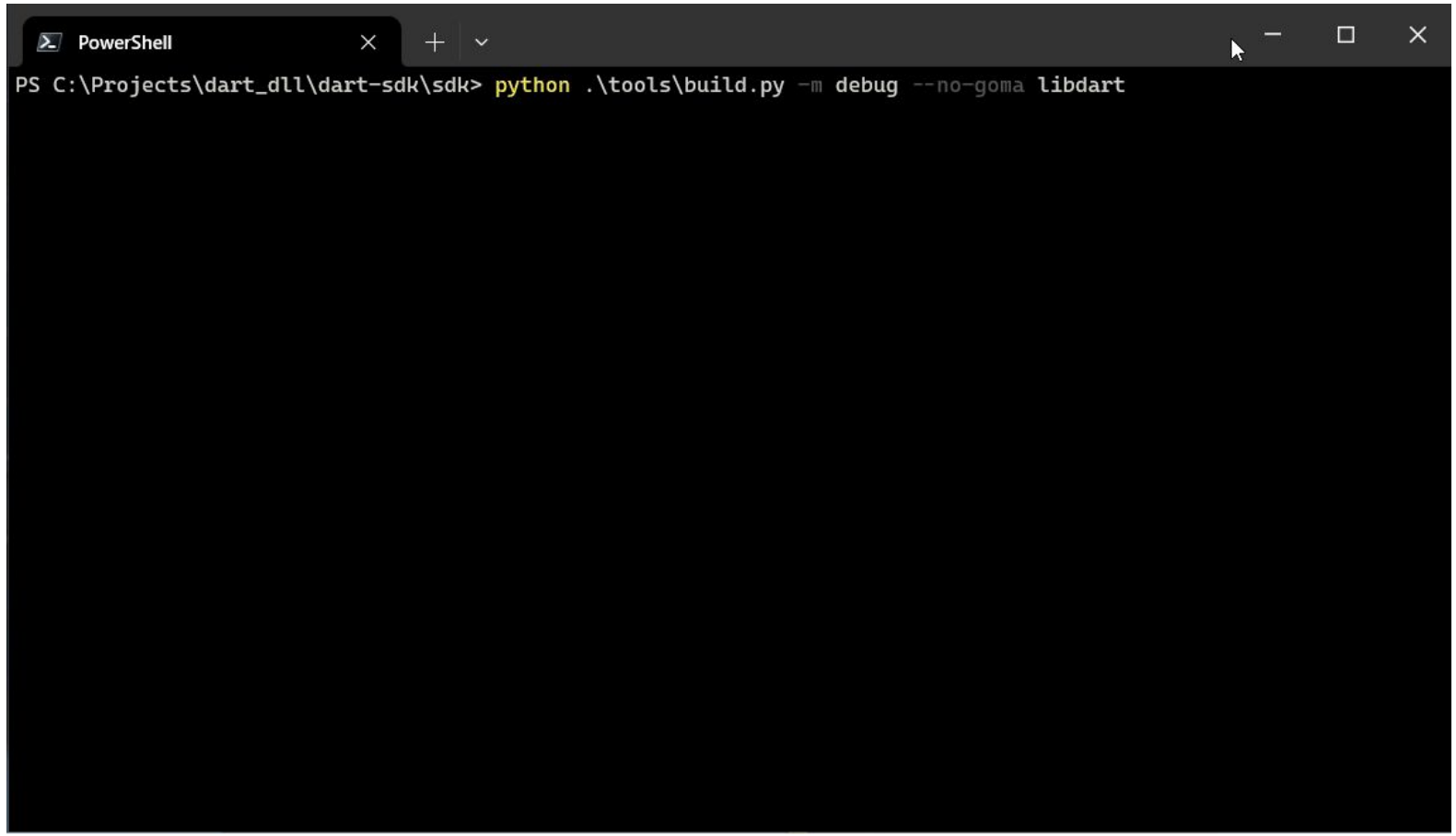
A mechanism that's similar to native extensions—the Dart Embedding API—is supported when the Dart VM is embedded as a library into another application. For examples of how to use the Dart Embedding API, see these examples maintained by the community.

Getting Started

The first step....

