Using MLIR from C and Python

Alex Zinenko

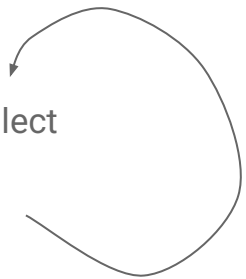
Agenda

- API design and basics
- Traversing IR
- Creating operations
- Creating attributes/types from a custom dialect
- Build system support
- Running passes

In C

Agenda

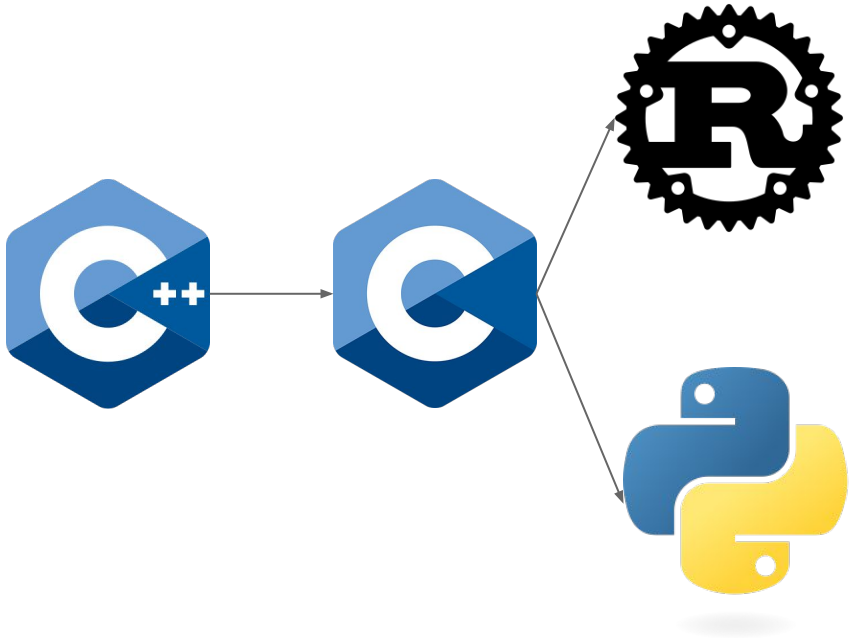
- API design and basics
- Traversing IR
- Creating operations
- Creating attributes/types from a custom dialect
- Build system support
- Running passes



In Python

C API Design Goals

- Primarily an interoperability layer with other languages.
- Bias towards minimalism at expense of usability.
- Weak stability guarantee.



Naming Conventions

General:

- Everything prefixed with mlir
- Types are capitalized:
  - MlirOperation
  - MlirAttribute
- Functions are not:
  - mlirOperationCreate
  - mlirAttributeGet

Functions:

- “Method” functions are prefixed with type:
  - mlirOperationCreate(MlirOperaiton, …)
  - mlirAttributeGet(MlirAttribute, …)
- Constructor/Destructor
  - MlirTypeNameCreate/Destroy
- Context-owned unique object
  - MlirTypeNameGet
- Accessor
  - MlirOperationGetContext

Type Model

Types are opaque structs typically holding a pointer:

- `MlirOperation` (`mlir::Operation *`)
- `MlirAttribute` (`mlir::Attribute` -> Impl *)
- `MlirType` (`mlir::Type` -> Impl *)
- …

Instances of every type are nullable:

- `bool mlir<Type>IsNull(Mlir<Type> x)`

Inheritance trees are not materialized. Functions always use the base type, but specify in the name if they expect a derived type:

- `mlirShapedTypeGetRank(MlirType type)`
- `mlirMemRefTypeGetLayout(MlirType type)`

and assert (`llvm::cast`) the expected type is given.

The user can check if an object is of a type:

```
bool mlirTypeIsAShapedType(MlirType type)
```

Traversing IR

Reminder: MLIR consists of:

- a top-level operation (module)
- with an attached list of regions
- that are linked lists of blocks
- that are linked lists of operations
  - with attached regions…

Traversing IR

Reminder: MLIR consists of:

- a top-level operation (module)
- with an attached list of regions
- that are linked lists of blocks
- that are linked lists of operations
  - with attached regions…

```
intptr_t mlirOperationGetNumRegions(MlirOperation op);
MlirRegion mlirOperationGetRegion(
  MlirOperation op, intptr_t pos);
```

Traversing IR

Reminder: MLIR consists of:

- a top-level operation (module)
- with an attached list of regions
- that are linked lists of blocks
- that are linked lists of operations
  - with attached regions…

```
intptr_t mlirOperationGetNumRegions(MlirOperation op);
MlirRegion mlirOperationGetRegion(
  MlirOperation op, intptr_t pos);

MlirBlock mlirRegionGetFirstBlock(MlirRegion region);
MlirBlock mlirBlockGetNextInRegion(MlirBlock block);
```

Traversing IR

Reminder: MLIR consists of:

- a top-level operation (module)
- with an attached list of regions
- that are linked lists of blocks
- that are linked lists of operations
  - with attached regions…

```
intptr_t mlirOperationGetNumRegions(MlirOperation op);
MlirRegion mlirOperationGetRegion(
  MlirOperation op, intptr_t pos);

MlirBlock mlirRegionGetFirstBlock(MlirRegion region);
MlirBlock mlirBlockGetNextInRegion(MlirBlock block);

MlirOperation mlirBlockGetFirstOperation(MlirBlock block);
MlirOperation mlirOperationGetNextInBlock(
  MlirOperation op);
```

Simple Example

```
MlirContext context = mlirContextCreate();
```

Simple Example

```
MlirContext context = mlirContextCreate();
MlirOperation module =                                                  // Parse from source
    mlirOperationCreateParse(context, mlirStringRefCreateFromCString("..."),
                             mlirStringRefCreateFromCString("input.mlir"));  // File name of the source
```

