IREE Python API

Release snapshot

IREE Authors

Sep 23, 2023

API DOCS

CHAPTER

ONE

RUNTIME API

Module init for the python bindings.

class iree.runtime.**BufferCompatibility**

Members:

NONE

ALLOCATABLE

IMPORTABLE

EXPORTABLE

QUEUE_TRANSFER

QUEUE_DISPATCH

ALLOCATABLE = <BufferCompatibility.ALLOCATABLE: 1>

EXPORTABLE = <BufferCompatibility.EXPORTABLE: 4>

IMPORTABLE = <BufferCompatibility.IMPORTABLE: 2>

NONE = <BufferCompatibility.NONE: 0>

QUEUE_DISPATCH = <BufferCompatibility.QUEUE_DISPATCH: 2048>

QUEUE_TRANSFER = <BufferCompatibility.QUEUE_TRANSFER: 1024>

property name

property value

class iree.runtime.**BufferUsage**

Members:

NONE

TRANSFER_SOURCE

TRANSFER_TARGET

TRANSFER

DISPATCH_INDIRECT_PARAMS

DISPATCH_UNIFORM_READ

DISPATCH_STORAGE_READ

DISPATCH_STORAGE_WRITE

DISPATCH_STORAGE

DISPATCH_IMAGE_READ

DISPATCH_IMAGE_WRITE

DISPATCH_IMAGE

SHARING_EXPORT

SHARING_REPLICATE

SHARING_CONCURRENT

SHARING_IMMUTABLE

MAPPING_SCOPED

MAPPING_PERSISTENT

MAPPING_OPTIONAL

MAPPING_ACCESS_RANDOM

MAPPING_ACCESS_SEQUENTIAL_WRITE

MAPPING

DEFAULT

DEFAULT = <BufferUsage.DEFAULT: 3075>

DISPATCH_IMAGE = <BufferUsage.DISPATCH_IMAGE: 12288>

DISPATCH_IMAGE_READ = <BufferUsage.DISPATCH_IMAGE_READ: 4096>

DISPATCH_IMAGE_WRITE = <BufferUsage.DISPATCH_IMAGE_WRITE: 8192>

DISPATCH_INDIRECT_PARAMS = <BufferUsage.DISPATCH_INDIRECT_PARAMS: 256>

DISPATCH_STORAGE = <BufferUsage.DISPATCH_STORAGE: 3072>

DISPATCH_STORAGE_READ = <BufferUsage.DISPATCH_STORAGE_READ: 1024>

DISPATCH_STORAGE_WRITE = <BufferUsage.DISPATCH_STORAGE_WRITE: 2048>

DISPATCH_UNIFORM_READ = <BufferUsage.DISPATCH_UNIFORM_READ: 512>

MAPPING = <BufferUsage.MAPPING: 150994944>

MAPPING_ACCESS_RANDOM = <BufferUsage.MAPPING_ACCESS_RANDOM: 134217728>

MAPPING_ACCESS_SEQUENTIAL_WRITE = <BufferUsage.MAPPING_ACCESS_SEQUENTIAL_WRITE: 268435456>

MAPPING_OPTIONAL = <BufferUsage.MAPPING_OPTIONAL: 67108864>

MAPPING_PERSISTENT = <BufferUsage.MAPPING_PERSISTENT: 33554432>

MAPPING_SCOPED = <BufferUsage.MAPPING_SCOPED: 16777216>

NONE = <BufferUsage.NONE: 0>

SHARING_CONCURRENT = <BufferUsage.SHARING_CONCURRENT: 262144>

SHARING_EXPORT = <BufferUsage.SHARING_EXPORT: 65536>

SHARING_IMMUTABLE = <BufferUsage.SHARING_IMMUTABLE: 524288>

SHARING_REPLICATE = <BufferUsage.SHARING_REPLICATE: 131072>

TRANSFER = <BufferUsage.TRANSFER: 3>

TRANSFER_SOURCE = <BufferUsage.TRANSFER_SOURCE: 1>

TRANSFER_TARGET = <BufferUsage.TRANSFER_TARGET: 2>

property name

property value

class iree.runtime.**Config**(*driver_name: Optional[str] = None*, ***, *device: Optional[*iree._runtime.HalDevice*] = None*, *tracer: Optional[*iree.runtime.tracing.Tracer*] = None*)

System configuration.

default_vm_modules: Tuple[*iree._runtime.VmModule*, ...]

device: *iree._runtime.HalDevice*

tracer: Optional[*iree.runtime.tracing.Tracer*]

vm_instance: *iree._runtime.VmInstance*

class iree.runtime.**DeviceArray**(*device:* iree._runtime.HalDevice, *buffer_view:* iree._runtime.HalBufferView, *implicit_host_transfer: bool = False*, *override_dtype=None*)

An IREE device array.

Device arrays can be in one of two states:

1. Host accessible: The array will be backed by host accessible memory and can have the usual things done with it that one expects to be able to do with an ndarray.
2. Device resident: The array is just a handle to a device resident Buffer (and BufferView wrapper). Metadata about the array are accessible (shape and dtype) but anything that touches the data cannot be accessed in this state.

How a device array comes into existence controls how it can transition between these states:

- A user can create a DeviceArray explicitly with a device allocator. Such an array will not be implicitly convertible to host accessible, although accessors exist to do so.
- When created by the platform with a synchronization policy, then implicit transfer back to the host will trigger appropriate waits and be performed automatically (this is the common case for function return values if not otherwise configured, as an example).

astype(*dtype*, *casting='unsafe'*, *copy=True*)

property dtype

property is_host_accessible

Whether this array is currently host accessible.

reshape(**args*)

property shape

to_host() → numpy.ndarray

class iree.runtime.**FunctionInvoker**(*vm_context:* iree._runtime.VmContext, *device:* iree._runtime.HalDevice, *vm_function:* iree._runtime.VmFunction, *tracer: Optional[iree.runtime.tracing.ContextTracer]*)

Wraps a VmFunction, enabling invocations against it.

property vm_function: *iree._runtime.VmFunction*

class iree.runtime.**HalAllocator**

allocate_buffer(*self:* iree._runtime.HalAllocator, *memory_type: int*, *allowed_usage: int*, *allocation_size: int*) → iree::python::HalBuffer

Allocates a new buffer with requested characteristics (does not initialize with specific data).

allocate_buffer_copy(*self:* iree._runtime.HalAllocator, *memory_type: int*, *allowed_usage: int*, *buffer: object*, *element_type: Optional[*iree._runtime.HalElementType*] = None*) → object

Allocates a new buffer and initializes it from a Python buffer object. If an element type is specified, wraps in a BufferView matching the characteristics of the Python buffer. The format is requested as ND/C-Contiguous, which may incur copies if not already in that format.

property formatted_statistics

property has_statistics

query_buffer_compatibility(*self:* iree._runtime.HalAllocator, *memory_type: int*, *allowed_usage: int*, *intended_usage: int*, *allocation_size: int*) → int

property statistics

trim(*self:* iree._runtime.HalAllocator) → None

class iree.runtime.**HalBuffer**

create_view(*self: iree._runtime.HalBuffer*, *shape: iree::python::HalShape*, *element_size: int*) → iree::python::HalBufferView

fill_zero(*self:* iree._runtime.HalBuffer, *byte_offset: int*, *byte_length: int*) → None

class iree.runtime.**HalBufferView**

property element_type

map(*self:* iree._runtime.HalBufferView) → iree::python::HalMappedMemory

property ref

property shape

class iree.runtime.**HalDevice**

property allocator

begin_profiling(*self:* iree._runtime.HalDevice, ***kwargs*) → None

end_profiling(*self:* iree._runtime.HalDevice) → None

class iree.runtime.**HalDriver**

create_default_device(*self:* iree._runtime.HalDriver, ***kwargs*) → *iree._runtime.HalDevice*

create_device(**args*, ***kwargs*)

Overloaded function.

1. create_device(self: iree._runtime.HalDriver, arg0: int, **kwargs) -> iree._runtime.HalDevice
2. create_device(self: iree._runtime.HalDriver, arg0: dict, **kwargs) -> iree._runtime.HalDevice

create_device_by_uri(*self:* iree._runtime.HalDriver, *arg0: str*, ***kwargs*) → *iree._runtime.HalDevice*

static query() → List[str]

query_available_devices(*self:* iree._runtime.HalDriver) → list

class iree.runtime.HalElementType

Members:

NONE

OPAQUE_8

OPAQUE_16

OPAQUE_32

OPAQUE_64

BOOL_8

INT_4

INT_8

INT_16

INT_32

INT_64

SINT_4

SINT_8

SINT_16

SINT_32

SINT_64

UINT_4

UINT_8

UINT_16

UINT_32

UINT_64

FLOAT_16

FLOAT_32

FLOAT_64

BFLOAT_16

COMPLEX_64

COMPLEX_128

BFLOAT_16 = <HalElementType.BFLOAT_16: 570425360>

BOOL_8 = <HalElementType.BOOL_8: 318767112>

COMPLEX_128 = <HalElementType.COMPLEX_128: 587202688>

COMPLEX_64 = <HalElementType.COMPLEX_64: 587202624>

FLOAT_16 = <HalElementType.FLOAT_16: 553648144>

FLOAT_32 = <HalElementType.FLOAT_32: 553648160>

FLOAT_64 = <HalElementType.FLOAT_64: 553648192>

INT_16 = <HalElementType.INT_16: 268435472>

INT_32 = <HalElementType.INT_32: 268435488>

INT_4 = <HalElementType.INT_4: 268435460>

INT_64 = <HalElementType.INT_64: 268435520>

INT_8 = <HalElementType.INT_8: 268435464>

NONE = <HalElementType.NONE: 0>

OPAQUE_16 = <HalElementType.OPAQUE_16: 16>

OPAQUE_32 = <HalElementType.OPAQUE_32: 32>

OPAQUE_64 = <HalElementType.OPAQUE_64: 64>

OPAQUE_8 = <HalElementType.OPAQUE_8: 8>

SINT_16 = <HalElementType.SINT_16: 285212688>

SINT_32 = <HalElementType.SINT_32: 285212704>

SINT_4 = <HalElementType.SINT_4: 285212676>

SINT_64 = <HalElementType.SINT_64: 285212736>

SINT_8 = <HalElementType.SINT_8: 285212680>

UINT_16 = <HalElementType.UINT_16: 301989904>

UINT_32 = <HalElementType.UINT_32: 301989920>

UINT_4 = <HalElementType.UINT_4: 301989892>

UINT_64 = <HalElementType.UINT_64: 301989952>

UINT_8 = <HalElementType.UINT_8: 301989896>

static map_to_dtype(*arg0: int*) → object

property name

property value

class iree.runtime.Linkage

Members:

INTERNAL

IMPORT

IMPORT_OPTIONAL

EXPORT

EXPORT = <Linkage.EXPORT: 2>

IMPORT = <Linkage.IMPORT: 1>

IMPORT_OPTIONAL = <Linkage.IMPORT_OPTIONAL: 3>

INTERNAL = <Linkage.INTERNAL: 0>

property name

property value

class iree.runtime.**MemoryAccess**

Members:

NONE

READ

WRITE

DISCARD

DISCARD_WRITE

ALL

ALL = <MemoryAccess.ALL: 7>

DISCARD = <MemoryAccess.DISCARD: 4>

DISCARD_WRITE = <MemoryAccess.DISCARD_WRITE: 6>

NONE = <MemoryAccess.NONE: 0>

READ = <MemoryAccess.READ: 1>

WRITE = <MemoryAccess.WRITE: 2>

property name

property value

class iree.runtime.**MemoryType**

Members:

NONE

OPTIMAL

HOST_VISIBLE

HOST_COHERENT

HOST_CACHED

HOST_LOCAL

DEVICE_VISIBLE

DEVICE_LOCAL

DEVICE_LOCAL = <MemoryType.DEVICE_LOCAL: 48>

DEVICE_VISIBLE = <MemoryType.DEVICE_VISIBLE: 16>

HOST_CACHED = <MemoryType.HOST_CACHED: 8>

HOST_COHERENT = <MemoryType.HOST_COHERENT: 4>

HOST_LOCAL = <MemoryType.HOST_LOCAL: 70>

HOST_VISIBLE = <MemoryType.HOST_VISIBLE: 2>

NONE = <MemoryType.NONE: 0>

OPTIMAL = <MemoryType.OPTIMAL: 1>

property name

property value

class iree.runtime.**PyModuleInterface**

create(*self:* iree._runtime.PyModuleInterface) → iree::python::VmModule

property destroyed

export(*self:* iree._runtime.PyModuleInterface, *name: str*, *cconv: str*, *callable: object*) → None

property initialized

class iree.runtime.**Shape**

class iree.runtime.**SystemContext**(*vm_modules=None*, *config: Optional[*iree.runtime.system_api.Config*] = None*)

Global system.

add_vm_module(*vm_module*)

add_vm_modules(*vm_modules*)

property config: *iree.runtime.system_api.Config*

property instance: *iree._runtime.VmInstance*

property is_dynamic: bool

property modules: iree.runtime.system_api.BoundModules

property vm_context: *iree._runtime.VmContext*

class iree.runtime.**Tracer**(*trace_path: str*)

Object for tracing calls made into the runtime.

get_unique_name(*local_name: str*) → str

persist_vm_module(*vm_module:* iree._runtime.VmModule) → iree.runtime.tracing.TracedModule

class iree.runtime.**VmBuffer**

property ref

class iree.runtime.**VmContext**

property context_id

invoke(*self:* iree._runtime.VmContext, *arg0:* iree._runtime.VmFunction, *arg1:* iree._runtime.VmVariantList, *arg2:* iree._runtime.VmVariantList) → None

register_modules(*self: iree._runtime.VmContext, arg0: List[iree::python::VmModule]*) → None

class iree.runtime.**VmFunction**

property linkage

property module_name

property name

property ordinal

property reflection

class iree.runtime.**VmInstance**

class iree.runtime.**VmModule**

static from_flatbuffer(*arg0:* iree._runtime.VmInstance, *arg1: object*) → *iree._runtime.VmModule*

property function_names

lookup_function(*self: iree._runtime.VmModule*, *name: str*, *linkage: iree._runtime.Linkage* = *<Linkage.EXPORT: 2>*) → Optional[*iree._runtime.VmFunction*]

property name

property stashed_flatbuffer_blob

property version

class iree.runtime.**VmVariantList**

get_as_list(*self:* iree._runtime.VmVariantList, *arg0: int*) → object

get_as_object(*self:* iree._runtime.VmVariantList, *arg0: int*, *arg1: object*) → object

get_as_ref(*self:* iree._runtime.VmVariantList, *arg0: int*) → object

get_serialized_trace_value(*self:* iree._runtime.VmVariantList, *arg0: int*) → object

get_variant(*self:* iree._runtime.VmVariantList, *arg0: int*) → object

push_float(*self:* iree._runtime.VmVariantList, *arg0: float*) → None

push_int(*self:* iree._runtime.VmVariantList, *arg0: int*) → None

push_list(*self:* iree._runtime.VmVariantList, *arg0:* iree._runtime.VmVariantList) → None

push_ref(*self:* iree._runtime.VmVariantList, *arg0: handle*) → None

property ref

property size

iree.runtime.**asdevicearray**(*device: iree._runtime.HalDevice*, *a*, *dtype=None*, ***, *implicit_host_transfer: bool* = *False*, *memory_type=<MemoryType.DEVICE_LOCAL: 48>*, *allowed_usage=<BufferUsage.???: 150998019>*, *element_type: Optional[iree._runtime.HalElementType]* = *None*) → *iree.runtime.array_interop.DeviceArray*

Helper to create a DeviceArray from an arbitrary array like.