DATADOG

Building Dart



Building Dart (Briefly)

- Fetch Dart - Instructions from the Dart SDK
- Build Dart
- Modify build files
- Build `libdart`

This is all done through scripts on `dart_shared_library`

```diff
diff --git a/runtime/bin/BUILD.gn b/runtime/bin/BUILD.gn
index 91ebf8feb65..4ecf43807d4 100644
--- a/runtime/bin/BUILD.gn
+++ b/runtime/bin/BUILD.gn
@@ -1143,3 +1143,47 @@ if (defined(is_linux) && is_linux && defined(is_asan) && is_asan &&
     }
   }
 }
+
+static_library("libdart") {
+  deps = [
+    ":standalone_dart_io",
+    "..:libdart_jit",
+    "../platform:libdart_platform_jit",
+    ":dart_snapshot_cc",
+    ":dart_kernel_platform_cc",
+    "//third_party/boringssl",
+    "//third_party/zlib",
+  ]
+  if (dart_runtime_mode != "release") {
+    deps += [ "../observatory:standalone_observatory_archive" ]
+  }
+
+  complete_static_lib = true
+
+  include_dirs = [
+    "..",
+    "//third_party",
+  ]
+
+  sources = [
+    "builtin.cc",
+    "error_exit.cc",
+    "error_exit.h",
+    "vmservice_impl.cc",
+    "vmservice_impl.h",
+    "snapshot_utils.cc",
+    "snapshot_utils.h",
+    "gzip.cc",
+    "gzip.h",
+    "dartdev_isolate.cc",
+    "dartdev_isolate.h",
+    "dfe.cc",
+    "dfe.h",
+    "loader.cc",
+    "loader.h",
+    "dart_embedder_api_impl.cc",
+  ]
+  if (dart_runtime_mode == "release") {
+    sources += [ "observatory_assets_empty.cc" ]
+  }
+}
```

Steps

01 Initializing Dart

02 Running Main

03 Running Code on Demand

04 Async Code and Frame Maintenance

05 Memory Management

01 - Initializing Dart

Initializing Dart

- Call `Dart_Initialize`

… but first….

C++

```
Dart_SetVMFlags(0, nullptr);
dart::embedder::InitOnce();
dfe.Init();
dfe.set_use_dfe();
dfe.set_use_incremental_compiler(true);
```

Actually Initializing Dart

C++

```
Dart_InitializeParams params = {};
params.version = DART_INITIALIZE_PARAMS_CURRENT_VERSION;
params.vm_snapshot_data = kDartVmSnapshotData;
params.vm_snapshot_instructions = kDartVmSnapshotInstructions;
params.create_group = CreateIsolateGroupAndSetup;
params.initialize_isolate = OnIsolateInitialize;
params.shutdown_isolate = OnIsolateShutdown;
params.cleanup_isolate = DeleteIsolateData;
params.cleanup_group = DeleteIsolateGroupData;
params.entropy_source = DartUtils::EntropySource;
params.get_service_assets = GetVMServiceAssetsArchiveCallback;
params.start_kernel_isolate =
    dfe.UseDartFrontend() && dfe.CanUseDartFrontend();

Dart_Initialize(&params);
```

Create An Isolate

C++

```
static Dart_Isolate CreateIsolateGroupAndSetup(const char* script_uri,
                                               const char* main,
                                               const char* package_root,
                                               const char* package_config,
                                               Dart_IsolateFlags* flags,
                                               void* callback_data,
                                               char** error) {
  Dart_Isolate isolate = nullptr;

  if (0 == strcmp(script_uri, DART_KERNEL_ISOLATE_NAME)) {
    return CreateKernelIsolate(script_uri, main, package_root, package_config,
                               flags, callback_data, error);
  } else if (0 == strcmp(script_uri, DART_VM_SERVICE_ISOLATE_NAME)) {
    return CreateVmServiceIsolate(script_uri, main, package_root,
                                  package_config, flags, callback_data,
                                  _dart_dll_config.service_port, error);
  } else {
    return CreateIsolate(false, script_uri, main, package_config, flags,
                         callback_data, error);
  }

  return isolate;
}
```

Create My Isolate

```
153 > Dart_Isolate CreateIsolate(bool is_main_isolate, ...
263   }
```