Simple Example

```
MlirContext context = mlirContextCreate();
MlirOperation module =                                                  // Parse from source
    mlirOperationCreateParse(context, mlirStringRefCreateFromCString("..."),
                             mlirStringRefCreateFromCString("input.mlir"));  // File name of the source
MlirRegion body = mlirOperationGetFirstRegion(module);                  // First region of the module
```

,

Simple Example

```
MlirContext context = mlirContextCreate();
MlirOperation module =                                                  // Parse from source
    mlirOperationCreateParse(context, mlirStringRefCreateFromCString("..."),
                             mlirStringRefCreateFromCString("input.mlir"));  // File name of the source
MlirRegion body = mlirOperationGetFirstRegion(module);                  // First region of the module
MlirBlock bodyBlock = mlirRegionGetFirstBlock(body);                    // First block of the region
```

Simple Example

```
MlirContext context = mlirContextCreate();
MlirOperation module =                                                  // Parse from source
    mlirOperationCreateParse(context, mlirStringRefCreateFromCString("..."),
                             mlirStringRefCreateFromCString("input.mlir"));  // File name of the source
MlirRegion body = mlirOperationGetFirstRegion(module);                  // First region of the module
MlirBlock bodyBlock = mlirRegionGetFirstBlock(body);                    // First block of the region
MlirStringRef visibility = mlirSymbolTableGetVisibilityAttributeName();
MlirStringRef publicVisibility = mlirStringRefCreateFromCString("public");
```

Simple Example

```
MlirContext context = mlirContextCreate();
MlirOperation module =                                                  // Parse from source
    mlirOperationCreateParse(context, mlirStringRefCreateFromCString("..."),
                             mlirStringRefCreateFromCString("input.mlir"));  // File name of the source
MlirRegion body = mlirOperationGetFirstRegion(module);                  // First region of the module
MlirBlock bodyBlock = mlirRegionGetFirstBlock(body);                    // First block of the region
MlirStringRef visibility = mlirSymbolTableGetVisibilityAttributeName();
MlirStringRef publicVisibility = mlirStringRefCreateFromCString("public");
for (MlirOperation op = mlirBlockGetFirstOperation(bodyBlock);          // Iterate over the linked list
     !mlirOperationIsNull(op);                                          // of operations in the block
     op = mlirOperationGetNextInBlock(op)) {




}
```

Simple Example

```
MlirContext context = mlirContextCreate();
MlirOperation module =                                                  // Parse from source
    mlirOperationCreateParse(context, mlirStringRefCreateFromCString("..."),
                             mlirStringRefCreateFromCString("input.mlir"));  // File name of the source
MlirRegion body = mlirOperationGetFirstRegion(module);                  // First region of the module
MlirBlock bodyBlock = mlirRegionGetFirstBlock(body);                    // First block of the region
MlirStringRef visibility = mlirSymbolTableGetVisibilityAttributeName();
MlirStringRef publicVisibility = mlirStringRefCreateFromCString("public");
for (MlirOperation op = mlirBlockGetFirstOperation(bodyBlock);          // Iterate over the linked list
     !mlirOperationIsNull(op);                                          // of operations in the block
     op = mlirOperationGetNextInBlock(op)) {
  MlirAttribute visibilityAttr =                                        // Assuming top-level ops are
      mlirOperationGetAttributeByName(op, visibility);                  // symbols, find those with
  MlirStringRef visibilityStr = mlirStringAttrGetValue(visibilityAttr);// public visibility


}
```

Simple Example

```
MlirContext context = mlirContextCreate();
MlirOperation module =                                                  // Parse from source
    mlirOperationCreateParse(context, mlirStringRefCreateFromCString("..."),
                             mlirStringRefCreateFromCString("input.mlir"));  // File name of the source
MlirRegion body = mlirOperationGetFirstRegion(module);                  // First region of the module
MlirBlock bodyBlock = mlirRegionGetFirstBlock(body);                    // First block of the region
MlirStringRef visibility = mlirSymbolTableGetVisibilityAttributeName();
MlirStringRef publicVisibility = mlirStringRefCreateFromCString("public");
for (MlirOperation op = mlirBlockGetFirstOperation(bodyBlock);          // Iterate over the linked list
     !mlirOperationIsNull(op);                                          // of operations in the block
     op = mlirOperationGetNextInBlock(op)) {
  MlirAttribute visibilityAttr =                                        // Assuming top-level ops are
      mlirOperationGetAttributeByName(op, visibility);                  // symbols, find those with
  MlirStringRef visibilityStr = mlirStringAttrGetValue(visibilityAttr);// public visibility
  if (mlirStringRefEqual(visibilityStr, publicVisibility))
    do something
}
```

Creating IR from C

Aka the “stable” API to MLIR

Ownership Model

By default, no ownership transfer.

`Mlir<Type> mlir<Type>Create(…)` -> the caller owns the result and *must destroy it* or transfer ownership.

Ownership Model

By default, no ownership transfer.

`Mlir<Type> mlir<Type>Create(…)` -> the caller owns the result and *must destroy it* or transfer ownership.

`Mlir<Type> mlir<Other>Take<Type>(…)` -> the caller owns the result with similar requirements.

Ownership Model

By default, no ownership transfer.

`Mlir<Type> mlir<Type>Create(…)` -> the caller owns the result and *must destroy it* or transfer ownership.

`Mlir<Type> mlir<Other>Take<Type>(…)` -> the caller owns the result with similar requirements.

`… mlir<something>Owned<Type>(…, Mlir<Type)` -> the caller transfers ownership to the callee.

Ownership Model

By default, no ownership transfer.

`Mlir<Type> mlir<Type>Create(…)` -> the caller owns the result and *must destroy it* or transfer ownership.

`Mlir<Type> mlir<Other>Take<Type>(…)` -> the caller owns the result with similar requirements.

`… mlir<something>Owned<Type>(…, Mlir<Type)` -> the caller transfers ownership to the callee.

`Mlir<Type> mlir<Type>Get(MlirContext)` -> the context owns the object.