This is similar in purpose and usage to np.asarray, except that it takes a device as the first argument. This may not be the best mechanism for getting a DeviceArray, depending on your use case, but it is reliable and simple. This function may make a defensive copy or cause implicit transfers to satisfy the request. If this is important to you, then a lower level API is likely more appropriate.

Note that additional flags *memory_type*, *allowed_usage* and *element_type* are only hints if creating a new DeviceArray. If *a* is already a DeviceArray, they are ignored.

iree.runtime.**benchmark_exe**()

iree.runtime.**benchmark_module**(*module*, *entry_functiong=None*, *inputs=[]*, ***kwargs*)

iree.runtime.**create_hal_module**(*arg0: iree::python::VmInstance*, *arg1: iree::python::HalDevice*) → iree::python::VmModule

iree.runtime.**get_default_tracer**() → Optional[*iree.runtime.tracing.Tracer*]

Gets a default call tracer based on environment variables.

iree.runtime.**get_device**(*device_uri: str*, *cache: bool = True*) → *iree._runtime.HalDevice*
Gets a cached device by URI.

Parameters

- **device_uri** – The URI of the device, either just a driver name for the default or a fully qualified "driver://path?params".
- **cache** – Whether to cache the device (default True).

Returns A HalDevice.

iree.runtime.**get_driver**(*device_uri: str*) → *iree._runtime.HalDriver*
Returns a HAL driver by device_uri (or driver name).

iree.runtime.**get_first_device**(*device_uris: Optional[Sequence[str]] = None*, *cache: bool = True*) → *iree._runtime.HalDevice*
Gets the first valid (cached) device for a prioritized list of names.

If no driver_names are given, and an environment variable of IREE_DEFAULT_DEVICE is available, then it is treated as a comma delimitted list of driver names to try.

This is meant to be used for default/automagic startup and is not suitable for any kind of multi-device setup.

Parameters

- **device_uris** – Explicit list of device URIs to try.
- **cache** – Whether to cache the device (default True).

Returns A HalDevice instance.

iree.runtime.**load_vm_flatbuffer**(*vm_flatbuffer: bytes*, ***, *driver: Optional[str] = None*, *backend: Optional[str] = None*) → iree.runtime.system_api.BoundModule
Loads a VM Flatbuffer into a callable module.

Either 'driver' or 'backend' must be specified.

iree.runtime.**load_vm_flatbuffer_file**(*path: str*, ***, *driver: Optional[str] = None*, *backend: Optional[str] = None*) → iree.runtime.system_api.BoundModule
Loads a file containing a VM Flatbuffer into a callable module.

Either 'driver' or 'backend' must be specified.

iree.runtime.**load_vm_module**(*vm_module*, *config: Optional[*iree.runtime.system_api.Config*] = None*)
Loads a VmModule into a new SystemContext and returns it.

iree.runtime.**load_vm_modules**(**vm_modules*, *config: Optional[*iree.runtime.system_api.Config*] = None*)
Loads VmModules into a new SystemContext and returns them.

iree.runtime.**normalize_value**(*value: Any*) → Optional[Union[numpy.ndarray, List[Any], Tuple[Any]]]
Normalizes the given value for input to (or comparison with) IREE.

iree.runtime.**query_available_drivers**() → Collection[str]
Returns a collection of driver names that are available.

CHAPTER

TWO

COMPILER API

2.1 In-Process Compiler API

IREE provides access to its MLIR-based compiler via a dedicated set of APIs presented here.

2.2 Invoking Command Line Tools

As with many compilers, IREE's compiler consists of many command line tools, some of which are designed for compiler devs and are only accessible via source builds. User level tools are distributed via the Python packages and are also accessible via dedicated Python APIs, documented here.

2.2.1 Core Compiler (*iree-compile*)

This module contains Python wrappers for various IREE command-line tools.

This top-level API provides access to the *iree-compiler* tool, which compiles MLIR ASM via IREE's compiler to a supported output format (i.e. VM FlatBuffer, C source code, etc).

Example

```
import iree.compiler.tools

SIMPLE_MUL_ASM = """
func.func @simple_mul(%arg0: tensor<4xf32>, %arg1: tensor<4xf32>) -> tensor<4xf32> {
    %0 = "tosa.mul"(%arg0, %arg1) {shift = 0 : i32} : (tensor<4xf32>, tensor<4xf32>) ->␣
↪tensor<4xf32>
    return %0 : tensor<4xf32>
}
"""

# Also see compile_file()
# There are many keyword options available.
# See iree.compiler.CompilerOptions
binary = iree.compiler.tools.compile_str(
    SIMPLE_MUL_ASM, input_type="tosa", target_backends=["llvm-cpu"])
```

iree.compiler.tools.**compile_file**(*input_file: str*, ***kwargs*)

Invokes the IREE compiler on an input file.

Parameters

- **input_file** – File containing MLIR assembly to compile.
- ****kwargs** – Keyword arguments corresponding to CompilerOptions.

Returns Either a byte buffer of the compiled content or None if output_file was specified in the options.

iree.compiler.tools.**compile_str**(*input_str: Union[str, bytes]*, ***kwargs*)

Invokes the IREE compiler with an input string.

Parameters

- **input_str** – MLIR assembly to parse/compile (str or bytes).
- ****kwargs** – Keyword arguments corresponding to CompilerOptions.

Returns Either a byte buffer of the compiled content or None if output_file was specified in the options.

class iree.compiler.tools.**CompilerOptions**(*output_file: Optional[str] = None*, *target_backends: Sequence[str] = ()*, *input_type: Union[iree.compiler.tools.core.InputType, str] = InputType.NONE*, *output_format: Union[iree.compiler.tools.core.OutputFormat, str] = OutputFormat.FLATBUFFER_BINARY*, *extra_args: Sequence[str] = ()*, *optimize: bool = True*, *output_mlir_debuginfo: bool = True*, *output_generic_mlir: bool = False*, *extended_diagnostics: bool = False*, *strip_debug_ops: bool = False*, *strip_source_map: bool = False*, *crash_reproducer_path: Optional[str] = None*, *enable_tflite_bindings: bool = False*, *enable_benchmark: bool = False*)

Options to the compiler backend.

Parameters

- **output_file** – Optionally save the compiled binary to a file instead of returning it.
- **target_backends** – List of str names of target backends to compile into the binary. The resulting binary will run on targets that match one or more of the compiled backends.
- **input_type** – The type of input legalization to perform prior to full compilation. Values can either be an *InputType* enum value or a case-insensitive name. Defaults to *InputType.NONE*.
- **output_format** – Override the output format. See the *OutputFormat* enum. Values can either be an enum value or a case-insensitive name of the option. Typically used for debugging Defaults to *OutputFormat.FLATBUFFER_BINARY*.
- **extra_args** – Optional list of additional arguments to pass to the compiler. Example: [“–mlir-print-ir-after-all”, “–some-other-arg”]. Individual arguments must be separate items in the list.
- **optimize** – Whether to apply some default high level optimizations (default True).
- **output_mlir_debuginfo** – Include debuginfo (including paths) in any saved or returned MLIR.
- **output_generic_mlir** – Use the generic (and more portable) MLIR formatting for any saved or returned MLIR instead of the per-dialect custom assembly.

- **extended_diagnostics** – Outputs extended information on diagnostics, potentially outputting very verbosely (defaults to False).
- **strip_debug_ops** – Whether to strip high level operations used to aid debugging.
- **strip_source_map** – Whether to strip source map information (used to generate better errors).
- **crash_reproducer_path** – File name to output an MLIR crash dump to if there is a compiler failure.
- **enable_tflite_bindings** – Support the IREE TFLite runtime bindings API shim.
- **enable_benchmark** – Whether to generate instrumented binaries suitable for benchmarking.

enum iree.compiler.tools.**InputType**(*value*)
Enumeration of allowable input types to the compiler.

An instance of this enum or the string form can be passed to *CompilerOptions.input_type*.

Valid values are as follows:

NONE

MHLO

TOSA

TM_TENSOR

XLA

enum iree.compiler.tools.**OutputFormat**(*value*)
The output format of the compiler.

Valid values are as follows:

FLATBUFFER_BINARY

FLATBUFFER_TEXT

MLIR_TEXT

2.2.2 Debugging

A number of optional arguments to the compiler can be useful for debugging:

- *extended_diagnostics=True* - Outputs verbose attached operations to diagnostics. Can output a large volume of information.
- *crash_reproducer_path=... some .mlir file path...* - On a crash or error,a reproducer will be output at the listed path.
- *extra_args=[...]* - Passes extra arguments to the compiler. Useful for various standard features of MLIR based compilers like *-mlir-print-ir-after-all*.

In addition, the core compiler and frontend compiler APIs have a unified mechanism for saving their temporary files, which are often useful for post mortem debugging. Since the need for this is often as part of a larger system, it is exposed both via an environment variable and an API.

In order to save all temporaries and reproducers, set the *IREE_SAVE_TEMPS* environment variable to a directory in which to dump artifacts. For complex programs that invoke the compiler many times, it will typically be necessary to further qualify the path, and there are a few placeholders that will be expanded:

- *{id}* - A per-process monotonically increasing number for each compiler invocation. Can be overridden by the API if a better symbolic name is available (i.e. test case, etc).
- *{pid}* - Process ID of the current process.
- *{main}* - Basename of *sys.argv[0]*, which is typically the name of the Python main file.

For interactive use, the following (on a Unix-like system) should provide value:

```
export IREE_SAVE_TEMPS="/tmp/ireedumps/{main}/{id}"
```

For the context manager based API, refer to the *iree.compiler.tools.debugging.TempFileSaver* class.

class iree.compiler.tools.debugging.**TempFileSaver**(*temp_path_pattern: Optional[str] = None*, *, *invocation_id: Optional[str] = None*)

Manages the saving of temp files resulting from tool invocations.

The TempFileSaver is a thread-local context bound object. An attempt to create a new one will return the most recent instance created and entered as a context manager. This allows up-stack callers to establish the policy for saving temporaries and deep implementations will inherit it.

Proper usage from users wishing to establish a saver context:

```
with TempFileSaver():
  # Do things with temp files.
```

Proper usage for implementors wishing to use an established saver context or set up a new one:

```
with TempFileSaver.implicit() as tfs:
  # Do things with temp files.
```

The outer-most creator can customize it with explicit arguments to __init__ but these will be ignored if an instance is already thread bound.

TEMP_PATH_ENV_KEY = 'IREE_SAVE_TEMPS'

alloc_optional(*file_name: str*, *, *export_as: Optional[str] = None*) → Optional[str]

Allocates an optional temporary file.

When in non-retained mode, the return value is 'export_as', meaning that the file is just some user specified output file.

When in retained mode, the output file will be an index-mangled variant of 'file_name' under the temp_path. In addition, a mapping will be added so that upon finalization, the file is also exported to 'export_as' if specified.

Returns None if neither a user-specified 'export_as' is specified nor in retained mode.

The distinction between retained temporaries and exports is to help in cases for when the caller has requested that an artifact be written to a specific place (i.e. an output file) but for debuggability, we also want to save it as a temporary. In this case, we save it to the temporary location and then conclude by moving artifacts to their final location once the saver goes out of scope.

static current()

static implicit()

2.2.3 TFLite Importer (*iree-import-tflite*)

Using the API below to access *iree-import-tflite* presumes that the tool itself is installed via the appropriate PIP package.

Imports TFLite binaries via the *iree-import-tflite* tool.

iree.compiler.tools.tflite.**compile_file**(*fb_path: str*, ***kwargs*)
Compiles a TFLite FlatBuffer file to an IREE binary.

Parameters

- **fb_path** – Path to the FlatBuffer.
- ****kwargs** – Keyword args corresponding to ImportOptions or CompilerOptions.

Returns A bytes-like object with the compiled output or None if output_file= was specified.

iree.compiler.tools.tflite.**compile_str**(*input_bytes: bytes*, ***kwargs*)
Compiles in-memory TFLite FlatBuffer to an IREE binary.

Parameters

- **input_bytes** – Flatbuffer content as bytes or IR string.
- ****kwargs** – Keyword args corresponding to ImportOptions or CompilerOptions.