- Get a platform kernel from the DFE.
- Create our Isolate from that kernel (`Dart_CreateIsolateGroupFromKernel`)
- Use `DartUtils` to initialize system libraries in our new isolate
- Use the DFE to compile and read our script.
- Load the script with `Dart_LoadScriptFromKernel`
- Exit the isolate and make in runnable with `Dart_MakeRunnable`

TL;DR

- Call `DartDll_Initialize` and `DartDll_LoadScript` from `dart_shared_library`

02 - Running Main

Running Main

C++

```cpp
Dart_Handle DartDll_RunMain(Dart_Handle library) {
  Dart_Handle mainClosure =
      Dart_GetField(library, Dart_NewStringFromCString("main"));
  if (!Dart_IsClosure(mainClosure)) {
    return mainClosure;
  }

  // Call _startIsolate in the isolate library to enable dispatching the
  // initial startup message.
  const intptr_t kNumIsolateArgs = 2;
  Dart_Handle isolateArgs[2] = {mainClosure, Dart_Null()};
  Dart_Handle isolateLib =
      Dart_LookupLibrary(Dart_NewStringFromCString("dart:isolate"));
  Dart_Handle result =
      Dart_Invoke(isolateLib, Dart_NewStringFromCString("_startMainIsolate"),
                  kNumIsolateArgs, isolateArgs);
  if (Dart_IsError(result)) {
    return result;
  }

  // Keep handling messages until the last active receive port is closed.
  result = Dart_RunLoop();

  return result;
}
```

Running Your Code

C++

```cpp
int main() {
  DartDll_Initialize();
  ...
  Dart_Isolate isolate = DartDll_LoadScript("hello_world.dart", nullptr);

  Dart_EnterIsolate(isolate);
  Dart_EnterScope();

  Dart_Handle library = Dart_RootLibrary();
  Dart_SetNativeResolver(library, ResolveNativeFunction, nullptr);

  DartDll_RunMain(library);
  ...
  Dart_ExitScope();
  Dart_ShutdownIsolate();

  // Don't forget to shutdown
  DartDll_Shutdown();

  return 0;
}
```

Dart

```dart
@pragma('vm:external-name','SimplePrint')
external void simplePrint(String s);

Run | Debug
void main() {
  simplePrint("Hello From Dart!\n");
}
```

Dart Native Resolver

C++

```cpp
Dart_NativeFunction ResolveNativeFunction(Dart_Handle name,
                                          int /* argc */,
                                          bool* /* auto_setup_scope */) {
  if (!Dart_IsString(name)) {
    return nullptr;
  }

  Dart_NativeFunction result = nullptr;

  const char* cname;
  HandleError(Dart_StringToCString(name, &cname));

  if (strcmp("SimplePrint", cname) == 0) {
    result = SimplePrint;
  }

  return result;
}
```

- The native resolver is passed a Dart String and an int number of arguments
- You provide a pointer to a function.

```cpp
void SimplePrint(Dart_NativeArguments arguments) {
  Dart_Handle string = HandleError(Dart_GetNativeArgument(arguments, 0));
  if (Dart_IsString(string)) {
    const char* cstring;
    Dart_StringToCString(string, &cstring);
    std::cout << "Hello from C++. Dart says:\n";
    std::cout << cstring;
  }
}
```

- The function takes one parameter, `Dart_NativeArguments` that has the arguments and is how you supply the return.

Demo Time

DATADOG

03 - Running Code On Demand

DATADOG

Running Code On Demand

Cute Framework

- `create_entity -> int`
- `get_drawable(int) -> Drawable*`

Cute will loop through all drawables, and draw them every frame.

C++

```cpp
class Drawable {
 public:
  int x;
  int y;
  int width;
  int height;

  Color color;
};
```

Cute Initialization

C++

```cpp
bool init_dart() {
  DartDllConfig config;
  DartDll_Initialize(config);

  //if package_config.json not exits run pub get
  _dart_isolate = DartDll_LoadScript("dart/main.dart",
                                     "dart/.dart_tool/package_config.json");
  if (_dart_isolate == nullptr) {
    return false;
  }

  Dart_EnterIsolate(_dart_isolate);
  Dart_SetMessageNotifyCallback(dart_message_notify_callback);

  Dart_EnterScope();
  Dart_Handle root_library = Dart_RootLibrary();
  _root_library = Dart_NewPersistentHandle(root_library);
  Dart_Handle init_function_name = Dart_NewStringFromCString("main");
  Dart_Handle result =
      Dart_Invoke(root_library, init_function_name, 0, nullptr);
  if (Dart_IsError(result)) {
    std::cout << Dart_GetError(result);
    Dart_ExitScope();
    return false;
  }

  Dart_ExitScope();

  return true;
}
```