Trivial Example

```
#include "mlir-c/IR.h"

MlirContext context = mlirContextCreate();                  // Create an owned context
```

Trivial Example

```
#include "mlir-c/IR.h"

MlirContext context = mlirContextCreate();                  // Create an owned context
MlirOperationState state = mlirOperationStateGet(                 // Create operation state to prepare
  mlirStringRefCreateFromCString("builtin.module"),               // Non-owning string reference
  mlirLocationUnknownGet(context));                               // Locations are owned by context
```

Trivial Example

```
#include "mlir-c/IR.h"

MlirContext context = mlirContextCreate();                  // Create an owned context
MlirOperationState state = mlirOperationStateGet(                 // Create operation state to prepare
  mlirStringRefCreateFromCString("builtin.module"),               // Non-owning string reference
  mlirLocationUnknownGet(context));                               // Locations are owned by context
MlirRegion region = mlirRegionCreate();                           // Create owned region
```

Trivial Example

```
#include "mlir-c/IR.h"

MlirContext context = mlirContextCreate();                  // Create an owned context
MlirOperationState state = mlirOperationStateGet(                 // Create operation state to prepare
  mlirStringRefCreateFromCString("builtin.module"),               // Non-owning string reference
  mlirLocationUnknownGet(context));                               // Locations are owned by context
MlirRegion region = mlirRegionCreate();                           // Create owned region
MlirRegion regions[] = {region};                                  // Make a list of regions
mlirOperationStateAddOwnedRegions(                                // Transfer owned regions to the operation
    &state, sizeof(regions) / sizeof(MlirRegion), regions);       //    state
```

Trivial Example

```
#include "mlir-c/IR.h"

MlirContext context = mlirContextCreate();                  // Create an owned context
MlirOperationState state = mlirOperationStateGet(                 // Create operation state to prepare
  mlirStringRefCreateFromCString("builtin.module"),               // Non-owning string reference
  mlirLocationUnknownGet(context));                               // Locations are owned by context
MlirRegion region = mlirRegionCreate();                           // Create owned region
MlirRegion regions[] = {region};                                  // Make a list of regions
mlirOperationStateAddOwnedRegions(                                // Transfer owned regions to the operation
    &state, sizeof(regions) / sizeof(MlirRegion), regions);       //    state
MlirOperation module = mlirOperationCreate(state);                // Create owned operation
...
```

Trivial Example

```
#include "mlir-c/IR.h"

MlirContext context = mlirContextCreate();                  // Create an owned context
MlirOperationState state = mlirOperationStateGet(                 // Create operation state to prepare
  mlirStringRefCreateFromCString("builtin.module"),               // Non-owning string reference
  mlirLocationUnknownGet(context));                               // Locations are owned by context
MlirRegion region = mlirRegionCreate();                           // Create owned region
MlirRegion regions[] = {region};                                  // Make a list of regions
mlirOperationStateAddOwnedRegions(                                // Transfer owned regions to the operation
    &state, sizeof(regions) / sizeof(MlirRegion), regions);       //    state
MlirOperation module = mlirOperationCreate(state);                // Create owned operation
...
mlirOperationDestroy(module);                                     // Free owned operation recursively
mlirContextDestroy(context);                                      // Free owned context
```

Creating Operations

Using the generic state object:

- Name as StringRef
- Location
- List of operands
- List of result types
- List of attributes (also properties)
- List of *owned* regions
- List of successors

Works for any dialect, but may be slow because of string name lookup.

```
mlirOperationStateGet(MlirStringRef name,
                      MlirLocation loc);
mlirOperationStateAddOperands(...);
mlirOperationStateAddResults(...);
mlirOperationStateAddAttributes(...);
mlirOperationStateAddOwnedRegions(...);
mlirOperationStateAddSuccessors(...);
```

Creating Operations

Common signature:

```
mlirOperationStateAdd<...>(
   MlirOperationState *state,
   intptr_t n,
   Mlir<Type> *elements)
```

```
mlirOperationStateGet(MlirStringRef name,
                      MlirLocation loc);
mlirOperationStateAddOperands(...);
mlirOperationStateAddResults(...);
mlirOperationStateAddAttributes(...);
mlirOperationStateAddOwnedRegions(...);
mlirOperationStateAddSuccessors(...);
```

Creating Operations

- Works for any dialect out of the box
- Does *not* call the build function, leading to duplication.
- Does *not* verify.

Creating Operations

- Works for any dialect out of the box.
- Does *not* call the build function, leading to duplication.
- Does *not* verify.

Creating Types

- No generic format unlike operations.
- Requires defining additional functions.

Creating Types

Declaration

```
#include "mlir-c/IR.h"
#include "mlir-c/Support.h"

#ifdef __cplusplus
extern "C" {
#endif

MLIR_DECLARE_CAPI_DIALECT_REGISTRATION(
      Transform, transform);
```

Creating Types

Declaration

```
#include "mlir-c/IR.h"
#include "mlir-c/Support.h"

#ifdef __cplusplus
extern "C" {
#endif

MLIR_DECLARE_CAPI_DIALECT_REGISTRATION(
      Transform, transform);
```

Definition

```
#include "mlir-c/Support.h"
#include "mlir/CAPI/Registration.h"

MLIR_DEFINE_CAPI_DIALECT_REGISTRATION(
      Transform, transform,
      transform::TransformDialect)
```

Creating Types (cont'd)

Declaration

```
MLIR_CAPI_EXPORTED bool
mlirTypeIsATransformOperationType(MlirType type);


MLIR_CAPI_EXPORTED MlirTypeID
mlirTransformOperationTypeGetTypeID(void);



MLIR_CAPI_EXPORTED MlirType
mlirTransformOperationTypeGet(MlirContext ctx,
  MlirStringRef operationName);
```

Creating Types (cont'd)

Declaration

```
MLIR_CAPI_EXPORTED bool
mlirTypeIsATransformOperationType(MlirType type);


MLIR_CAPI_EXPORTED MlirTypeID
mlirTransformOperationTypeGetTypeID(void);



MLIR_CAPI_EXPORTED MlirType
mlirTransformOperationTypeGet(MlirContext ctx,
  MlirStringRef operationName);
```