Returns A bytes-like object with the compiled output or None if output_file= was specified.

class iree.compiler.tools.tflite.**ImportOptions**(*output_file: Optional[str] = None*, *target_backends: Sequence[str] = ()*, *input_type: Optional[str] = 'tosa'*, *output_format: Union[OutputFormat, str] = OutputFormat.FLATBUFFER_BINARY*, *extra_args: Sequence[str] = ()*, *optimize: bool = True*, *output_mlir_debuginfo: bool = True*, *output_generic_mlir: bool = False*, *extended_diagnostics: bool = False*, *strip_debug_ops: bool = False*, *strip_source_map: bool = False*, *crash_reproducer_path: Optional[str] = None*, *enable_tflite_bindings: bool = False*, *enable_benchmark: bool = False*, *input_arrays: Sequence[str] = ()*, *output_arrays: Sequence[str] = ()*, *import_only: bool = False*, *import_extra_args: Sequence[str] = ()*, *save_temp_tfl_input: Optional[str] = None*, *save_temp_iree_input: Optional[str] = None*)
Import options layer on top of the backend compiler options.

Parameters

- **input_arrays** – Sequence of input array node names (if different from default).
- **output_arrays** – Sequence of output array node names (if different from default).
- **import_only** – Only import the module. If True, the result will be textual MLIR that can be further fed to the IREE compiler. If False (default), the result will be the fully compiled IREE binary. In both cases, bytes-like output is returned. Note that if the output_file= is specified and import_only=True, then the MLIR form will be written to the output file.
- **import_extra_args** – Extra arguments to pass to the iree-import-tf tool.
- **save_temp_tfl_input** – Optionally save the IR that results from importing the flatbuffer (prior to any further transformations).

- **save_temp_iree_input** – Optionally save the IR that is the result of the import (ready to be passed to IREE).

2.2.4 TensorFlow Importer (*iree-import-tf*)

Using the API below to access *iree-import-tf* presumes that the tool itself is installed via the appropriate PIP package.

Imports TensorFlow artifacts via the `iree-import-tf tool.

iree.compiler.tools.tf.**compile_module**(*module*, *saved_model_dir: Optional[str] = None*, ***kwargs*)
Compiles a tf.Module to an IREE binary (by saving to disk).

Parameters

- **module** – The tf.Module instance to convert to MLIR
- **saved_model_dir** – Optional path to save the tf.Module to. The module will not be persisted on disk outside of this call if this is not provided.
- ****kwargs** – Keyword args corresponding to ImportOptions or CompilerOptions.

Returns Same as compile_saved_model().

iree.compiler.tools.tf.**compile_saved_model**(*saved_model_dir: str*, ***kwargs*)
Compiles an on-disk saved model to an IREE binary.

Parameters

- **saved_model_dir** – Path to directory where the model was saved.
- ****kwargs** – Keyword args corresponding to ImportOptions or CompilerOptions.

Returns A bytes-like object with the compiled output or None if output_file= was specified.

class iree.compiler.tools.tf.**ImportOptions**(*output_file: Optional[str] = None*, *target_backends: Sequence[str] = ()*, *input_type: Union[InputType, str] = InputType.XLA*, *output_format: Union[OutputFormat, str] = OutputFormat.FLATBUFFER_BINARY*, *extra_args: Sequence[str] = ()*, *optimize: bool = True*, *output_mlir_debuginfo: bool = True*, *output_generic_mlir: bool = False*, *extended_diagnostics: bool = False*, *strip_debug_ops: bool = False*, *strip_source_map: bool = False*, *crash_reproducer_path: Optional[str] = None*, *enable_tflite_bindings: bool = False*, *enable_benchmark: bool = False*, *exported_names: Sequence[str] = ()*, *import_only: bool = False*, *import_type: Union[ImportType, str] = ImportType.OBJECT_GRAPH*, *saved_model_tags: Set[str] = <factory>*, *import_extra_args: Sequence[str] = ()*, *save_temp_tf_input: Optional[str] = None*, *save_temp_mid_level_input: Optional[str] = None*, *save_temp_iree_input: Optional[str] = None*, *use_tosa: bool = False*)
Import options layer on top of the backend compiler options.

Parameters

- **exported_names** – Optional sequence representing the exported names to keep (object graph/v2 models only).
- **import_only** – Only import the module. If True, the result will be textual MLIR that can be further fed to the IREE compiler. If False (default), the result will be the fully compiled

IREE binary. In both cases, bytes-like output is returned. Note that if the output_file= is specified and import_only=True, then the MLIR form will be written to the output file.

- **import_type** – Type of import to perform. See ImportType enum.
- **saved_model_tags** – Set of tags to export (signature def/v1 saved models only).
- **import_extra_args** – Extra arguments to pass to the iree-import-tf tool.
- **save_temp_tf_input** – Optionally save the IR that is input to the TensorFlow pipeline.
- **save_temp_mid_level_input** – Optionally save the IR that is input to the mid level IR.
- **save_temp_iree_input** – Optionally save the IR that is the result of the import (ready to be passed to IREE).

enum iree.compiler.tools.tf.**ImportType**(*value*)

Import type of the model.

Valid values are as follows:

OBJECT_GRAPH

SIGNATURE_DEF

2.3 IREE Dialect

This dialect contains frontend-oriented ops and types which are intended to be used as input to IREE's compiler (in addition to various MLIR core dialects).

2.4 IREE Python DataModel Dialect

This dialect defines a data model for representing a Python program, suitable for compilation to IREE and for the construction of Python-based DSLs.

2.4.1 Importer

2.4.2 Operations and Types

2.5 MLIR Core Dialects

2.5.1 *builtin* dialect

class iree.compiler.dialects.builtin.**ModuleOp**(*, *loc=None*, *ip=None*)

class iree.compiler.dialects.builtin.**UnrealizedConversionCastOp**(*outputs*, *inputs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'builtin.unrealized_conversion_cast'

property inputs

property outputs

2.5.2 *linalg* dialect

class iree.compiler.dialects.linalg.**AffineBuildState**(*, *global_state: Optional[*iree.compiler.dialects.linalg.opdsl.lang.affine.AffineBuildSt = *None*, *allow_new_symbols: bool = True*, *allow_new_dims: bool = True*)

Internal state for the AffineExprDef._create impls.

Note that a "local" AffineBuildState can be created relative to a "global" AffineBuildState. In that case, any affine expressions built will inherit symbol and dim bindings from the global state and will update both as new ones are discovered. This allows for building expressions across contexts which share a common symbol and dim space.

property dim_count: int

get_dim(*dimname: str*) → int
Gets the dim position given a name.

get_symbol(*symname: str*) → int
Geta a symbol position given a name.

property local_dim_count: int

property local_symbol_count: int

property symbol_count: int

class iree.compiler.dialects.linalg.**AffineExprDef**
Base class for an affine expression being defined.

build(*state: Optional[*iree.compiler.dialects.linalg.opdsl.lang.affine.AffineBuildState*] = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineExpr*
Builds the corresponding _ir.AffineExpr from the definitions.

static coerce_from(*py_value*)

visit_affine_exprs(*callback*)
Visits all AffineExprDefs including self.

class iree.compiler.dialects.linalg.**BatchMatmulOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**BatchMatmulTransposeBOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**BatchMatvecOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**BatchReduceMatmulOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**BinaryFn**
Binary function namespace.

As the integer types are signless, signedness is implement by different functions that treat integers as signed or unsigned values.

Examples: - max -> *arith.MaxSIOp* - max_unsinged -> *arith.MaxUIOp*

add = add

max_signed = max_signed

max_unsigned = max_unsigned

min_signed = min_signed

min_unsigned = min_unsigned

mul = mul

sub = sub

class iree.compiler.dialects.linalg.**BinaryFnAttrDef**(*default:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.BinaryFnTy

Binary function attribute definition.

Binary function attributes provide a way to make the arithmetic computation parametrizable. Every attribute specifies a default binary function that may be overwritten at operation instantiation time.

class iree.compiler.dialects.linalg.**BinaryFnType**(*fn_name: str*)

Binary function.

A binary function takes two tensor expressions and returns the function evaluation result.

class iree.compiler.dialects.linalg.**BroadcastOp**

OPERATION_NAME = 'linalg.broadcast'

property init

property input

property region

property result

Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.linalg.**Comprehension**(**bindings: Tuple[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorUse iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorExpress

Represents a single comprehension.

property all_reduction_dims:
Set[Tuple[*iree.compiler.dialects.linalg.opdsl.lang.affine.DimDef*, ...]]

Gets the reduction dims for the comprehension or None.

class iree.compiler.dialects.linalg.**Conv1DNcwFcwOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv1DNwcWcfOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv1DOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv2DNchwFchwOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv2DNgchwFgchwOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv2DNhwcFhwcOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv2DNhwcHwcfOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv2DNhwcHwcfQOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv2DOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv3DNdhwcDhwcfOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv3DNdhwcDhwcfQOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Conv3DOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**CopyOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DefinedOpCallable**(*op_name: str*, *op_def:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.LinalgOp

Callable that wraps any defined op function.

class iree.compiler.dialects.linalg.**DepthwiseConv1DNwcWcOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv1DNwcWcmOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv2DNchwChwOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv2DNhwcHwcOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv2DNhwcHwcQOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv2DNhwcHwcmOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv2DNhwcHwcmQOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv3DNdhwcDhwcOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DepthwiseConv3DNdhwcDhwcmOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**DimDef**(*dimname: str*)

Represents a named dimension.

ALL_DIMS = {'__delete__': Dim(__delete__), '__get__': Dim(__get__), '__qualname__': Dim(__qualname__), '__set__': Dim(__set__), '__sphinx_mock__': Dim(__sphinx_mock__), 'b': Dim(b), 'c': Dim(c), 'cm': Dim(cm), 'f': Dim(f), 'fg': Dim(fg), 'g': Dim(g), 'ic': Dim(ic), 'k': Dim(k), 'k0': Dim(k0), 'kd': Dim(kd), 'kh': Dim(kh), 'kw': Dim(kw), 'm': Dim(m), 'm0': Dim(m0), 'n': Dim(n), 'n0': Dim(n0), 'od': Dim(od), 'oh': Dim(oh), 'ow': Dim(ow)}

classmethod create_expando()

Create an expando class that creates unique symbols based on attr access.

class iree.compiler.dialects.linalg.**DotOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**ElemwiseBinaryOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**ElemwiseUnaryOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

enum iree.compiler.dialects.linalg.**Enum**(*value*)
Generic enumeration.

Derive from this class to define new enumerations.

The *Enum* and its members have the following methods:

name
The name of the Enum member.

value
The value of the Enum member.

class iree.compiler.dialects.linalg.**FillOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**FillRng2DOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

enum iree.compiler.dialects.linalg.**FunctionKind**(*value*)
An enumeration.

Valid values are as follows:

UNARY

BINARY

TYPE

class iree.compiler.dialects.linalg.**GenericOp**(*result_tensors*, *inputs*, *outputs*, *indexing_maps*, *iterator_types*, ***, *doc=None*, *library_call=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'linalg.generic'

property doc

property inputs

property library_call

property outputs

property region

property result_tensors

class iree.compiler.dialects.linalg.**IndexAttrDef**(**sizes:* iree.compiler.dialects.linalg.opdsl.lang.affine.SymbolDef, *default: Sequence[int]*)
Index attribute definition.

Index attributes provide a way to define and set symbols that can be used in indexing expressions. Every attribute specifies a tuple of symbols that at compile-time are replaced by integer values as well as their default values.

class iree.compiler.dialects.linalg.**IndexOp**(*dim*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'linalg.index'

property dim

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.linalg.**LinalgOpConfig**(*metadata:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OpMetadataD **, structured_op: Optional[*iree.compiler.dialects.linalg.opdsl.lang.config.LinalgStructure *= None*)

Container for any supported linalg op type.

This includes the concrete type by name for ease of parsing by systems that ignore tags.

static from_linalg_op_def(*op_def: iree.compiler.dialects.linalg.opdsl.lang.comprehension.LinalgOpDef*, *context: Optional[iree.compiler._mlir_libs._site_initialize.<locals>.Context] = None*) → Sequence[*iree.compiler.dialects.linalg.opdsl.lang.config.LinalgOpConfig*]

Expands a LinalgOpDef into corresponding Linalg configured ops.

to_yaml_custom_dict()

yaml_tag = '!LinalgOpConfig'

class iree.compiler.dialects.linalg.**LinalgOpDef**(*name: str*, *cpp_class_name: Optional[str] = None*, *doc: Optional[str] = None*)

Definition of a linalg op.

add_operand(*name: str*, *operand:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandDef)

Registers an operand.

class iree.compiler.dialects.linalg.**LinalgStructuredOpConfig**(*comprehension: iree.compiler.dialects.linalg.opdsl.lang.comprehension.* *domain: Sequence[iree.compiler.dialects.linalg.opdsl.lang.affine.D* *registered_operands: Sequence[iree.compiler.dialects.linalg.opdsl.lang.compre* *context: Optional[iree.compiler._mlir_libs._site_initialize.<locals>* *= None*)

Configuration for metadata sufficient to construct a linalg named op.

add_indexed_operand(*operand_def:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandDef)

add_operand(*operand_def:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandDef)

add_tensor_use(*tensor_use:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorUse)

property indexing_maps: Sequence[*iree.compiler._mlir_libs._mlir.ir.AffineMap*]

property iterator_types: Sequence[str]

property ordered_dims: Sequence[Tuple[str, int]]

Gets the ordered list of dim bindings (symbolic name, position).