Dart

```dart
List<Wall> walls = [];

// Dot!
int dotEntity = 0;
int dotX = 0;
int dotY = 0;
bool movingLeft = true;

Run | Debug
void main() {
  print('main');
  walls.add(
    Wall(-320, -240, 5, 480),
  );
  walls.add(
    Wall(315, -240, 5, 480),
  );
  walls.add(
    Wall(-320, -240, 640, 5),
  );
  walls.add(
    Wall(-320, 235, 640, 5),
  );

  dotEntity = ffi.createEntity(0, 0, 10, 10);
  final drawable = ffi.getDrawable(dotEntity);
  drawable.ref.color.g = 1.0;
}
```

Cute Running

C++

```cpp
void dart_frame(float delta_time) {
  Dart_EnterScope();

  Dart_Handle frame_function_name = Dart_NewStringFromCString("frame");
  Dart_Handle args[1] = {
      Dart_NewDouble(delta_time),
  };
  Dart_Handle root_library = Dart_HandleFromPersistent(_root_library);
  Dart_Handle result =
      Dart_Invoke(root_library, frame_function_name, 1, args);

  Dart_ExitScope();
}
```

Dart

```dart
void frame(double dt) {
  if (movingLeft) {
    dotX -= 3;
    if (dotX < -200) {
      dotX = -200;
      movingLeft = false;
    }
  } else {
    dotX += 3;
    if (dotX > 200) {
      dotX = 200;
      movingLeft = true;
    }
  }

  final dotDrawable = ffi.getDrawable(dotEntity);
  dotDrawable.ref.x = dotX;
  dotDrawable.ref.y = dotY;
}
```

Demo Time

DATADOG

04 - Async Code

Async Code For Games

Dart

```
await gamePanel.delayed(Duration(seconds: 2, milliseconds: 50
await gamePanel.say(systems, 'opening_imIn',
    location: TalkingBoxLocation.Top, portrait: ginoPortrait);
await gamePanel.say(systems, 'opening_howsItLook',
    location: TalkingBoxLocation.Top, portrait: handlerPortrait);
await gamePanel.say(systems, 'opening_old',
    location: TalkingBoxLocation.Top, portrait: ginoPortrait);
await gamePanel.say(systems, 'opening_findWhatWeCan',
    location: TalkingBoxLocation.Top, portrait: handlerPortrait);
await gamePanel.say(systems, 'opening_energySignature',
    location: TalkingBoxLocation.Top, portrait: handlerPortrait);
await gamePanel.say(systems, 'opening_hunting',
    location: TalkingBoxLocation.Top, portrait: ginoPortrait);
```

Async Code In Dart

- What happens if you `await` in a method?
- What do we call to resume that code?

C++

```cpp
Dart_Handle DartDll_DrainMicrotaskQueue() {
  Dart_EnterScope();

  Dart_Handle libraryName = Dart_NewStringFromCString("dart:isolate");
  Dart_Handle isolateLib = Dart_LookupLibrary(libraryName);
  if (Dart_IsError(isolateLib)) {
    std::cerr << "Error looking up 'dart:isolate' library: "
              << Dart_GetError(isolateLib);
    Dart_ExitScope();
    return isolateLib;
  }

  Dart_Handle invokeName =
      Dart_NewStringFromCString("_runPendingImmediateCallback");
  Dart_Handle result = Dart_Invoke(isolateLib, invokeName, 0, nullptr);
  if (Dart_IsError(result)) {
    std::cerr << "Error draining microtask queue: " << Dart_GetError(result);
    return result;
  }
  result = Dart_HandleMessage();
  if (Dart_IsError(result)) {
    std::cerr << "Error draining microtask queue: %s" << Dart_GetError(result);
    return result;
  }

  Dart_ExitScope();

  return result;
}
```