Definition

```
bool mlirTypeIsATransformOperationType(
      MlirType type) {
 return isa<transform::OperationType>(
   unwrap(type)); }

MlirTypeID
mlirTransformOperationTypeGetTypeID(void) {
 return wrap(
   transform::OperationType::getTypeID()); }

MlirType
mlirTransformOperationTypeGet(MlirContext ctx,
    MlirStringRef operationName) {
 return wrap(transform::OperationType::get(
    unwrap(ctx), unwrap(operationName))); }
```

Creating Types (cont'd)

Declaration

```
MLIR_CAPI_EXPORTED MlirStringRef
mlirTransformOperationTypeGetOperationName(
  MlirType type);

#ifdef __cplusplus
}
#endif
```

Creating Types (cont'd)

Declaration

```
MLIR_CAPI_EXPORTED MlirStringRef
mlirTransformOperationTypeGetOperationName(
  MlirType type);

#ifdef __cplusplus
}
#endif
```

Definition

```
MlirStringRef
mlirTransformOperationTypeGetOperationName(
    MlirType type) {
 return wrap(cast<transform::OperationType>(
    unwrap(type)).getOperationName());
}
```

Creating Types (cont’d)

CMakeLists.txt

```
add_mlir_public_c_api_library(MLIRCAPITransformDialect
 Transform.cpp

 PARTIAL_SOURCES_INTENDED
 LINK_LIBS PUBLIC
 MLIRCAPIIR
 MLIRTransformDialect
)
```

Using Interfaces

Regular (instance) interfaces

Similar to operations/types:
- not materialized as a C struct
- is-a method can be replaced with TypeID

```
MLIR_CAPI_EXPORTED MlirTypeID
mlirInferTypeOpInterfaceTypeID();

MLIR_CAPI_EXPORTED bool
mlirOperationImplementsInterface(
  MlirOperation operation, MlirTypeID ifaceID);

MLIR_CAPI_EXPORTED int
mlirSomeOpInterfaceDoSomething(MlirOperation)
```

Using Interfaces

Regular (instance) interfaces

Similar to operations/types:
- not materialized as a C struct
- is-a method can be replaced with TypeID

```
MLIR_CAPI_EXPORTED MlirTypeID
mlirInferTypeOpInterfaceTypeID();

MLIR_CAPI_EXPORTED bool
mlirOperationImplementsInterface(
  MlirOperation operation, MlirTypeID ifaceID);

MLIR_CAPI_EXPORTED int
mlirSomeOpInterfaceDoSomething(MlirOperation)
```

Static interfaces

Uses MlirStringRef for operation name, MlirTypeID for attribute/type name.

```
MLIR_CAPI_EXPORTED bool
mlirOperationImplementsInterfaceStatic(
  MlirStringRef operationName, MlirContext ctx,
  MlirTypeID interfaceTypeID);

MLIR_CAPI_EXPORTED MlirLogicalResult
mlirInferTypeOpInterfaceInferReturnTypes(
   MlirStringRef opName, ...);
```

Mutating IR from C

In case you want to run a compiler in something other than C++

Running a Pass Pipeline

```
#include "mlir-c/Pass.h"

void runPassPipeline(MlirContext context, MlirOperation module) {
```

Running a Pass Pipeline

```
#include "mlir-c/Pass.h"

void runPassPipeline(MlirContext context, MlirOperation module) {
 MlirPassManager pm = mlirPassManagerCreateOnOperation(
     context, mlirStringRefCreateFromCString("builtin.module"));
 MlirOpPassManager opm = mlirPassManagerGetAsOpPassManager(pm);
```

Running a Pass Pipeline

```
#include "mlir-c/Pass.h"

void runPassPipeline(MlirContext context, MlirOperation module) {
 MlirPassManager pm = mlirPassManagerCreateOnOperation(
     context, mlirStringRefCreateFromCString("builtin.module"));
 MlirOpPassManager opm = mlirPassManagerGetAsOpPassManager(pm);
 char *error = 0;
 MlirLogicalResult result = mlirParsePassPipeline(
     opm, mlirStringRefCreateFromCString("canonicalize,cse"), appendError,
     error);
```

Running a Pass Pipeline

```
#include "mlir-c/Pass.h"

void runPassPipeline(MlirContext context, MlirOperation module) {
 MlirPassManager pm = mlirPassManagerCreateOnOperation(
     context, mlirStringRefCreateFromCString("builtin.module"));
 MlirOpPassManager opm = mlirPassManagerGetAsOpPassManager(pm);
 char *error = 0;
 MlirLogicalResult result = mlirParsePassPipeline(
     opm, mlirStringRefCreateFromCString("canonicalize,cse"), appendError,
     error);
 if (mlirLogicalResultIsFailure(result))
   fprintf(stderr, "%s\n", error);

 ...
```

Running a Pass Pipeline

```
#include "mlir-c/Pass.h"
 mlirRegisterTransformsCSE()
 mlirRegisterTransformsCanonicalizer()
 mlirRegisterTransformsPasses()
```

But, registration?!

Running a Pass Pipeline

```
#include "mlir-c/Pass.h"

void runPassPipeline(MlirContext context, MlirOperation module) {
 MlirPassManager pm = mlirPassManagerCreateOnOperation(
     context, mlirStringRefCreateFromCString("builtin.module"));
 mlirPassManagerAddOwnedPass(pm, mlirCreateTransformsCanonicalizer());
 mlirPassManagerAddOwnedPass(pm, mlirCreateTransformsCSE());

 MlirLogicalResult result = mlirPassManagerRunOnOp(pm, module);
 if (mlirLogicalResultIsFailure(result))
   // report pass error
 mlirPassManagerDestroy(pm);
}
```

Where are these defined?