TODO: The original parser relies on parse ordering to arrive at the iterator types, but that ordering is not defined on the Python side, so this may be ambiguous.

property ordered_operands: Sequence[*iree.compiler.dialects.linalg.opdsl.lang.config.OperandDefConfig*]

to_yaml_custom_dict()

yaml_tag = '!LinalgStructuredOpConfig'

class iree.compiler.dialects.linalg.**MapOp**(*result*, *inputs*, *init*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'linalg.map'

property init

property inputs

property mapper

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.linalg.**MatmulOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**MatmulTransposeBOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**MatmulUnsignedOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**MatvecOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**Mmt4DOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**OpDefinitionDef**(*def_name: str*)
A method that an op implements.

class iree.compiler.dialects.linalg.**OpInterfaceDef**(*cpp_name: str*)
An interface that an op implements.

class iree.compiler.dialects.linalg.**OpMetadataDef**(*name: str*, *cpp_class_name: Optional[str]*, *doc: Optional[str]*)
Metadata about the op (generally not behavior impacting).

to_yaml_custom_dict()

yaml_tag = '!LinalgOpMetadata'

class iree.compiler.dialects.linalg.**OperandDef**(*kind:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandKind, *type_var: Optional[*iree.compiler.dialects.linalg.opdsl.lang.types.TypeVar*]* = *None*, *size_exprs: Optional[Sequence[*iree.compiler.dialects.linalg.opdsl.lang.affine.AffineExpr*]]* = *None*, *index_dims: Optional[Sequence[*iree.compiler.dialects.linalg.opdsl.lang.affine.DimDef*]]* = *None*, *default_indices: Optional[Sequence[int]]* = *None*, *default_fn: Optional[str]* = *None*)
Definition of an operand passed to an operation.

Keep the meta information of Tensor, Scalar, and Attribute operands and provide the shared registration functionality.

attach(*index: int*, *name: str*, *owner:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.LinalgOpDef)

is_attribute() → bool

is_input() → bool

is_tensor() → bool

class iree.compiler.dialects.linalg.**OperandDefConfig**(*operand_def: iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandDe shape_map: Optional[iree.compiler._mlir_libs._mlir.ir.AffineMap] = None, index_attr_map: Optional[iree.compiler._mlir_libs._mlir.ir.AffineMap] = None*)

Wrapper containing an operand definition with additional state.

property kind: *iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandKind*

property name: str

to_yaml_custom_dict()

property type_var: *iree.compiler.dialects.linalg.opdsl.lang.types.TypeVar*

yaml_tag = '!LinalgOperandDefConfig'

enum iree.compiler.dialects.linalg.**OperandKind**(*value*)

An enumeration.

Valid values are as follows:

INPUT_TENSOR

SCALAR

OUTPUT_TENSOR

INDEX_ATTR

UNARY_FN_ATTR

BINARY_FN_ATTR

TYPE_FN_ATTR

class iree.compiler.dialects.linalg.**PoolingNchwMaxOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNchwSumOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNcwMaxOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNcwSumOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNdhwcMaxOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNdhwcMinOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNdhwcSumOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNhwcMaxOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNhwcMaxUnsignedOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNhwcMinOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNhwcMinUnsignedOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNhwcSumOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNwcMaxOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNwcMaxUnsignedOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNwcMinOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNwcMinUnsignedOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**PoolingNwcSumOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**QuantizedBatchMatmulOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**QuantizedMatmulOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**ReduceFn**

add = **reduce_add**

max_signed = **reduce_max_signed**

max_unsigned = **reduce_max_unsigned**

min_signed = **reduce_min_signed**

min_unsigned = **reduce_min_unsigned**

mul = **reduce_mul**

class iree.compiler.dialects.linalg.**ReduceFnType**(*binary_fn:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.BinaryFnType)

Reduction function.

A binary function that reduces its RHS into its LHS.

class iree.compiler.dialects.linalg.**ReduceFnUse**(*binary_fn: Optional[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.BinaryFnT *binary_attr: Optional[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.BinaryFnA **reduce_dims:* iree.compiler.dialects.linalg.opdsl.lang.affine.DimDef)

Reduction function use.

A reduction use specifies the reduction function and dimensions.

class iree.compiler.dialects.linalg.**ReduceOp**(*result*, *inputs*, *inits*, *dimensions*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = **'linalg.reduce'**

property combiner

property inits

property inputs

class iree.compiler.dialects.linalg.ScalarArg(*arg: str*)
A type of ScalarExpression that references a named argument.

expr() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

class iree.compiler.dialects.linalg.ScalarAssign(*arg: str*, *value:* iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression)
An assignment to a named argument (LHS of a comprehension).

to_yaml_custom_dict()

yaml_tag = '!ScalarAssign'

class iree.compiler.dialects.linalg.ScalarConst(*value: str*)
A type of ScalarExpression representing a constant.

expr() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

class iree.compiler.dialects.linalg.ScalarDef(*type_var:* iree.compiler.dialects.linalg.opdsl.lang.types.TypeVar)
Scalar operand definition.

Scalar operands are forwarded to the body of the structured op as they are. A unique name identifies the scalars and an index determines their position in the operation's parameter list.

property scalar_name: str

to_scalar_expression() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

class iree.compiler.dialects.linalg.ScalarExpression(*scalar_fn: Optional[*iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarF*] = None*, *scalar_arg: Optional[*iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarA*] = None*, *scalar_const: Optional[*iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarC*] = None*, *scalar_index: Optional[*iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarI*] = None*)
An expression on scalar values.

Can be one of:

- ScalarFn
- ScalarArg
- ScalarConst
- ScalarIndex

to_yaml_custom_dict()

yaml_tag = '!ScalarExpression'

class iree.compiler.dialects.linalg.ScalarFn(*kind:* FunctionKind, *fn_name: Optional[str]*, *attr_name: Optional[str]*, *type_var: Optional[*TypeVar*]*, *operands: Sequence[*ScalarExpression*]*)
A type of ScalarExpression that applies a function.

expr() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

class iree.compiler.dialects.linalg.**ScalarIndex**(*dim: int*)
A type of ScalarExpression accessing an iteration index.

expr() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

class iree.compiler.dialects.linalg.**SymbolDef**(*symname: str*)
Represents a named symbol.

```
>>> s1 = SymbolDef("s1")
>>> s1
Symbol(s1)
>>> s2 = SymbolDef("s2")
>>> s1 is s2
False
>>> s1 is SymbolDef("s1")
True
```

ALL_SYMBOLS = {'Batch': Symbol(Batch), 'C': Symbol(C), 'CM': Symbol(CM), 'DD': Symbol(DD), 'DH': Symbol(DH), 'DW': Symbol(DW), 'F': Symbol(F), 'FG': Symbol(FG), 'G': Symbol(G), 'IC': Symbol(IC), 'K': Symbol(K), 'K0': Symbol(K0), 'KD': Symbol(KD), 'KH': Symbol(KH), 'KW': Symbol(KW), 'M': Symbol(M), 'M0': Symbol(M0), 'N': Symbol(N), 'N0': Symbol(N0), 'OD': Symbol(OD), 'OH': Symbol(OH), 'OW': Symbol(OW), 'SD': Symbol(SD), 'SH': Symbol(SH), 'SW': Symbol(SW), '__delete__': Symbol(__delete__), '__get__': Symbol(__get__), '__qualname__': Symbol(__qualname__), '__set__': Symbol(__set__), '__sphinx_mock__': Symbol(__sphinx_mock__)}

classmethod create_expando()
Create an expando class that creates unique symbols based on attr access.

class iree.compiler.dialects.linalg.**TensorDef**(*type_var:* iree.compiler.dialects.linalg.opdsl.lang.types.TypeVar, **shape:* iree.compiler.dialects.linalg.opdsl.lang.affine.AffineExprDef, *index_dims: Optional[Sequence[*iree.compiler.dialects.linalg.opdsl.lang.affine.DimDef*]] = None*, *output: bool = False*)
Tensor operand definition.

Tensor operands are indexed using the associated indexing_map when forwarded to the body of the structured op. A unique name identifies the tensor operands and an index determines their position in the operation's parameter list. A tensor definition takes type, a shape, and an optional flag to mark output tensors. Additionally, a tuple of index dimensions may be used to map the tensor to the loop dimensions of the operation. This mapping is needed to compute the indexing map of shape-only tensors that have no uses.

class iree.compiler.dialects.linalg.**TensorExpression**
An expression that can appear on the RHS of a comprehension.

collect_dim_uses(*uses: Set[*iree.compiler.dialects.linalg.opdsl.lang.affine.DimDef*]*)
Collects all DimDefs reachable through this expression.

collect_indices(*indices: Set[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.index*]*)
Collects all index accesses reachable through this expression.

collect_scalar_uses(*uses: Set[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.ScalarDef*]*)
Collects all ScalarDefs reachable through this expression.

collect_tensor_uses(*uses: Set[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorUse*]*)
Collects all TensorUses reachable through this expression.

to_scalar_expression() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

visit_tensor_exprs(*callback: Callable[[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorExpression*], None]*)
Visits all tensor expression reachable by the expression.

class iree.compiler.dialects.linalg.**TensorFn**(*kind:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.FunctionKind, *name: Optional[str]*, *operand_def: Optional[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandDef*]*, *type_var: Optional[*iree.compiler.dialects.linalg.opdsl.lang.types.TypeVar*]*, *args: Sequence[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorExpres

Application of a tensor function.

to_scalar_expression() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

visit_tensor_exprs(*callback: Callable[[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorExpression*], None]*)
Visits all tensor expression reachable by the expression.

class iree.compiler.dialects.linalg.**TensorReduceFn**(*reduce_use:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.ReduceFnUse, *args: Sequence[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.Tenso

Application of a reduction function.

This captures the lhs (initial value) separately from the rhs.

to_scalar_expression() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

visit_tensor_exprs(*callback: Callable[[*iree.compiler.dialects.linalg.opdsl.lang.comprehension.TensorExpression*], None]*)
Visits all tensor expression reachable by the expression.

class iree.compiler.dialects.linalg.**TensorUse**(*operand_def:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OperandDef, *indices: Sequence[*iree.compiler.dialects.linalg.opdsl.lang.affine.AffineExprDef*]*)
A used tensor represented by its (tensor_name, indices).

Note that forming a comprehension via direct assignment is performed through __setitem__ on the TensorDef level. However, performing a reduction with compound ops (+=, *=, etc) is done by doing a:

TensorDef.__getitem__ TensorUse.__iadd__ TensorDef.__setitem__

property tensor_name: str

to_scalar_expression() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

class iree.compiler.dialects.linalg.**TransposeOp**

OPERATION_NAME = 'linalg.transpose'

property init

property input

property region

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.linalg.TypeFn
Type conversion function namespace.

As the integer types are signless, signedness is implement by different cast functions that treat integers as signed (*cast_signed*) or unsigned (*cast_unsigned*) values.

Examples: - cast_signed(I32 -> I64) -> *arith.ExtSIOp* - cast_unsigned(I32 -> I64) -> *arith.ExtUIOp*

cast_signed = cast_signed

cast_unsigned = cast_unsigned

class iree.compiler.dialects.linalg.TypeFnAttrDef(*default:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.TypeFnType)
Type conversion function attribute definition.

Type conversion function attributes provide a way to make type conversions parameterizable. Every attribute specifies a default type conversion function that may be overwritten at operation instantiation time.

class iree.compiler.dialects.linalg.TypeFnType(*fn_name: str*)
Type conversion function.

A type conversion function takes a target type and a tensor expression and returns the casted tensor expression.

class iree.compiler.dialects.linalg.TypeVar(*name: str*)
A replaceable type variable.

Type variables are uniqued by name.

```
ALL_TYPEVARS = {'AccumType': TypeVar(AccumType), 'F32': TypeVar(F32), 'F64':
TypeVar(F64), 'I32': TypeVar(I32), 'I64': TypeVar(I64), 'LhsType':
TypeVar(LhsType), 'RhsType': TypeVar(RhsType), 'T': TypeVar(T), 'T1': TypeVar(T1),
'T2': TypeVar(T2), 'U': TypeVar(U), 'V': TypeVar(V), '__delete__':
TypeVar(__delete__), '__get__': TypeVar(__get__), '__qualname__':
TypeVar(__qualname__), '__set__': TypeVar(__set__), '__sphinx_mock__':
TypeVar(__sphinx_mock__)}
```

classmethod create_expando()
Create an expando class that creates unique type vars on attr access.

class iree.compiler.dialects.linalg.UnaryFn
Unary function namespace.

abs = abs

ceil = ceil

exp = exp

floor = floor

log = log

negf = negf

class iree.compiler.dialects.linalg.**UnaryFnAttrDef**(*default:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.UnaryFnType

Unary function attribute definition.

Unary function attributes provide a way to make the arithmetic computation parametrizable. Every attribute specifies a default unary function that may be overwritten at operation instantiation time.

class iree.compiler.dialects.linalg.**UnaryFnType**(*fn_name: str*)
Unary function.

A unary function takes one tensor expression and returns the function evaluation result.

class iree.compiler.dialects.linalg.**VecmatOp**(*inputs*, *outputs=()*, *results=()*, *loc=None*, *ip=None*)

class iree.compiler.dialects.linalg.**YAMLObject**

as_linalg_yaml()

classmethod to_yaml(*dumper*, *self*)
Default to a custom dictionary mapping.

to_yaml_custom_dict()

class iree.compiler.dialects.linalg.**YieldOp**(*values*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'linalg.yield'

property values

iree.compiler.dialects.linalg.**bind_op_def**(*op_def:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.LinalgOpDef)

class iree.compiler.dialects.linalg.**const**(*value: Any*)
Returns the given constant floating point or integer value.

to_scalar_expression() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

iree.compiler.dialects.linalg.**contextmanager**(*func*)
@contextmanager decorator.