05 - Memory Management

Memory Management

- Native Code Holding Dart Objects
- Dart Holding Memory
- Dart Wrapping Native Objects

Native Holding Dart - Dart_PersistentHandle

- `Dart_Handles` are only valid between `Dart_EnterScope` / `Dart_ExitScope`
- Two flavors - `Dart_PersistentHandle` and `Dart_WeakPersistentHandle`
- Both created with `Dart_New*PersistentHandle`
- Both deleted with `Dart_Delete*PersistentHandle`
- Get objects using `Dart_HandleFrom*Persistent`

- Weak handles are allocated with “peer” objects (void*), and require a `Dart_HandleFinalizer`
  - The handle finalizer is invoked “sometime after the object is garbage collected, unless the handle has been deleted.”

Dart Holding Memory - FFI Pointers & Finalizers

- Dart can allocate your native structures - `malloc` / `calloc` in `dart:ffi`
- You need to cleanup that memory
  - Implement `Finalizable`
- Two types of Finalizers:
  - `Finalizer`
    - Get a callback when Dart finalizes the object to perform cleanup
    - Good for when Dart controls the object and allocated the memory
  - `NativeFinalizer`
    - Takes a C method
    - Does not give you back the Dart object
    - Good for when C controls the object and allocated the memory

> Compared to the Finalizer from `dart:core`, which makes no promises to ever call an attached callback, this native finalizer promises that all attached finalizers are definitely called at least once before the isolate group shuts down, and the callbacks are called as soon as possible after an object is recognized as inaccessible.

Dart Wrapping Objects - `NativeFieldWrapperClassX`

- Not documented in the Dart SDK (that I can find)
- Constructed with X pointers
- Created with `Dart_AllocateWithNativeFields`, which doesn't call a constructor
- Retrieve native fields with:
  - `Dart_GetNativeFieldsOfArgument`
  - `Dart_GetNativeReciever`

- Great when C controls the whole object lifecycle, and you just want to pass around a typed wrapper.
- Can't be used with FFI (so far as I know)

Making Sure Finalizers Get Called

- The Dart Message Queue

```
Dart_EnterIsolate(_dart_isolate);
Dart_SetMessageNotifyCallback(dart_message_notify_callback);
```

```
static unsigned int _dart_pending_messages = 0;

void dart_message_notify_callback(Dart_Isolate isolate) {
  _dart_pending_messages++;
}
```

```
while (_dart_pending_messages > 0) {

  Dart_HandleMessage();
  _dart_pending_messages--;
}
```