Running a Pass Pipeline

```
set(LLVM_TARGET_DEFINITIONS Passes.td)
mlir_tablegen(Passes.h.inc -gen-pass-decls -name GPU)
mlir_tablegen(Passes.capi.h.inc -gen-pass-capi-header --prefix GPU)
mlir_tablegen(Passes.capi.cpp.inc -gen-pass-capi-impl --prefix GPU)
add_public_tablegen_target(MLIRGPUPassIncGen)
```

Creating an External Pass

```
struct MlirExternalPassCallbacks {
 void (*construct)(void *userData);                                    // Pass::Pass
```

Creating an External Pass

```
struct MlirExternalPassCallbacks {
 void (*construct)(void *userData);                    // Pass::Pass

 void (*destruct)(void *userData);                     // Pass::~Pass
```

Creating an External Pass

```
struct MlirExternalPassCallbacks {
 void (*construct)(void *userData);                                        // Pass::Pass

 void (*destruct)(void *userData);                                         // Pass::~Pass

 MlirLogicalResult (*initialize)(MlirContext ctx, void *userData);         // Pass::initialize(MLIRContext *)
```

Creating an External Pass

```
struct MlirExternalPassCallbacks {
 void (*construct)(void *userData);                                        // Pass::Pass

 void (*destruct)(void *userData);                                         // Pass::~Pass

 MlirLogicalResult (*initialize)(MlirContext ctx, void *userData);         // Pass::initialize(MLIRContext *)

 void *(*clone)(void *userData);                                           // Pass::clonePass()
```

Creating an External Pass

```
struct MlirExternalPassCallbacks {
 void (*construct)(void *userData);                                        // Pass::Pass

 void (*destruct)(void *userData);                                         // Pass::~Pass

 MlirLogicalResult (*initialize)(MlirContext ctx, void *userData);         // Pass::initialize(MLIRContext *)

 void *(*clone)(void *userData);                                           // Pass::clonePass()

 void (*run)(MlirOperation op, MlirExternalPass pass, void *userData); // Pass::runOnOperation().
};
```

Creating an External Pass

```
struct MlirExternalPassCallbacks {
 void (*construct)(void *userData);                                        // Pass::Pass

 void (*destruct)(void *userData);                                         // Pass::~Pass

 MlirLogicalResult (*initialize)(MlirContext ctx, void *userData);         // Pass::initialize(MLIRContext *)

 void *(*clone)(void *userData);                                           // Pass::clonePass()

 void (*run)(MlirOperation op, MlirExternalPass pass, void *userData); // Pass::runOnOperation().
};

MLIR_CAPI_EXPORTED MlirPass mlirCreateExternalPass(MlirTypeID passID, MlirStringRef name,
   MlirStringRef argument, MlirStringRef description, MlirStringRef opName, intptr_t nDependentDialects,
   MlirDialectHandle *dependentDialects, MlirExternalPassCallbacks callbacks, void *userData);
```

Using MLIR from Python

Python is native to many ML frameworks

Python API Design

Support users who expect that an installed version of LLVM/MLIR will yield the ability to `import mlir` and use the API in a pure way out of the box.

Downstream integrations will likely want to include parts of the API in their private namespace or specially built libraries, probably mixing it with other python native bits.

- Build on C API
  (avoid linking and exception problems)
- Use pybind11 to define API
  (nanobind anyone?)
- Header-only C++ utilities are okay.
- Explicit registration.

Everything must be linked into one big library

MLIR registration mechanism is hard…
Typical symptoms: assertion about repeated registration, or pass / operation not found despite being clearly loaded.

Traversing IR

Reminder: MLIR consists of:

- a top-level operation (module)
- with an attached list of regions
- that are linked lists of blocks
- that are linked lists of operations
  - with attached regions…

```
from mlir import ir

def traverse_ir(op: ir.Operation):
  for region in op.regions:
    for block in region.blocks:
      for nested in block.operations:
        print(nested.attributes["my_attr"])
```

Structure and Packaging Conventions

Naming and Structure

- Drop the `mlir` prefix (`Operation`, `Type`, `Context`)
- Use properties for simple, always-possible accessors and explicit methods otherwise (`.context` vs `.get_asm()`)
- Use container-like objects compatible with Python protocols (`Iterable`, `Dict`).
- Objects nullable in C may be passed as `None`.

Structure and Packaging Conventions

Naming and Structure

- Drop the `mlir` prefix (`Operation`, `Type`, `Context`)
- Use properties for simple, always-possible accessors and explicit methods otherwise (`.context` vs `.get_asm()`)
- Use container-like objects compatible with Python protocols (`Iterable`, `Dict`).
- Objects nullable in C may be passed as `None`.

Packaging

- Core IR components live in the `mlir.ir` package.
- Individual dialects live in subpackages of `mlir.dialects`, e.g., `mlir.dialects.linalg`.
- Dialect-specific passes, and generally other C++ libraries map to subpackages.

Simple Example

```
from mlir import ir
```

Simple Example

```
from mlir import ir

with ir.Context() as ctx:                                   // Context is a Python context manager
```

Simple Example

```
from mlir import ir

with ir.Context() as ctx:                          // Context is a Python context manager
   top_level = ir.Module.parse("...", context=ctx) // Parsing requires a context
```

Simple Example

```
from mlir import ir

with ir.Context() as ctx:                          // Context is a Python context manager
   top_level = ir.Module.parse("...", context=ctx) // Parsing requires a context
   body = top_level.regions[0]                  // Regions/blocks are directly indexable
   body_block = body.blocks[0]                  // but it is expensive as they are linked lists
```

Simple Example

```python
from mlir import ir

with ir.Context() as ctx:
    top_level = ir.Module.parse("...", context=ctx)
    body = top_level.regions[0]
    body_block = body.blocks[0]
    for op in body_block.operations:
```

```
    // Context is a Python context manager
    // Parsing requires a context
// Regions/blocks are directly indexable
// but it is expensive as they are linked lists
```