Typical usage:

@contextmanager def some_generator(<arguments>):

<setup> try:

yield <value>

finally: <cleanup>

This makes this:

with some_generator(<arguments>) as <variable>: <body>

equivalent to this:

<setup> try:

<variable> = <value> <body>

finally: <cleanup>

iree.compiler.dialects.linalg.**current_op_def**() → *iree.compiler.dialects.linalg.opdsl.lang.comprehension.LinalgOpDef*

iree.compiler.dialects.linalg.**defines**(**definitions:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OpDefinitionDef)

iree.compiler.dialects.linalg.**domain**(**dimensions:* iree.compiler.dialects.linalg.opdsl.lang.affine.DimDef)

iree.compiler.dialects.linalg.**emit_generic_structured_op**(*op_config:* iree.compiler.dialects.linalg.opdsl.lang.config.LinalgStructur **ins:* iree.compiler._mlir_libs._mlir.ir.Value, *outs:* *Union[Sequence[*iree.compiler._mlir_libs._mlir.ir.Value*]*, iree.compiler._mlir_libs._mlir.ir.OpResultList*]*, ***attrs: Sequence[*int*]*)

iree.compiler.dialects.linalg.**emit_named_structured_op**(*op_config:* iree.compiler.dialects.linalg.opdsl.lang.config.LinalgStructured *op_name: str*, *op_class_name: str*, **ins:* iree.compiler._mlir_libs._mlir.ir.Value, *outs:* *Union[Sequence[*iree.compiler._mlir_libs._mlir.ir.Value*]*, iree.compiler._mlir_libs._mlir.ir.OpResultList*]*, ***attrs: Sequence[*int*]*)

iree.compiler.dialects.linalg.**fill_builtin_region**(*op: MlirOperation*) → None
Fill the region for *op*, which is assumed to be a builtin named Linalg op.

iree.compiler.dialects.linalg.**implements**(**interfaces:* iree.compiler.dialects.linalg.opdsl.lang.comprehension.OpInterfaceDef)

class iree.compiler.dialects.linalg.**index**(*dim:* iree.compiler.dialects.linalg.opdsl.lang.affine.DimDef)
Returns the iteration index for a given dimension name.

Resolves the given dimension name to obtain its position in the iteration domain of the operation.

resolve_dimension_name(*affine_state:* iree.compiler.dialects.linalg.opdsl.lang.affine.AffineBuildState)

to_scalar_expression() → *iree.compiler.dialects.linalg.opdsl.lang.scalar_expr.ScalarExpression*

iree.compiler.dialects.linalg.**linalg_structured_op**(*dsl_func=None*, ***, *op_name=None*, *op_class_name=None*) → *iree.compiler.dialects.linalg.opdsl.lang.dsl.DefinedOpCallable*

iree.compiler.dialects.linalg.**yaml_dump**(*data*, *sort_keys=False*, ***kwargs*)

iree.compiler.dialects.linalg.**yaml_dump_all**(*data*, *sort_keys=False*, *explicit_start=True*, ***kwargs*)

2.5.3 *math* dialect

class iree.compiler.dialects.math.**AbsFOp**(*operand*, ***, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.absf'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**AbsIOp**(*operand*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.absi'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**Atan2Op**(*lhs*, *rhs*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.atan2'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.math.**AtanOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.atan'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**CbrtOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.cbrt'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**CeilOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.ceil'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**CopySignOp**(*lhs*, *rhs*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.copysign'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.math.**CosOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.cos'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**CountLeadingZerosOp**(*operand*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.ctlz'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**CountTrailingZerosOp**(*operand*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.cttz'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**CtPopOp**(*operand*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.ctpop'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**ErfOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.erf'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**Exp2Op**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.exp2'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**ExpM1Op**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.expm1'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**ExpOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.exp'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**FPowIOp**(*lhs*, *rhs*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.fpowi'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.math.**FloorOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.floor'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**FmaOp**(*a*, *b*, *c*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.fma'

property a

property b

property c

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**IPowIOp**(*lhs*, *rhs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.ipowi'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.math.**Log10Op**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.log10'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**Log1pOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.log1p'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**Log2Op**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.log2'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**LogOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.log'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**PowFOp**(*lhs*, *rhs*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.powf'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.math.**RoundEvenOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.roundeven'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**RoundOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.round'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**RsqrtOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.rsqrt'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**SinOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.sin'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**SqrtOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.sqrt'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**TanOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.tan'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**TanhOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.tanh'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.math.**TruncOp**(*operand*, *, *fastmath=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'math.trunc'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

2.5.4 *memref* dialect

class iree.compiler.dialects.memref.**AllocOp**(*memref*, *dynamicSizes*, *symbolOperands*, *, *alignment=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.alloc'

property alignment

property dynamicSizes

property memref

property symbolOperands

class iree.compiler.dialects.memref.**AllocaOp**(*memref*, *dynamicSizes*, *symbolOperands*, *, *alignment=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.alloca'

property alignment

property dynamicSizes

property memref

property symbolOperands

class iree.compiler.dialects.memref.**AllocaScopeOp**(*results_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.alloca_scope'

property bodyRegion

property results_

class iree.compiler.dialects.memref.**AllocaScopeReturnOp**(*results_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.alloca_scope.return'

property results_

class iree.compiler.dialects.memref.**AssumeAlignmentOp**(*memref*, *alignment*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.assume_alignment'

property alignment

property memref

class iree.compiler.dialects.memref.**AtomicRMWOp**(*kind*, *value*, *memref*, *indices*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.atomic_rmw'

property indices

property memref

property result

Shortcut to get an op result if it has only one (throws an error otherwise).

property value

class iree.compiler.dialects.memref.**AtomicYieldOp**(*result*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.atomic_yield'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.memref.**CastOp**(*dest*, *source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.cast'

property dest

property source

class iree.compiler.dialects.memref.**CollapseShapeOp**(*result*, *src*, *reassociation*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.collapse_shape'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property src

class iree.compiler.dialects.memref.**CopyOp**(*source*, *target*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.copy'

property source

property target

class iree.compiler.dialects.memref.**DeallocOp**(*memref*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.dealloc'

property memref

class iree.compiler.dialects.memref.**DimOp**(*source*, *index*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.dim'

property index

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.memref.**DmaStartOp**(*operands_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.dma_start'

property operands_

class iree.compiler.dialects.memref.**DmaWaitOp**(*tagMemRef*, *tagIndices*, *numElements*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.dma_wait'

property numElements

property tagIndices

property tagMemRef

class iree.compiler.dialects.memref.**ExpandShapeOp**(*result*, *src*, *reassociation*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.expand_shape'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property src

class iree.compiler.dialects.memref.**ExtractAlignedPointerAsIndexOp**(*source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.extract_aligned_pointer_as_index'

property aligned_pointer

property source

class iree.compiler.dialects.memref.**ExtractStridedMetadataOp**(*source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.extract_strided_metadata'

property base_buffer

property offset

property sizes

property source

property strides

class iree.compiler.dialects.memref.**GenericAtomicRMWOp**

OPERATION_NAME = 'memref.generic_atomic_rmw'

property atomic_body

property indices

property memref

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.memref.**GetGlobalOp**(*result*, *name*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.get_global'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.memref.**GlobalOp**(*sym_name*, *type_*, ***, *sym_visibility=None*, *initial_value=None*, *constant=None*, *alignment=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.global'

property alignment

property constant

property initial_value

property sym_name

property sym_visibility

class iree.compiler.dialects.memref.**LoadOp**(*memref: Union[*iree.compiler._mlir_libs._mlir.ir.Operation, iree.compiler._mlir_libs._mlir.ir.OpView, iree.compiler._mlir_libs._mlir.ir.Value*]*, *indices: Optional[Union[*iree.compiler._mlir_libs._mlir.ir.Operation, iree.compiler._mlir_libs._mlir.ir.OpView, *Sequence[*iree.compiler._mlir_libs._mlir.ir.Value*]]] = None*, ***, *loc=None*, *ip=None*)

class iree.compiler.dialects.memref.**MemorySpaceCastOp**(*dest*, *source*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.memory_space_cast'

property dest

property source

class iree.compiler.dialects.memref.**PrefetchOp**(*memref*, *indices*, *isWrite*, *localityHint*, *isDataCache*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.prefetch'

property indices

property isDataCache

property isWrite

property localityHint

property memref

class iree.compiler.dialects.memref.**RankOp**(*memref*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.rank'

property memref

class iree.compiler.dialects.memref.**ReallocOp**(*result*, *source*, ***, *dynamicResultSize=None*, *alignment=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.realloc'

property alignment

property dynamicResultSize

property source

class iree.compiler.dialects.memref.**ReinterpretCastOp**(*result*, *source*, *offsets*, *sizes*, *strides*, *static_offsets*, *static_sizes*, *static_strides*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.reinterpret_cast'

property offsets

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property sizes

property source

property strides

class iree.compiler.dialects.memref.**ReshapeOp**(*result*, *source*, *shape*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.reshape'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shape

property source

class iree.compiler.dialects.memref.**StoreOp**(*value*, *memref*, *indices*, ***, *nontemporal=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.store'

property indices

property memref

property nontemporal

property value

class iree.compiler.dialects.memref.**SubViewOp**(*result*, *source*, *offsets*, *sizes*, *strides*, *static_offsets*, *static_sizes*, *static_strides*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.subview'

property offsets

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property sizes

property source

property strides

class iree.compiler.dialects.memref.**TensorStoreOp**(*tensor*, *memref*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.tensor_store'

property memref

property tensor

class iree.compiler.dialects.memref.**TransposeOp**(*result*, *in_*, *permutation*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.transpose'

property in_

class iree.compiler.dialects.memref.**ViewOp**(*result*, *source*, *byte_shift*, *sizes*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'memref.view'

property byte_shift

property sizes

property source

2.5.5 *shape* dialect

class iree.compiler.dialects.shape.**AddOp**(*lhs*, *rhs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.add'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.shape.**AnyOp**(*result*, *inputs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.any'

property inputs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**AssumingAllOp**(*inputs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.assuming_all'

property inputs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**AssumingOp**(*results_*, *witness*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.assuming'

property doRegion

property results_

property witness

class iree.compiler.dialects.shape.**AssumingYieldOp**(*operands_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.assuming_yield'

property operands_

class iree.compiler.dialects.shape.**BroadcastOp**(*result*, *shapes*, *, *error=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.broadcast'

property error

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shapes

class iree.compiler.dialects.shape.**ConcatOp**(*result*, *lhs*, *rhs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.concat'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.shape.**ConstShapeOp**(*shape*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.const_shape'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shape

class iree.compiler.dialects.shape.**ConstSizeOp**(*value*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.const_size'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property value

class iree.compiler.dialects.shape.**ConstWitnessOp**(*passing*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.const_witness'

property passing

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**CstrBroadcastableOp**(*shapes*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.cstr_broadcastable'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shapes

class iree.compiler.dialects.shape.**CstrEqOp**(*shapes*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.cstr_eq'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shapes

class iree.compiler.dialects.shape.**CstrRequireOp**(*pred*, *msg*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.cstr_require'

property msg

property pred

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**DebugPrintOp**(*output*, *input*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.debug_print'

property input

property output

class iree.compiler.dialects.shape.**DimOp**(*value*, *index*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.dim'

property extent

property index

property value

class iree.compiler.dialects.shape.**DivOp**(*lhs*, *rhs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.div'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.shape.**FromExtentTensorOp**(*input*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.from_extent_tensor'

property input

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**FromExtentsOp**(*extents*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.from_extents'

property extents

property shape

class iree.compiler.dialects.shape.**FuncOp**(*sym_name*, *function_type*, *, *arg_attrs=None*, *res_attrs=None*, *sym_visibility=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.func'

property body

property sym_name

property sym_visibility

class iree.compiler.dialects.shape.**FunctionLibraryOp**

OPERATION_NAME = 'shape.function_library'

property body

property sym_name

property sym_visibility

class iree.compiler.dialects.shape.**GetExtentOp**(*shape*, *dim*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.get_extent'

property dim

property extent

property shape

class iree.compiler.dialects.shape.**IndexToSizeOp**(*arg*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.index_to_size'

property arg

property result

Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**IsBroadcastableOp**(*shapes*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.is_broadcastable'

property result

Shortcut to get an op result if it has only one (throws an error otherwise).

property shapes

class iree.compiler.dialects.shape.**MaxOp**(*lhs*, *rhs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.max'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.shape.**MeetOp**(*arg0*, *arg1*, *, *error=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.meet'

property arg0

property arg1

property error

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**MinOp**(*lhs*, *rhs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.min'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.shape.**MulOp**(*lhs*, *rhs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.mul'