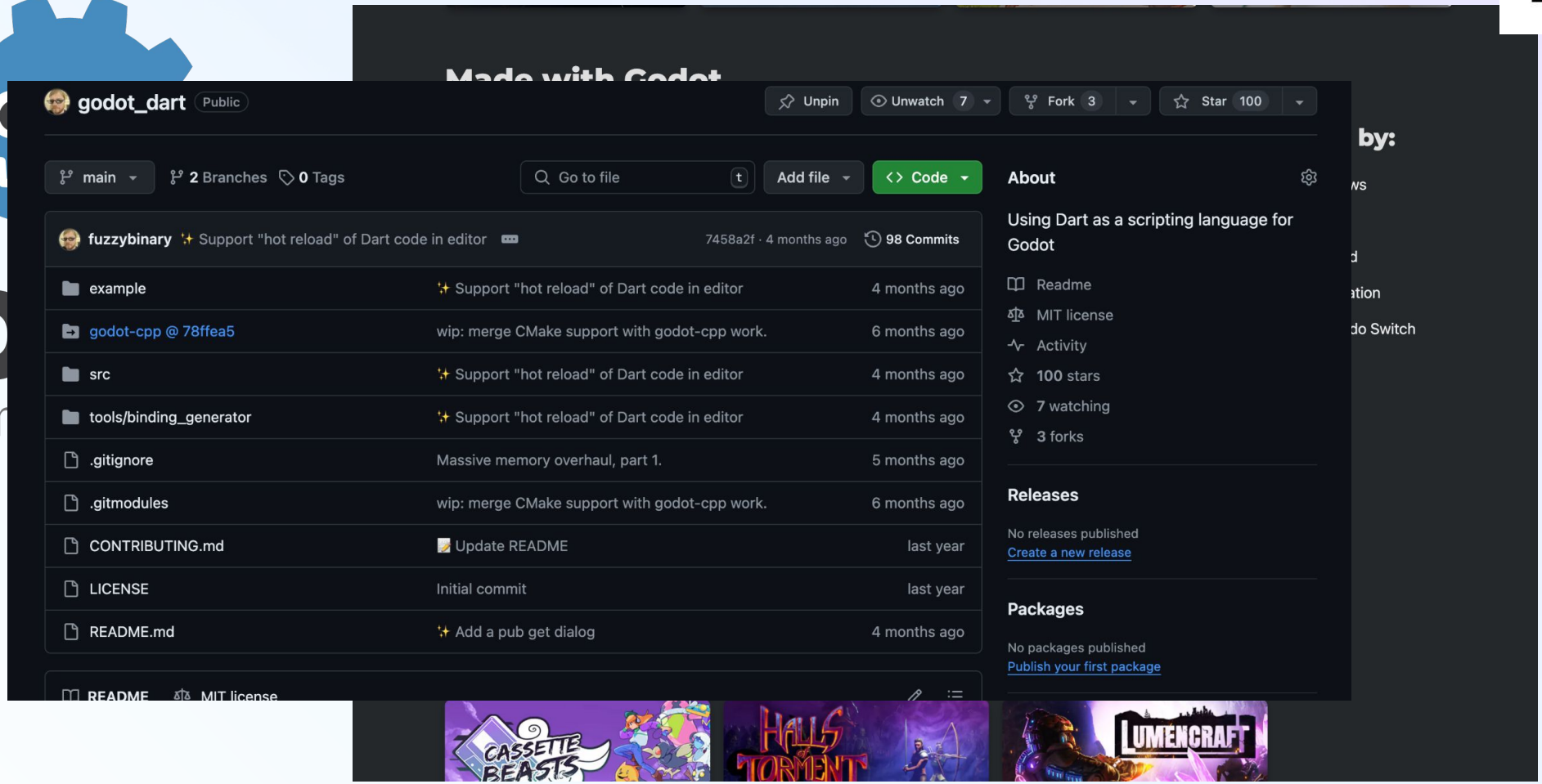
Helping Dart's GC - `Dart_NotifyIdle`

- Advisory info for Dart
- Tells Dart that you are unlikely to make any Dart calls in the foreseeable future, and helps it schedule garbage collection.
- Flutter does this by…
  - at the end of every frame, informing Dart how much time is left in the frame until the next frame (vsync)
  - If no frames are scheduled, sets an arbitrary "large" value.

"Again, this notification does not guarantee collection, just gives the Dart VM more hints about opportune moments to perform collections."

Putting it All Together

Waiting for Godot



Challenges with Godot

- Godot virtual methods start with _ – Dart doesn't like that.
  - Solution, virtual methods start with v now.
- Some features (varargs) are hard to replicate in Dart.

Challenges with Godot

- Relies heavily on code generation
  - Talking with Godot systems is all generated.
  - User code also requires generation.
  - Eventually will be helped by Dart macros

Dart

```dart
import 'dart:ffi';

import 'package:godot_dart/godot_dart.dart';

part 'player.g.dart';

@GodotScript()
class Player extends Area2D {
  static TypeInfo get sTypeInfo => _$PlayerTypeInfo();
  @override
  TypeInfo get typeInfo => Player.sTypeInfo;

  Player() : super();

  Player.withNonNullOwner(Pointer<Void> owner) : super.withNonNullOwner(owner);

  @GodotSignal('hit')
  late final Signal _hit = Signal.fromObjectSignal(this, 'hit');

  @GodotProperty()
  var speed = 400;

  late Vector2 _screenSize;

  @override
  void vReady() {
    hide();
    _screenSize = getViewportRect().size;
  }

  @override
  void vProcess(double delta) {
    var velocity = Vector2.fromXY(0, 0);
    var input = Input.singleton;
    if (input.isActionPressed('move_right')) {
      velocity.x += 1;
    }
    if (input.isActionPressed('move_left')) {
      velocity.x -= 1;
    }
    if (input.isActionPressed('move_down')) {
      velocity.y += 1;
    }
    if (input.isActionPressed('move_up')) {
      velocity.y -= 1;
    }
```