Simple Example

```python
from mlir import ir

with ir.Context() as ctx:
   top_level = ir.Module.parse("...", context=ctx)
   body = top_level.regions[0]
   body_block = body.blocks[0]
   for op in body_block.operations:
     visibility = op.attributes["sym_visibility"]
     if visibility is None: continue
```

```
                                        // Context is a Python context manager
                                        // Parsing requires a context
// Regions/blocks are directly indexable
// but it is expensive as they are linked lists

                                        // Attributes may be accessed by name
// None is used pervasively
```

Simple Example

```python
from mlir import ir

with ir.Context() as ctx:
   top_level = ir.Module.parse("...", context=ctx)
   body = top_level.regions[0]
   body_block = body.blocks[0]
   for op in body_block.operations:
     visibility = op.attributes["sym_visibility"]
     if visibility is None: continue
     if (visibility.value == "public"):
       do_something(op)
```

```
      // Context is a Python context manager
      // Parsing requires a context
// Regions/blocks are directly indexable
// but it is expensive as they are linked lists

      // Attributes may be accessed by name
// None is used pervasively
      // Simple accessors are Python properties
```

Simple Example

Can be derived from surrounding context managers!

```python
from mlir import ir

with ir.Context() as ctx:
   top_level = ir.Module.parse("...", context=ctx)
   body = top_level.regions[0]
   body_block = body.blocks[0]
   for op in body_block.operations:
     visibility = op.attributes["sym_visibility"]
     if visibility is None: continue
     if (visibility.value == "public"):
       do_something(op)
```

```
// Context is a Python context manager
// Parsing requires a context
// Regions/blocks are directly indexable
// but it is expensive as they are linked lists

// Attributes may be accessed by name
// None is used pervasively
// Simple accessors are Python properties
```

Creating IR from Python

The main use case

Ownership Model

Owned by Python caller:

- Context: `mlir.ir.Context`
- Module: `mlir.ir.Module`
- *Detached* operation, i.e. operation not nested in another operation, including modules: `mlir.ir.Operation`

Kept alive:

- Immutable objects owned by context keep alive the context.
- Operations keep alive ancestors until a detached one or a module.

Warning: erasing an operation in C++ code is *invisible* to Python.

Trivial Example

```
from mlir import ir

with ir.Context():                                  # Context is a context manager
```

Trivial Example

```
from mlir import ir


with ir.Context():
 with ir.UnknownLoc():
```

```
# Context is a context manager
      # Location is a context manager
```

Trivial Example

```
from mlir import ir


with ir.Context():
 with ir.UnknownLoc():
   module = ir.Module.create()
   with ir.InsertionPoint(module.regions[0].blocks[0]):
```

```
# Context is a context manager
      # Location is a context manager
# Context/location inferred from context
      # Insertion point is a context manager
```

Trivial Example

```
from mlir import ir


with ir.Context():
 with ir.UnknownLoc():
   module = ir.Module.create()
   with ir.InsertionPoint(module.regions[0].blocks[0]):
     ir.Operation.create("func.func",
       regions=1,
       attributes={"function_type":
                     ir.FunctionType.get([], []),
                   "sym_name": ir.StrAttr.get("f")})
```

```
# Context is a context manager
      # Location is a context manager
# Context/location inferred from context
      # Insertion point is a context manager
      # Created at the inferred insertion pt
      # Operation state as kwargs
      # NOTE: no verification

      # Non-owned objects use implicit context
```

Trivial Example

```
from mlir import ir


with ir.Context():
 with ir.UnknownLoc():
   module = ir.Module.create()
   with ir.InsertionPoint(module.regions[0].blocks[0]):
     ir.Operation.create("func.func",
       regions=1,
       attributes={"function_type":
                     ir.FunctionType.get([], []),
                   "sym_name": ir.StrAttr.get("f")})
   # ...
```

```
# Context is a context manager
      # Location is a context manager
# Context/location inferred from context
      # Insertion point is a context manager
      # Created at the inferred insertion pt
      # Operation state as kwargs
      # NOTE: no verification

      # Non-owned objects use implicit context

      # Past this point, Python may
      # garbage-collect everything
```

Trivial Example

```
from mlir import ir
from mlir.dialects import func

with ir.Context():
 with ir.UnknownLoc():
   module = ir.Module.create()
   with ir.InsertionPoint(module.regions[0].blocks[0]):
     func.FuncOp("f", ([], []))                    # Dialects may provide custom builders
   # ...
```

Custom Builders

```
from mlir import ir
from mlir.dialects import func

with ir.Context():
 with ir.UnknownLoc():
   module = ir.Module.create()
   with ir.InsertionPoint(module.regions[0].blocks[0]):
     func.FuncOp("f", ([], []))
   # ...
```

In mlir/python/dialects/func.py

```
from ._func_ops_gen import *, _Dialect
from ._ods_common import _cext as _ods_cext

@_ods_cext.register_operation(
  _Dialect, replace=True)
class FuncOp(FuncOp):
   def __init__(self, name, type, *,
       visibility=None, body_builder=None,
       loc=None, ip=None):
     # ...
     super().__init__(..., loc=loc, ip=ip)
```

Custom Builders

In mlir/python/dialects/func.py

Autogenerated from ODS:
`mlir-tblgen -gen-python-op-bindings`

```
from ._func_ops_gen import *, _Dialect
from ._ods_common import _cext as _ods_cext
```

Core binding functionality

Override default object for this operation

```
@_ods_cext.register_operation(
  _Dialect, replace=True)
class FuncOp(FuncOp):
```

Constructor

```
  def __init__(self, name, type, *,
      visibility=None, body_builder=None,
      loc=None, ip=None):
    # ...
```

Location and insertion point *must* use `loc`, `ip` names

Delegate to the generated constructor

```
    super().__init__(..., loc=loc, ip=ip)
```

Warning: inspect the generated constructor first

Additional Op Functionality

Autogenerated from ODS:
`mlir-tblgen -gen-python-op-bindings`

In mlir/python/dialects/func.py

```
from ._func_ops_gen import *, _Dialect
from ._ods_common import _cext as _ods_cext

@_ods_cext.register_operation(
  _Dialect, replace=True)
class FuncOp(FuncOp):

  @property
  def sym_name(self):
    return self.operation.attributes["sym_name"]
```