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.shape.**NumElementsOp**(*shape*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.num_elements'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shape

class iree.compiler.dialects.shape.**RankOp**(*shape*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.rank'

property rank

property shape

class iree.compiler.dialects.shape.**ReduceOp**(*result*, *shape*, *initVals*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.reduce'

property initVals

property region

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shape

class iree.compiler.dialects.shape.**ReturnOp**(*operands_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.return'

property operands_

class iree.compiler.dialects.shape.**ShapeEqOp**(*shapes*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.shape_eq'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shapes

class iree.compiler.dialects.shape.**ShapeOfOp**(*arg*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.shape_of'

property arg

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**SizeToIndexOp**(*arg*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.size_to_index'

property arg

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**SplitAtOp**(*head*, *tail*, *operand*, *index*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.split_at'

property head

property index

property operand

property tail

class iree.compiler.dialects.shape.**ToExtentTensorOp**(*result*, *input*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.to_extent_tensor'

property input

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**ValueAsShapeOp**(*result*, *arg*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.value_as_shape'

property arg

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**ValueOfOp**(*result*, *arg*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.value_of'

property arg

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.shape.**WithOp**(*operand*, *shape*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.with_shape'

property operand

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shape

class iree.compiler.dialects.shape.**YieldOp**(*operands_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'shape.yield'

property operands_

2.5.6 *std* dialect

2.5.7 *tensor* dialect

class iree.compiler.dialects.tensor.**CastOp**(*dest*, *source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.cast'

property dest

property source

class iree.compiler.dialects.tensor.**CollapseShapeOp**(*result*, *src*, *reassociation*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.collapse_shape'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property src

class iree.compiler.dialects.tensor.**DimOp**(*source*, *index*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.dim'

property index

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.tensor.**EmptyOp**(*sizes: Sequence[Union[int, iree.compiler._mlir_libs._mlir.ir.Value]]*, *element_type: iree.compiler._mlir_libs._mlir.ir.Type*, *, *loc=None*, *ip=None*)

class iree.compiler.dialects.tensor.**ExpandShapeOp**(*result*, *src*, *reassociation*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.expand_shape'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property src

class iree.compiler.dialects.tensor.**ExtractOp**(*tensor*, *indices*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.extract'

property indices

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property tensor

class iree.compiler.dialects.tensor.**ExtractSliceOp**(*result*, *source*, *offsets*, *sizes*, *strides*, *static_offsets*, *static_sizes*, *static_strides*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.extract_slice'

property offsets

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property sizes

property source

property strides

class iree.compiler.dialects.tensor.**FromElementsOp**

OPERATION_NAME = 'tensor.from_elements'

property elements

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.tensor.**GatherOp**(*result*, *source*, *indices*, *gather_dims*, *, *unique=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.gather'

property indices

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

property unique

class iree.compiler.dialects.tensor.**GenerateOp**(*result*, *dynamicExtents*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.generate'

property body

property dynamicExtents

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.tensor.**InsertOp**(*scalar*, *dest*, *indices*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.insert'

property dest

property indices

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property scalar

class iree.compiler.dialects.tensor.**InsertSliceOp**(*source*, *dest*, *offsets*, *sizes*, *strides*, *static_offsets*, *static_sizes*, *static_strides*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.insert_slice'

property dest

property offsets

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property sizes

property source

property strides

class iree.compiler.dialects.tensor.**PackOp**(*source*, *dest*, *inner_dims_pos*, *inner_tiles*, *static_inner_tiles*, *, *padding_value=None*, *outer_dims_perm=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.pack'

property dest

property inner_tiles

property padding_value

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.tensor.**PadOp**(*result*, *source*, *low*, *high*, *static_low*, *static_high*, ***, *nofold=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.pad'

property high

property low

property nofold

property region

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.tensor.**ParallelInsertSliceOp**(*source*, *dest*, *offsets*, *sizes*, *strides*, *static_offsets*, *static_sizes*, *static_strides*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.parallel_insert_slice'

property dest

property offsets

property sizes

property source

property strides

class iree.compiler.dialects.tensor.**RankOp**(*tensor*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.rank'

property tensor

class iree.compiler.dialects.tensor.**ReshapeOp**(*result*, *source*, *shape*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.reshape'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property shape

property source

class iree.compiler.dialects.tensor.**ScatterOp**(*result*, *source*, *dest*, *indices*, *scatter_dims*, ***, *unique=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.scatter'

property dest

property indices

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

property unique

class iree.compiler.dialects.tensor.**SplatOp**(*aggregate*, *input*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.splat'

property aggregate

property input

class iree.compiler.dialects.tensor.**UnPackOp**(*source*, *dest*, *inner_dims_pos*, *inner_tiles*, *static_inner_tiles*, ***, *outer_dims_perm=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.unpack'

property dest

property inner_tiles

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.tensor.**YieldOp**(*value*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tensor.yield'

property value

2.5.8 *tosa* dialect

class iree.compiler.dialects.tosa.**AbsOp**(*output*, *input1*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.abs'

property input1

property output

class iree.compiler.dialects.tosa.**AddOp**(*output*, *input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.add'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**ApplyScaleOp**(*output*, *value*, *multiplier*, *shift*, *double_round*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.apply_scale'

property double_round

property multiplier

property output

property shift

property value

class iree.compiler.dialects.tosa.**ArgMaxOp**(*output*, *input*, *axis*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.argmax'

property axis

property input

property output

class iree.compiler.dialects.tosa.**ArithmeticRightShiftOp**(*output*, *input1*, *input2*, *round*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.arithmetic_right_shift'

property input1

property input2

property output

property round

class iree.compiler.dialects.tosa.**AvgPool2dOp**(*output*, *input*, *kernel*, *stride*, *pad*, ***, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.avg_pool2d'

property input

property output

class iree.compiler.dialects.tosa.**BitwiseAndOp**(*output*, *input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.bitwise_and'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**BitwiseNotOp**(*output*, *input1*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.bitwise_not'

property input1

property output

class iree.compiler.dialects.tosa.**BitwiseOrOp**(*output*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.bitwise_or'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**BitwiseXorOp**(*output*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.bitwise_xor'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**CastOp**(*output*, *input*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.cast'

property input

property output

class iree.compiler.dialects.tosa.**CeilOp**(*output*, *input1*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.ceil'

property input1

property output

class iree.compiler.dialects.tosa.**ClampOp**(*output*, *input*, *min_int*, *max_int*, *min_fp*, *max_fp*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.clamp'

property input

property max_fp

property max_int

property min_fp

property min_int

property output

class iree.compiler.dialects.tosa.**ClzOp**(*output*, *input1*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.clz'

property input1

property output

class iree.compiler.dialects.tosa.ConcatOp(*input1*, *axis*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.concat'

property axis

property input1

property output

class iree.compiler.dialects.tosa.ConstOp(*value*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.const'

property output

class iree.compiler.dialects.tosa.Conv2DOp(*output*, *input*, *weight*, *bias*, *pad*, *stride*, *dilation*, *, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.conv2d'

property bias

property input

property output

property weight

class iree.compiler.dialects.tosa.Conv3DOp(*output*, *input*, *weight*, *bias*, *pad*, *stride*, *dilation*, *, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.conv3d'

property bias

property input

property output

property weight

class iree.compiler.dialects.tosa.CustomOp(*outputs*, *identifier*, *config*, *implementation_attrs*, *inputs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.custom'

property config

property identifier

property implementation_attrs

property inputs

property outputs

class iree.compiler.dialects.tosa.**DepthwiseConv2DOp**(*output*, *input*, *weight*, *bias*, *pad*, *stride*, *dilation*, ***, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.depthwise_conv2d'

property bias

property input

property output

property weight

class iree.compiler.dialects.tosa.**DivOp**(*output*, *input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.div'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**EqualOp**(*input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.equal'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**ExpOp**(*output*, *input1*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.exp'

property input1

property output

class iree.compiler.dialects.tosa.**FFT2dOp**(*output_real*, *output_imag*, *input_real*, *input_imag*, *inverse*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.fft2d'

property input_imag

property input_real

property inverse

property output_imag

property output_real

class iree.compiler.dialects.tosa.**FloorOp**(*output*, *input1*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.floor'

property input1

property output

class iree.compiler.dialects.tosa.**FullyConnectedOp**(*output*, *input*, *weight*, *bias*, ***, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.fully_connected'

property bias

property input

property output

property weight

class iree.compiler.dialects.tosa.**GatherOp**(*output*, *values*, *indices*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.gather'

property indices

property output

property values

class iree.compiler.dialects.tosa.**GreaterEqualOp**(*output*, *input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.greater_equal'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**GreaterOp**(*output*, *input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.greater'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**IdentityOp**(*output*, *input1*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.identity'

property input1

property output

class iree.compiler.dialects.tosa.**IfOp**(*output*, *cond*, *inputs*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.cond_if'

property cond

property else_branch

property inputs

property output

property then_branch

class iree.compiler.dialects.tosa.**LogOp**(*output*, *input1*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.log'

property input1

property output

class iree.compiler.dialects.tosa.**LogicalAndOp**(*z*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.logical_and'

property input1

property input2

property z

class iree.compiler.dialects.tosa.**LogicalLeftShiftOp**(*output*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.logical_left_shift'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**LogicalNotOp**(*input1*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.logical_not'

property input1

property output

class iree.compiler.dialects.tosa.**LogicalOrOp**(*z*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.logical_or'

property input1

property input2

property z

class iree.compiler.dialects.tosa.**LogicalRightShiftOp**(*output*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.logical_right_shift'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**LogicalXorOp**(*z*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.logical_xor'

property input1

property input2

property z

class iree.compiler.dialects.tosa.**MatMulOp**(*c*, *a*, *b*, ***, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.matmul'

property a

property b

property c

class iree.compiler.dialects.tosa.**MaxPool2dOp**(*output*, *input*, *kernel*, *stride*, *pad*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.max_pool2d'

property input

property output

class iree.compiler.dialects.tosa.**MaximumOp**(*output*, *input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.maximum'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**MinimumOp**(*output*, *input1*, *input2*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.minimum'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**MulOp**(*output*, *input1*, *input2*, *shift*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.mul'

property input1

property input2

property output

property shift

class iree.compiler.dialects.tosa.**NegateOp**(*output*, *input1*, ***, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.negate'

property input1

property output

class iree.compiler.dialects.tosa.**PadOp**(*output*, *input1*, *padding*, *, *pad_const=None*, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.pad'

property input1

property output

property pad_const

property padding

class iree.compiler.dialects.tosa.**PowOp**(*z*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.pow'

property input1

property input2

property z

class iree.compiler.dialects.tosa.**RFFT2dOp**(*output_real*, *output_imag*, *input*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.rfft2d'

property input

property output_imag

property output_real

class iree.compiler.dialects.tosa.**ReciprocalOp**(*output*, *input1*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reciprocal'

property input1

property output

class iree.compiler.dialects.tosa.**ReduceAllOp**(*output*, *input*, *axis*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reduce_all'

property axis

property input

property output

class iree.compiler.dialects.tosa.**ReduceAnyOp**(*output*, *input*, *axis*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reduce_any'

property axis

property input

property output

class iree.compiler.dialects.tosa.**ReduceMaxOp**(*output*, *input*, *axis*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reduce_max'

property axis

property input

property output

class iree.compiler.dialects.tosa.**ReduceMinOp**(*output*, *input*, *axis*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reduce_min'

property axis

property input

property output

class iree.compiler.dialects.tosa.**ReduceProdOp**(*output*, *input*, *axis*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reduce_prod'

property axis

property input

property output

class iree.compiler.dialects.tosa.**ReduceSumOp**(*output*, *input*, *axis*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reduce_sum'

property axis

property input

property output

class iree.compiler.dialects.tosa.**RescaleOp**(*output*, *input*, *input_zp*, *output_zp*, *multiplier*, *shift*, *scale32*, *double_round*, *per_channel*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.rescale'

property double_round

property input

property input_zp

property output

property output_zp

property per_channel

property scale32

class iree.compiler.dialects.tosa.**ReshapeOp**(*output*, *input1*, *new_shape*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reshape'

property input1

property output

class iree.compiler.dialects.tosa.**ResizeOp**(*output*, *input*, *scale*, *offset*, *border*, *mode*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.resize'

property input

property mode

property output

class iree.compiler.dialects.tosa.**ReverseOp**(*output*, *input*, *axis*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.reverse'

property axis

property input

property output

class iree.compiler.dialects.tosa.**RsqrtOp**(*output*, *input1*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.rsqrt'

property input1

property output

class iree.compiler.dialects.tosa.**ScatterOp**(*values_out*, *values_in*, *indices*, *input*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.scatter'

property indices

property input

property values_in

property values_out

class iree.compiler.dialects.tosa.**SelectOp**(*output*, *pred*, *on_true*, *on_false*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.select'

property on_false

property on_true

property output

property pred

class iree.compiler.dialects.tosa.**SigmoidOp**(*output*, *input*, ***, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.sigmoid'

property input

property output

class iree.compiler.dialects.tosa.**SliceOp**(*output*, *input*, *start*, *size*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.slice'

property input

property output

class iree.compiler.dialects.tosa.**SubOp**(*output*, *input1*, *input2*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.sub'

property input1

property input2

property output

class iree.compiler.dialects.tosa.**TableOp**(*output*, *input*, *table*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.table'

property input

property output

property table

class iree.compiler.dialects.tosa.**TanhOp**(*output*, *input*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.tanh'

property input

property output

class iree.compiler.dialects.tosa.**TileOp**(*output*, *input1*, *multiples*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.tile'

property input1

property output

class iree.compiler.dialects.tosa.**TransposeConv2DOp**(*output*, *input*, *filter*, *bias*, *out_pad*, *stride*, *out_shape*, *, *quantization_info=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.transpose_conv2d'

property bias

property filter

property input

property output

class iree.compiler.dialects.tosa.**TransposeOp**(*output*, *input1*, *perms*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.transpose'

property input1

property output

property perms

class iree.compiler.dialects.tosa.**WhileOp**(*output*, *inputs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.while_loop'

property body

property cond

property inputs

property output

class iree.compiler.dialects.tosa.**YieldOp**(*inputs*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'tosa.yield'

property inputs

2.5.9 *vector* dialect

class iree.compiler.dialects.vector.**BitCastOp**(*result*, *source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.bitcast'

property result

Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.vector.**BroadcastOp**(*vector*, *source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.broadcast'

property source

property vector

class iree.compiler.dialects.vector.**CompressStoreOp**(*base*, *indices*, *mask*, *valueToStore*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.compressstore'

property base

property indices

property mask

property valueToStore

class iree.compiler.dialects.vector.**ConstantMaskOp**(*result*, *mask_dim_sizes*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.constant_mask'

class iree.compiler.dialects.vector.**ContractionOp**(*result*, *lhs*, *rhs*, *acc*, *masks*, *indexing_maps*, *iterator_types*, *, *kind=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.contract'

property acc

property lhs

property masks

property rhs

class iree.compiler.dialects.vector.**CreateMaskOp**(*result*, *operands_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.create_mask'

property operands_

class iree.compiler.dialects.vector.**ExpandLoadOp**(*result*, *base*, *indices*, *mask*, *pass_thru*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.expandload'

property base

property indices

property mask

property pass_thru

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.vector.**ExtractElementOp**(*vector*, *, *position=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.extractelement'

property position

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property vector

class iree.compiler.dialects.vector.**ExtractOp**(*vector*, *position*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.extract'

property vector

class iree.compiler.dialects.vector.**ExtractStridedSliceOp**(*result*, *vector*, *offsets*, *sizes*, *strides*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.extract_strided_slice'

property vector

class iree.compiler.dialects.vector.**FMAOp**(*lhs*, *rhs*, *acc*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.fma'

property acc

property lhs

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property rhs

class iree.compiler.dialects.vector.**FlatTransposeOp**(*res*, *matrix*, *rows*, *columns*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.flat_transpose'