Challenges with Godot

- Godot is Multithreaded, Dart is “not”.
  - godot_dart’s solution is to move the main isolate from thread to thread (and exit it when we don’t need it)

C++

```
void GodotDartBindings::execute_on_dart_thread(std::function<void()> work) {
  std::thread::id current_thread_id = std::this_thread::get_id();
  if (_isolate_current_thread == current_thread_id) {
    work();
    return;
  }

  _work_lock.lock();
  _isolate_current_thread = std::this_thread::get_id();
  Dart_EnterIsolate(_isolate);
  work();

  Dart_ExitIsolate();
  _isolate_current_thread = std::thread::id();
  _work_lock.unlock();
}
```

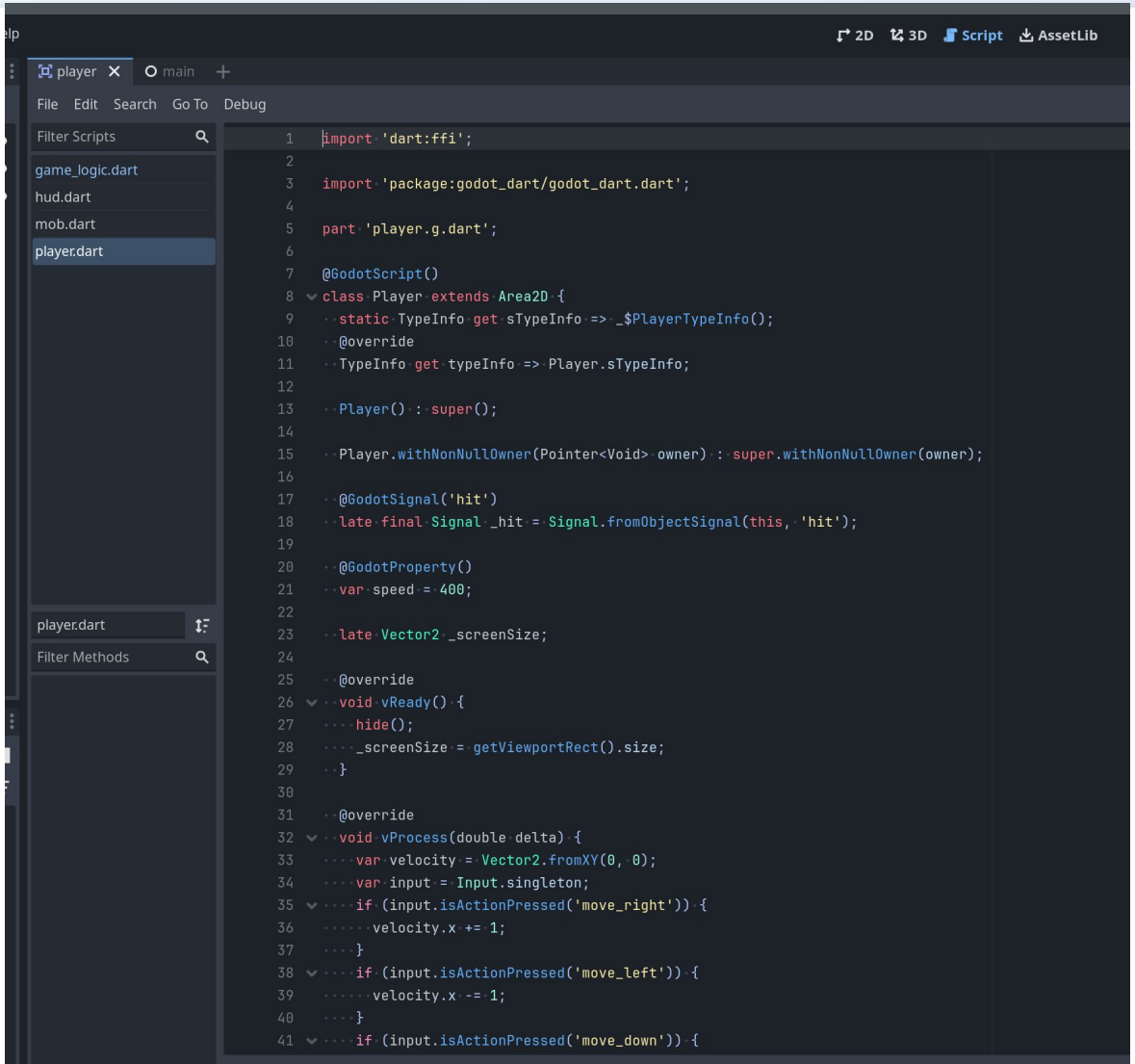
Challenges with Godot

- Godot's communication protocol is complicated, at times inherently unsafe.
  - (and requires a lot of memory allocation)

Dart

```
void setAnchorsAndOffsetsPreset(ControlLayoutPreset preset, {ControlLayoutPresetMode? resizeMode, int margin = 0}) {
  using((arena) {
    resizeMode ??= ControlLayoutPresetMode.presetModeMinsize;
    final ptrArgArray = arena.allocate<GDExtensionConstTypePtr>(sizeOf<GDExtensionConstTypePtr>() * 3);
    final presetPtr = arena.allocate<Uint32>(sizeOf<Uint32>())..value = preset.value;
    (ptrArgArray + 0).value = presetPtr.cast();
    final resizeModePtr = arena.allocate<Uint32>(sizeOf<Uint32>())..value = resizeMode!.value;
    (ptrArgArray + 1).value = resizeModePtr.cast();
    final marginPtr = arena.allocate<Int64>(sizeOf<Int64>())..value = margin;
    (ptrArgArray + 2).value = marginPtr.cast();
    gde.ffiBindings.gde_object_method_bind_ptrcall(
      _bindings.methodSetAnchorsAndOffsetsPreset, nativePtr.cast(), ptrArgArray, nullptr.cast());
  });
}
```

It is feasible



Demo Time

(or more like outta time….)

DATADOG

Disadvantages of Dart Embedding
(over other languages)

- Initial embedding is quite difficult
- Compile time
- Isolates vs. Threads
- The Dart team doesn’t support it