For example, convenience accessors
(many are autogenerated in the base class)

Custom Builders

In mlir/include/mlir/Dialect/Func/IR/FuncOps.td

Autogenerated from ODS:
`mlir-tblgen -gen-python-op-bindings`

```
@_ods_cext.register_operation(_Dialect)
class FuncOp(_ods_ir.OpView):
  def __init__(self, sym_name, function_type, *,
      sym_visibility=None, arg_attrs=None,
      res_attrs=None, loc=None, ip=None):
    ...
```



```
let arguments = (ins SymbolNameAttr:$sym_name,
  TypeAttrOf<FunctionType>:$function_type,
  OptionalAttr<StrAttr>:$sym_visibility,
  OptionalAttr<DictArrayAttr>:$arg_attrs,
  OptionalAttr<DictArrayAttr>:$res_attrs);
```

Same order as ODS arguments, types extracted from conversion rules

Automatic Conversion of Types

In mlir/include/mlir/Dialect/Func/IR/FuncOps.td

Autogenerated from ODS:
`mlir-tblgen -gen-python-op-bindings`

```
@_ods_cext.register_operation(_Dialect)
class FuncOp(_ods_ir.OpView):
  def __init__(self, sym_name, function_type, *,
      sym_visibility=None, arg_attrs=None,
      res_attrs=None, loc=None, ip=None):
    attributes["sym_name"] = (sym_name if (
    isinstance(sym_name, _ods_ir.Attribute) or
    not _ods_ir.AttrBuilder.contains('SymbolNameAttr')) else
      _ods_ir.AttrBuilder.get('SymbolNameAttr')(sym_name, context=_ods_context))
```



```
let arguments = (ins SymbolNameAttr:$sym_name,
  TypeAttrOf<FunctionType>:$function_type,
  OptionalAttr<StrAttr>:$sym_visibility,
  OptionalAttr<DictArrayAttr>:$arg_attrs,
  OptionalAttr<DictArrayAttr>:$res_attrs);
```

Automatic Conversion of Types

Autogenerated from ODS:
`mlir-tblgen -gen-python-op-bindings`

```
@_ods_cext.register_operation(_Dialect)
class FuncOp(_ods_ir.OpView):
  def __init__(self, sym_name, function_type, *,
      sym_visibility=None, arg_attrs=None,
      res_attrs=None, loc=None, ip=None):
    attributes["sym_name"] = (sym_name if (
    isinstance(sym_name, _ods_ir.Attribute) or
    not _ods_ir.AttrBuilder.contains('SymbolNameAttr')) else
      _ods_ir.AttrBuilder.get('SymbolNameAttr')(sym_name, context=_ods_context))
```

In mlir/python/mlir/ir.py

```
def register_attribute_builder(kind):
   def decorator_builder(func):
       AttrBuilder.insert(kind, func)
       return func
   return decorator_builder

@register_attribute_builder("SymbolNameAttr")
def _symbolNameAttr(x, context):
  return StringAttr.get(x, context=context)
```

Automatic Conversion of Types

`Attr/TypeBuilder` is a map from ODS name to the function creating the corresponding object.

Already defined for (most) core types.

Can be easily added for user-defined attributes and types assuming they are available in Python.

In mlir/python/mlir/ir.py

```
def register_attribute_builder(kind):
    def decorator_builder(func):
        AttrBuilder.insert(kind, func)
        return func
    return decorator_builder


@register_attribute_builder("SymbolNameAttr")
def _symbolNameAttr(x, context):
  return StringAttr.get(x, context=context)
```

Bindings for Custom Attributes and Types

```
#include "mlir/Bindings/Python/PybindAdaptors.h"

auto operationType =
     mlir_type_subclass(m, "OperationType", mlirTypeIsATransformOperationType,
                        mlirTransformOperationTypeGetTypeID);
operationType.def_classmethod("get",
     [](py::object cls, const std::string &operationName, MlirContext ctx) {
       return cls(mlirTransformOperationTypeGet(ctx, cOperationName));
     });
operationType.def_property_readonly(
     "operation_name", [](MlirType type) {
       MlirStringRef operationName = mlirTransformOperationTypeGetOperationName(type);
       return py::str(operationName.data, operationName.length);
     });
```

Requires is-a and TypeID

C types are supported by pybind11 type casters

Don’t forget other conversions

Bindings for Custom Attributes and Types

```cpp
#include "mlir/Bindings/Python/PybindAdaptors.h"

void populateSubmodule(const pybind11::module &m) {
  auto operationType =
     mlir_type_subclass(...);
  operationType.def_classmethod("get", ...);
  operationType.def_property_readonly(
     "operation_name", ...);
}

PYBIND11_MODULE(_mlirDialectsTransform, m) {
  m.doc() = "MLIR Transform dialect.";
  populateSubmodule(m);
}
```


In mlir/python/mlir/dialects/transform/__init__.py

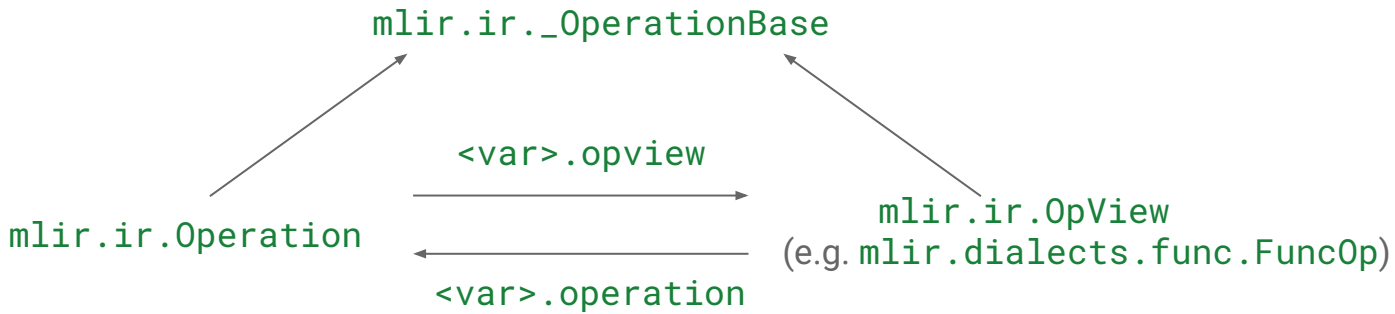
```
from ..._mlir_libs._mlirDialectsTransform import *
```