property columns

property matrix

property res

property rows

class iree.compiler.dialects.vector.**GatherOp**(*result*, *base*, *indices*, *index_vec*, *mask*, *pass_thru*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.gather'

property base

property index_vec

property indices

property mask

property pass_thru

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.vector.**InsertElementOp**(*source*, *dest*, *, *position=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.insertelement'

property dest

property position

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.vector.**InsertOp**(*source*, *dest*, *position*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.insert'

property dest

property res

property source

class iree.compiler.dialects.vector.**InsertStridedSliceOp**(*source*, *dest*, *offsets*, *strides*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.insert_strided_slice'

property dest

property res

property source

class iree.compiler.dialects.vector.**LoadOp**(*result*, *base*, *indices*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.load'

property base

property indices

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.vector.**MaskOp**

OPERATION_NAME = 'vector.mask'

property mask

property maskRegion

property passthru

property results_

class iree.compiler.dialects.vector.**MaskedLoadOp**(*result*, *base*, *indices*, *mask*, *pass_thru*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.maskedload'

property base

property indices

property mask

property pass_thru

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.vector.**MaskedStoreOp**(*base*, *indices*, *mask*, *valueToStore*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.maskedstore'

property base

property indices

property mask

property valueToStore

class iree.compiler.dialects.vector.**MatmulOp**(*res*, *lhs*, *rhs*, *lhs_rows*, *lhs_columns*, *rhs_columns*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.matrix_multiply'

property lhs

property lhs_columns

property lhs_rows

property res

property rhs

property rhs_columns

class iree.compiler.dialects.vector.**MultiDimReductionOp**(*kind*, *source*, *acc*, *reduction_dims*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.multi_reduction'

property acc

property dest

property source

class iree.compiler.dialects.vector.**OuterProductOp**(*result*, *lhs*, *rhs*, *acc*, *, *kind=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.outerproduct'

property acc

property lhs

property rhs

class iree.compiler.dialects.vector.**PrintOp**(*source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.print'

property source

class iree.compiler.dialects.vector.**ReductionOp**(*dest*, *kind*, *vector*, *, *acc=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.reduction'

property acc

property dest

property vector

class iree.compiler.dialects.vector.**ReshapeOp**(*result*, *vector*, *input_shape*, *output_shape*, *fixed_vector_sizes*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.reshape'

property input_shape

property output_shape

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property vector

class iree.compiler.dialects.vector.**ScalableExtractOp**(*res*, *source*, *pos*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.scalable.extract'

property pos

property res

property source

class iree.compiler.dialects.vector.**ScalableInsertOp**(*source*, *dest*, *pos*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.scalable.insert'

property dest

property pos

property res

property source

class iree.compiler.dialects.vector.**ScanOp**(*kind*, *source*, *initial_value*, *reduction_dim*, *inclusive*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.scan'

property accumulated_value

property dest

property inclusive

property initial_value

property reduction_dim

property source

class iree.compiler.dialects.vector.**ScatterOp**(*base*, *indices*, *index_vec*, *mask*, *valueToStore*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.scatter'

property base

property index_vec

property indices

property mask

property valueToStore

class iree.compiler.dialects.vector.**ShapeCastOp**(*result*, *source*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.shape_cast'

property result

Shortcut to get an op result if it has only one (throws an error otherwise).

property source

class iree.compiler.dialects.vector.**ShuffleOp**(*v1*, *v2*, *mask*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.shuffle'

property v1

property v2

property vector

class iree.compiler.dialects.vector.**SplatOp**(*aggregate*, *input*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.splat'

property aggregate

property input

class iree.compiler.dialects.vector.**StoreOp**(*valueToStore*, *base*, *indices*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.store'

property base

property indices

property valueToStore

class iree.compiler.dialects.vector.**TransferReadOp**(*vector*, *source*, *indices*, *permutation_map*, *padding*, *, *mask=None*, *in_bounds=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.transfer_read'

property indices

property mask

property padding

property source

property vector

class iree.compiler.dialects.vector.**TransferWriteOp**(*result*, *vector*, *source*, *indices*, *permutation_map*, *, *mask=None*, *in_bounds=None*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.transfer_write'

property indices

property mask

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property source

property vector

class iree.compiler.dialects.vector.**TransposeOp**(*result*, *vector*, *transp*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.transpose'

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

property vector

class iree.compiler.dialects.vector.**TypeCastOp**(*result*, *memref*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.type_cast'

property memref

property result
Shortcut to get an op result if it has only one (throws an error otherwise).

class iree.compiler.dialects.vector.**VectorScaleOp**(*, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.vscale'

property res

class iree.compiler.dialects.vector.**WarpExecuteOnLane0Op**

OPERATION_NAME = 'vector.warp_execute_on_lane_0'

property args

property laneid

property results_

property warpRegion

property warp_size

class iree.compiler.dialects.vector.**YieldOp**(*operands_*, *, *loc=None*, *ip=None*)

OPERATION_NAME = 'vector.yield'

property operands_

2.6 TensorFlow Dialects

IREE's compiler is capable of ingesting selected TensorFlow dialects directly. Those that are integrated into the in-process compiler API are presented here.

2.6.1 *chlo* dialect

2.6.2 *mhlo* dialect

2.7 Embedded MLIR API

2.7.1 *iree.compiler.ir* module

class iree.compiler.ir.**AffineAddExpr**

static get(*arg0:* iree.compiler._mlir_libs._mlir.ir.AffineExpr, *arg1:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → *iree.compiler._mlir_libs._mlir.ir.AffineAddExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

class iree.compiler.ir.**AffineBinaryExpr**

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

property lhs

property rhs

class iree.compiler.ir.**AffineCeilDivExpr**

static get(*arg0:* iree.compiler._mlir_libs._mlir.ir.AffineExpr, *arg1:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → *iree.compiler._mlir_libs._mlir.ir.AffineCeilDivExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

class iree.compiler.ir.**AffineConstantExpr**

static get(*value: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineConstantExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

property value

class iree.compiler.ir.**AffineDimExpr**

static get(*position: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineDimExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

property position

class iree.compiler.ir.**AffineExpr**

compose(*self: iree.compiler._mlir_libs._mlir.ir.AffineExpr*, *arg0: mlir::python::PyAffineMap*) → *iree.compiler._mlir_libs._mlir.ir.AffineExpr*

property context

dump(*self:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → None
Dumps a debug representation of the object to stderr.

static get_add(**args*, ***kwargs*)
Overloaded function.

1. get_add(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineAddExpr

Gets an affine expression containing a sum of two expressions.

2. get_add(arg0: int, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineAddExpr

Gets an affine expression containing a sum of a constant and another expression.

3. get_add(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: int) -> (anonymous namespace)::PyAffineAddExpr

Gets an affine expression containing a sum of an expression and a constant.

static get_ceil_div(**args*, ***kwargs*)
Overloaded function.

1. get_ceil_div(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineCeilDivExpr

Gets an affine expression containing the rounded-up result of dividing one expression by another.

2. get_ceil_div(arg0: int, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineCeilDivExpr

Gets a semi-affine expression containing the rounded-up result of dividing a constant by an expression.

3. get_ceil_div(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: int) -> (anonymous namespace)::PyAffineCeilDivExpr

Gets an affine expression containing the rounded-up result of dividing an expression by a constant.

static get_constant(*value: int*, *context: mlir.ir.Context = None*) → (anonymous namespace)::PyAffineConstantExpr
Gets a constant affine expression with the given value.

static get_dim(*position: int*, *context: mlir.ir.Context = None*) → (anonymous namespace)::PyAffineDimExpr
Gets an affine expression of a dimension at the given position.

static get_floor_div(**args*, ***kwargs*)
Overloaded function.

1. get_floor_div(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineFloorDivExpr

Gets an affine expression containing the rounded-down result of dividing one expression by another.

2. get_floor_div(arg0: int, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineFloorDivExpr

Gets a semi-affine expression containing the rounded-down result of dividing a constant by an expression.

3. get_floor_div(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: int) -> (anonymous namespace)::PyAffineFloorDivExpr

Gets an affine expression containing the rounded-down result of dividing an expression by a constant.

static get_mod(**args*, ***kwargs*)

Overloaded function.

1. get_mod(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineModExpr

Gets an affine expression containing the modulo of dividing one expression by another.

2. get_mod(arg0: int, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineModExpr

Gets a semi-affine expression containing the modulo of dividing a constant by an expression.

3. get_mod(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: int) -> (anonymous namespace)::PyAffineModExpr

Gets an affine expression containing the module of dividingan expression by a constant.

static get_mul(**args*, ***kwargs*)

Overloaded function.

1. get_mul(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineMulExpr

Gets an affine expression containing a product of two expressions.

2. get_mul(arg0: int, arg1: iree.compiler._mlir_libs._mlir.ir.AffineExpr) -> (anonymous namespace)::PyAffineMulExpr

Gets an affine expression containing a product of a constant and another expression.

3. get_mul(arg0: iree.compiler._mlir_libs._mlir.ir.AffineExpr, arg1: int) -> (anonymous namespace)::PyAffineMulExpr

Gets an affine expression containing a product of an expression and a constant.

static get_symbol(*position: int*, *context: mlir.ir.Context = None*) → (anonymous namespace)::PyAffineSymbolExpr

Gets an affine expression of a symbol at the given position.

class iree.compiler.ir.**AffineExprList**

class iree.compiler.ir.**AffineFloorDivExpr**

static get(*arg0:* iree.compiler._mlir_libs._mlir.ir.AffineExpr, *arg1:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → *iree.compiler._mlir_libs._mlir.ir.AffineFloorDivExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

class iree.compiler.ir.**AffineMap**

static compress_unused_symbols(*arg0: list*, *arg1: mlir.ir.Context*) → List[*iree.compiler._mlir_libs._mlir.ir.AffineMap*]

property context

Context that owns the Affine Map

dump(*self:* iree.compiler._mlir_libs._mlir.ir.AffineMap) → None
Dumps a debug representation of the object to stderr.

static get(*dim_count: int*, *symbol_count: int*, *exprs: list*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*
Gets a map with the given expressions as results.

static get_constant(*value: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*
Gets an affine map with a single constant result

static get_empty(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*
Gets an empty affine map.

static get_identity(*n_dims: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*
Gets an identity map with the given number of dimensions.

get_major_submap(*self:* iree.compiler._mlir_libs._mlir.ir.AffineMap, *n_results: int*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*

static get_minor_identity(*n_dims: int*, *n_results: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*
Gets a minor identity map with the given number of dimensions and results.

get_minor_submap(*self:* iree.compiler._mlir_libs._mlir.ir.AffineMap, *n_results: int*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*

static get_permutation(*permutation: List[int]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*
Gets an affine map that permutes its inputs.

get_submap(*self:* iree.compiler._mlir_libs._mlir.ir.AffineMap, *result_positions: List[int]*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*

property is_permutation

property is_projected_permutation

property n_dims

property n_inputs

property n_symbols

replace(*self:* iree.compiler._mlir_libs._mlir.ir.AffineMap, *expr:* iree.compiler._mlir_libs._mlir.ir.AffineExpr, *replacement:* iree.compiler._mlir_libs._mlir.ir.AffineExpr, *n_result_dims: int*, *n_result_syms: int*) → *iree.compiler._mlir_libs._mlir.ir.AffineMap*

property results

class iree.compiler.ir.**AffineMapAttr**

static get(*affine_map:* iree.compiler._mlir_libs._mlir.ir.AffineMap) → *iree.compiler._mlir_libs._mlir.ir.AffineMapAttr*
Gets an attribute wrapping an AffineMap.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**AffineModExpr**

static get(*arg0:* iree.compiler._mlir_libs._mlir.ir.AffineExpr, *arg1:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → *iree.compiler._mlir_libs._mlir.ir.AffineModExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

class iree.compiler.ir.**AffineMulExpr**

static get(*arg0:* iree.compiler._mlir_libs._mlir.ir.AffineExpr, *arg1:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → *iree.compiler._mlir_libs._mlir.ir.AffineMulExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

class iree.compiler.ir.**AffineSymbolExpr**

static get(*position: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.AffineSymbolExpr*

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.AffineExpr) → bool

property position

class iree.compiler.ir.**ArrayAttr**

static get(*attributes: list*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.ArrayAttr*
Gets a uniqued Array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**ArrayAttributeIterator**

class iree.compiler.ir.**AttrBuilder**

static contains(*arg0: str*) → bool

static get(*arg0: str*) → function

static insert(*arg0: str*, *arg1: function*) → None

class iree.compiler.ir.**Attribute**

property context
Context that owns the Attribute

dump(*self:* iree.compiler._mlir_libs._mlir.ir.Attribute) → None
Dumps a debug representation of the object to stderr.

get_named(*self:* iree.compiler._mlir_libs._mlir.ir.Attribute, *arg0: str*) → mlir::python::PyNamedAttribute
Binds a name to the attribute

static parse(*asm: str*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Attribute*
Parses an attribute from an assembly form. Raises an MLIRError on failure.

property type

class iree.compiler.ir.**BF16Type**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.BF16Type*
Create a bf16 type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Block**

append(*self:* iree.compiler._mlir_libs._mlir.ir.Block, *operation:* *iree.compiler._mlir_libs._mlir.ir._OperationBase*) → None
Appends an operation to this block. If the operation is currently in another block, it will be moved.

append_to(*self:* iree.compiler._mlir_libs._mlir.ir.Block, *arg0:* iree.compiler._mlir_libs._mlir.ir.Region) → None
Append this block to a region, transferring ownership if necessary

property arguments
Returns a list of block arguments.

create_after(*self:* iree.compiler._mlir_libs._mlir.ir.Block, **args*) → *iree.compiler._mlir_libs._mlir.ir.Block*
Creates and returns a new Block after this block (with given argument types).

static create_at_start(*parent:* iree.compiler._mlir_libs._mlir.ir.Region, *arg_types: list = []*) → *iree.compiler._mlir_libs._mlir.ir.Block*
Creates and returns a new Block at the beginning of the given region (with given argument types).

create_before(*self:* iree.compiler._mlir_libs._mlir.ir.Block, **args*) → *iree.compiler._mlir_libs._mlir.ir.Block*
Creates and returns a new Block before this block (with given argument types).

property operations
Returns a forward-optimized sequence of operations.

property owner
Returns the owning operation of this block.

property region
Returns the owning region of this block.

class iree.compiler.ir.**BlockArgument**

property arg_number

static isinstance(*other_value:* iree.compiler._mlir_libs._mlir.ir.Value) → bool

property owner

set_type(*self:* iree.compiler._mlir_libs._mlir.ir.BlockArgument, *type:* iree.compiler._mlir_libs._mlir.ir.Type) → None

class iree.compiler.ir.**BlockArgumentList**

property types

class iree.compiler.ir.**BlockIterator**

class iree.compiler.ir.**BlockList**

append(*self:* iree.compiler._mlir_libs._mlir.ir.BlockList, **args*) → *iree.compiler._mlir_libs._mlir.ir.Block*
Appends a new block, with argument types as positional args.