  - Though I have it on good authority this is changing

Help Do More with Dart

dart_shared_libray Public

Unpin | Unwatch 6 | Fork 1

main | 3 Branches | 0 Tags | Go to file | Add file | Code

fuzzybinary Remove output dir changes so assemble artifacts works. ccef86b · 4 months ago 39 Commits

.github/workflows	chore: Integrate build fixes	4 months ago
.vscode	Now Build and run Sample with 3.2.6	5 months ago
examples	chore: Integrate build fixes	4 months ago
scripts/build_helpers	Update To 3.3.0 and get FFI sample running	
src	chore: Integrate build fixes	
.clang-format	Initial commit	
.dart_version	Update To 3.3.0 and get FFI sample running	
.gitignore	chore: Integrate build fixes	
CMakeLists.txt	Remove output dir changes so assemble artifacts w	
LICENSE	Initial commit	

About

An attempt
usable shar

Readme
View lice
Activity
68 stars
6 watching

godot_dart Public

Unpin | Unwatch 7 | For

main | 2 Branches | 0 Tags | Go to file | Add file | Code

fuzzybinary ✨ Support "hot reload" of Dart code in editor 7458a2f · 4 months ago 98 Commits

example	✨ Support "hot reload" of Dart code in editor	4 months ago
godot-cpp @ 78ffea5	wip: merge CMake support with godot-cpp work.	6 months ago
src	✨ Support "hot reload" of Dart code in editor	4 months ago
tools/binding_generator	✨ Support "hot reload" of Dart code in editor	4 months ago
.gitignore	Massive memory overhaul, part 1.	5 months ago
.gitmodules	wip: merge CMake support with godot-cpp work.	6 months ago
CONTRIBUTING.md	📝 Update README	last year
LICENSE	Initial commit	last year
README.md	✨ Add a pub get dialog	4 months ago

README MIT license

About

Using D
Godot

Read
MIT
Activ
100 stars
7 watching
3 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Questions?

github.com/fuzzybinary
fuzzybinary@mastodon.gamedev.place

DATADOG

Thank you

github.com/fuzzybinary

fuzzybinary@mastodon.gamedev.place

@fuzzybinary.bsky.social

DATADOG