In mlir/python/CMakeLists.txt

```
declare_mlir_python_extension(
 MLIRPythonExtension.Dialects.Transform.Pybind
 MODULE_NAME _mlirDialectsTransform
 ADD_TO_PARENT MLIRPythonSources.Dialects.transform
 ROOT_DIR "${PYTHON_SOURCE_DIR}"
 SOURCES DialectTransform.cpp
 PRIVATE_LINK_LIBS LLVMSupport
 EMBED_CAPI_LINK_LIBS
   MLIRCAPIIR
   MLIRCAPITransform
)
```

Class Hierarchy

Types/Attributes (isinstance and constructor+assert)



Warning: `isinstance(<var>, func.FuncOp) != isinstance(<var>.opview, func.FuncOp)`

Generic APIs work on `ir.Operation` not specific `OpView` instances

Class Hierarchy

Regular class hierarchy (`isinstance` works), including custom dialect attributes and types

Mutating IR from Python

Something the bindings were not really designed for, except maybe for running passes.

Direct Mutation

```
operation.attributes["attr_name"] = ir.IntegerAttr.get(...)
```


Attributes are assignable

Direct Mutation

```
operation.attributes["attr_name"] = ir.IntegerAttr.get(...)
```

Attributes are assignable

```
operation.operands[0] = some_value
```

Operand list items are assignable

Direct Mutation

```
operation.attributes["attr_name"] = ir.IntegerAttr.get(...)
```

Attributes are assignable

```
operation.operands[0] = some_value
```

Operand list items are assignable

```
operation.operands[1].set_type(ir.SomeType.get(...))
```

Types can be updated

Direct Mutation

```
operation.attributes["attr_name"] = ir.IntegerAttr.get(...)
```

Attributes are assignable

```
operation.operands[0] = some_value
```

Operand list items are assignable

```
operation.operands[1].set_type(ir.SomeType.get(...))
```

Types can be updated

```
operation.regions.append(...)
operation.regions[0].blocks.append(...)
```

Linked lists can be appended to

Direct Mutation

```
operation.attributes["attr_name"] = ir.IntegerAttr.get(...)
```

Attributes are assignable

```
operation.operands[0] = some_value
```

Operand list items are assignable

```
operation.operands[1].set_type(ir.SomeType.get(...))
```

Types can be updated

```
operation.regions.append(...)
operation.regions[0].blocks.append(...)
```

Linked lists can be appended to

```
operation.erase()
```

Operations can be erased

Direct Mutation

```
operation.attributes["attr_name"] = ir.IntegerAttr.get(...)
```
Attributes are assignable

```
operation.operands[0] = some_value
```
Operand list items are assignable

```
operation.operands[1].set_type(ir.SomeType.get(...))
```
Types can be updated

```
operation.regions.append(...)
operation.regions[0].blocks.append(...)
```
Linked lists can be appended to

```
operation.erase()
```
Operations can be erased

That's all, folks!

Running Passes

```
from mlir import ir
from mlir.passmanager import PassManager


with ir.Context():
  pm = PassManager.parse("builtin.module(canonicalize,cse)")
  try:
    pm.run(operation)
  except MLIRError as e:
    # Do something...
    raise
```

Textual pass pipeline syntax 🤦

A failing pass will raise an exception

Running Passes

```
from mlir import ir
from mlir.passmanager import PassManager

handle = operation.regions[0].blocks[0].operations[0]
with ir.Context():
  pm = PassManager.parse("builtin.module(canonicalize,cse)")
  try:
    pm.run(operation)
  except MLIRError as e:
    # Do something...
    raise
print(handle) ## Will assert
```

Textual pass pipeline syntax 🤦

A failing pass will raise an exception

Warning: running a pass manager invalidates Python handles to operations nested under the root operation.

Registration

Not recommended: include all upstream passes.

- CMake dependency on `MLIRPythonExtension.RegisterEverything`
- Automatically registers when loading any package.

Recommended:

Create a new pybind11 package and register on initialization.

```
PYBIND11_MODULE(_mlirGPUPasses, m) {
  m.doc() = "MLIR GPU Dialect Passes";

  // Register all GPU passes on load.
  mlirRegisterGPUPasses();
}
```



```
import mlir.dialects.gpu.passes
```

Registration

Alternative

Create a new pybind11 package and register in a function.

```
PYBIND11_MODULE(_mlirGPUPasses, m) {
  m.doc() = "MLIR GPU Dialect Passes";

  m.def("register", [] {
    mlirRegisterGPUPasses();
  });
}
```



```
import mlir.dialects.gpu.passes
passes.register()
```

Recommended:

Create a new pybind11 package and register on initialization.

```
PYBIND11_MODULE(_mlirGPUPasses, m) {
  m.doc() = "MLIR GPU Dialect Passes";

  // Register all GPU passes on load.
  mlirRegisterGPUPasses();
}
```



```
import mlir.dialects.gpu.passes
```

Error Handling

```
def handler(diag: ir.Diagnostic):
 if diag.severity == ir.DiagnosticSeverity.ERROR:
   assert False, diag.message
 pass

context.attach_diagnostic_handler(handler)
```

Diagnostic handlers can be provided in Python

Where to find more information?

https://mlir.llvm.org/docs/CAPI/

https://mlir.llvm.org/docs/Bindings/Python/

Inside `mlir/lib/Bindings/Python/IRCore.cpp`

Inside `mlir/include/mlir-c/…h`