Returns The created block.

class iree.compiler.ir.**BoolAttr**

static get(*value: bool*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.BoolAttr*
Gets an uniqued bool attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

property value
Returns the value of the bool attribute

class iree.compiler.ir.**ComplexType**

property element_type
Returns element type.

static get(*arg0:* iree.compiler._mlir_libs._mlir.ir.Type) → *iree.compiler._mlir_libs._mlir.ir.ComplexType*
Create a complex type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Context**(**args*, ***kwargs*)

class iree.compiler.ir.**DenseBoolArrayAttr**

static get(*values: List[int]*, *context: mlir.ir.Context = None*) →
iree.compiler._mlir_libs._mlir.ir.DenseBoolArrayAttr
Gets a uniqued dense array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseBoolArrayIterator**

class iree.compiler.ir.**DenseElementsAttr**

static get(*array: buffer*, *signless: bool = True*, *type: Optional[*iree.compiler._mlir_libs._mlir.ir.Type*] =*
None, *shape: Optional[List[int]] = None*, *context: mlir.ir.Context = None*) →
iree.compiler._mlir_libs._mlir.ir.DenseElementsAttr
Gets a DenseElementsAttr from a Python buffer or array.

When *type* is not provided, then some limited type inferencing is done based on the buffer format. Support presently exists for 8/16/32/64 signed and unsigned integers and float16/float32/float64. DenseElementsAttrs of these types can also be converted back to a corresponding buffer.

For conversions outside of these types, a *type=* must be explicitly provided and the buffer contents must be bit-castable to the MLIR internal representation:

- Integer types (except for i1): the buffer must be byte aligned to the next byte boundary.
- Floating point types: Must be bit-castable to the given floating point size.
- i1 (bool): Bit packed into 8bit words where the bit pattern matches a row major ordering. An arbitrary Numpy *bool_* array can be bit packed to this specification with: *np.packbits(ary, axis=None, bitorder='little')*.

If a single element buffer is passed (or for i1, a single byte with value 0 or 255), then a splat will be created.

Parameters

- **array** – The array or buffer to convert.

- **signless** – If inferring an appropriate MLIR type, use signless types for integers (defaults True).
- **type** – Skips inference of the MLIR element type and uses this instead. The storage size must be consistent with the actual contents of the buffer.
- **shape** – Overrides the shape of the buffer when constructing the MLIR shaped type. This is needed when the physical and logical shape differ (as for i1).
- **context** – Explicit context, if not from context manager.

Returns DenseElementsAttr on success.

Raises **ValueError** – If the type of the buffer or array cannot be matched to an MLIR type or if the buffer does not meet expectations.

static get_splat(*shaped_type:* iree.compiler._mlir_libs._mlir.ir.Type, *element_attr:* iree.compiler._mlir_libs._mlir.ir.Attribute) → *iree.compiler._mlir_libs._mlir.ir.DenseElementsAttr*

Gets a DenseElementsAttr where all values are the same

get_splat_value(*self:* iree.compiler._mlir_libs._mlir.ir.DenseElementsAttr) → *iree.compiler._mlir_libs._mlir.ir.Attribute*

property is_splat

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseF32ArrayAttr**

static get(*values: List[float]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.DenseF32ArrayAttr*

Gets a uniqued dense array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseF32ArrayIterator**

class iree.compiler.ir.**DenseF64ArrayAttr**

static get(*values: List[float]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.DenseF64ArrayAttr*

Gets a uniqued dense array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseF64ArrayIterator**

class iree.compiler.ir.**DenseFPElementsAttr**

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseI16ArrayAttr**

static get(*values: List[int]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.DenseI16ArrayAttr*

Gets a uniqued dense array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseI16ArrayIterator**

class iree.compiler.ir.**DenseI32ArrayAttr**

static get(*values: List[int]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.DenseI32ArrayAttr*

Gets a uniqued dense array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseI32ArrayIterator**

class iree.compiler.ir.**DenseI64ArrayAttr**

static get(*values: List[int]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.DenseI64ArrayAttr*

Gets a uniqued dense array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseI64ArrayIterator**

class iree.compiler.ir.**DenseI8ArrayAttr**

static get(*values: List[int]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.DenseI8ArrayAttr*

Gets a uniqued dense array attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**DenseI8ArrayIterator**

class iree.compiler.ir.**DenseIntElementsAttr**

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**Diagnostic**

property location

property message

property notes

property severity

class iree.compiler.ir.**DiagnosticHandler**

property **attached**

detach(*self:* iree.compiler._mlir_libs._mlir.ir.DiagnosticHandler) → None

property **had_error**

class iree.compiler.ir.**DiagnosticInfo**

property **location**

property **message**

property **notes**

property **severity**

class iree.compiler.ir.**DiagnosticSeverity**

Members:

ERROR

WARNING

NOTE

REMARK

ERROR = <DiagnosticSeverity.ERROR: 0>

NOTE = <DiagnosticSeverity.NOTE: 2>

REMARK = <DiagnosticSeverity.REMARK: 3>

WARNING = <DiagnosticSeverity.WARNING: 1>

property **name**

property **value**

class iree.compiler.ir.**Dialect**

property **descriptor**

class iree.compiler.ir.**DialectDescriptor**

property **namespace**

class iree.compiler.ir.**DialectRegistry**

class iree.compiler.ir.**Dialects**

class iree.compiler.ir.**DictAttr**

static **get**(*value: dict = {}*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.DictAttr*

Gets an uniqued dict attribute

static **isinstance**(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property **type**

class iree.compiler.ir.**F16Type**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.F16Type*
Create a f16 type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**F32Type**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.F32Type*
Create a f32 type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**F64Type**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.F64Type*
Create a f64 type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**FlatSymbolRefAttr**

static get(*value: str*, *context: mlir.ir.Context = None*) →
iree.compiler._mlir_libs._mlir.ir.FlatSymbolRefAttr
Gets a uniqued FlatSymbolRef attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

property value
Returns the value of the FlatSymbolRef attribute as a string

class iree.compiler.ir.**Float8E4M3B11FNUZType**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Float8E4M3B11FNUZType*
Create a float8_e4m3b11fnuz type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Float8E4M3FNType**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Float8E4M3FNType*
Create a float8_e4m3fn type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Float8E4M3FNUZType**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Float8E4M3FNUZType*
Create a float8_e4m3fnuz type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Float8E5M2FNUZType**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Float8E5M2FNUZType*
Create a float8_e5m2fnuz type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Float8E5M2Type**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Float8E5M2Type*
Create a float8_e5m2 type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**FloatAttr**

static get(*type:* iree.compiler._mlir_libs._mlir.ir.Type, *value: float*, *loc: mlir.ir.Location = None*) → *iree.compiler._mlir_libs._mlir.ir.FloatAttr*
Gets an uniqued float point attribute associated to a type

static get_f32(*value: float*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.FloatAttr*
Gets an uniqued float point attribute associated to a f32 type

static get_f64(*value: float*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.FloatAttr*
Gets an uniqued float point attribute associated to a f64 type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

property value
Returns the value of the float point attribute

class iree.compiler.ir.**FunctionType**

static get(*inputs: List[*iree.compiler._mlir_libs._mlir.ir.Type*]*, *results: List[*iree.compiler._mlir_libs._mlir.ir.Type*]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.FunctionType*
Gets a FunctionType from a list of input and result types

property inputs
Returns the list of input types in the FunctionType.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

property results
Returns the list of result types in the FunctionType.

class iree.compiler.ir.**IndexType**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.IndexType*
Create a index type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**InferTypeOpInterface**

inferReturnTypes(*self:* iree.compiler._mlir_libs._mlir.ir.InferTypeOpInterface, *operands: Optional[list] = None*, *attributes: Optional[*iree.compiler._mlir_libs._mlir.ir.Attribute*] = None*, *regions: Optional[List[*iree.compiler._mlir_libs._mlir.ir.Region*]] = None*, *context: mlir.ir.Context = None*, *loc: mlir.ir.Location = None*) → List[*iree.compiler._mlir_libs._mlir.ir.Type*]
Given the arguments required to build an operation, attempts to infer its return types. Raises ValueError on failure.

property operation
Returns an Operation for which the interface was constructed.

property opview
Returns an OpView subclass _instance_ for which the interface was constructed

class iree.compiler.ir.**InsertionPoint**

static at_block_begin(*block:* iree.compiler._mlir_libs._mlir.ir.Block) → *iree.compiler._mlir_libs._mlir.ir.InsertionPoint*
Inserts at the beginning of the block.

static at_block_terminator(*block:* iree.compiler._mlir_libs._mlir.ir.Block) → *iree.compiler._mlir_libs._mlir.ir.InsertionPoint*
Inserts before the block terminator.

property block
Returns the block that this InsertionPoint points to.

property current

insert(*self:* iree.compiler._mlir_libs._mlir.ir.InsertionPoint, *operation: iree.compiler._mlir_libs._mlir.ir._OperationBase*) → None
Inserts an operation.

class iree.compiler.ir.**IntegerAttr**

static get(*type:* iree.compiler._mlir_libs._mlir.ir.Type, *value: int*) → *iree.compiler._mlir_libs._mlir.ir.IntegerAttr*
Gets an uniqued integer attribute associated to a type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

property value
Returns the value of the integer attribute

class iree.compiler.ir.**IntegerSet**

property constraints

property context

dump(*self:* iree.compiler._mlir_libs._mlir.ir.IntegerSet) → None
Dumps a debug representation of the object to stderr.

static get(*num_dims: int*, *num_symbols: int*, *exprs: list*, *eq_flags: List[bool]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.IntegerSet*

static get_empty(*num_dims: int*, *num_symbols: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.IntegerSet*

get_replaced(*self:* iree.compiler._mlir_libs._mlir.ir.IntegerSet, *dim_exprs: list*, *symbol_exprs: list*, *num_result_dims: int*, *num_result_symbols: int*) → *iree.compiler._mlir_libs._mlir.ir.IntegerSet*

property is_canonical_empty

property n_dims

property n_equalities

property n_inequalities

property n_inputs

property n_symbols

class iree.compiler.ir.**IntegerSetConstraint**

property expr

property is_eq

class iree.compiler.ir.**IntegerSetConstraintList**

class iree.compiler.ir.**IntegerType**

static get_signed(*width: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.IntegerType*

Create a signed integer type

static get_signless(*width: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.IntegerType*

Create a signless integer type

static get_unsigned(*width: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.IntegerType*

Create an unsigned integer type

property is_signed

Returns whether this is a signed integer

property is_signless

Returns whether this is a signless integer

property is_unsigned

Returns whether this is an unsigned integer

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

property width

Returns the width of the integer type

class iree.compiler.ir.**Location**

property attr

Get the underlying LocationAttr

static callsite(*callee:* iree.compiler._mlir_libs._mlir.ir.Location, *frames:* *List[*iree.compiler._mlir_libs._mlir.ir.Location*]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Location*

Gets a Location representing a caller and callsite

property context

Context that owns the Location

property current

emit_error(*self:* iree.compiler._mlir_libs._mlir.ir.Location, *message: str*) → None

Emits an error at this location

static file(*filename: str*, *line: int*, *col: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Location*

Gets a Location representing a file, line and column

static from_attr(*attribute: mlir::python::PyAttribute*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Location*

Gets a Location from a LocationAttr

static fused(*locations: List[iree.compiler._mlir_libs._mlir.ir.Location]*, *metadata: Optional[mlir::python::PyAttribute] = None*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Location*

Gets a Location representing a fused location with optional metadata

static name(*name: str*, *childLoc: Optional[*iree.compiler._mlir_libs._mlir.ir.Location*] = None*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Location*

Gets a Location representing a named location with optional child location

static unknown(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Location*

Gets a Location representing an unknown location

exception iree.compiler.ir.**MLIRError**(*message*, *error_diagnostics*)

An exception with diagnostic information. Has the following fields: message: str error_diagnostics: List[ir.DiagnosticInfo]

class iree.compiler.ir.**MemRefType**

property affine_map

The layout of the MemRef type as an affine map.

static get(*shape: List[int]*, *element_type:* iree.compiler._mlir_libs._mlir.ir.Type, *layout:* iree.compiler._mlir_libs._mlir.ir.Attribute *= None*, *memory_space:* iree.compiler._mlir_libs._mlir.ir.Attribute *= None*, *loc: mlir.ir.Location = None*) → *iree.compiler._mlir_libs._mlir.ir.MemRefType*

Create a memref type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

property layout

The layout of the MemRef type.

property memory_space

Returns the memory space of the given MemRef type.

class iree.compiler.ir.**Module**

property body

Return the block for this module

property context

Context that created the Module

static create(*loc: mlir.ir.Location = None*) → object

Creates an empty module

dump(*self:* iree.compiler._mlir_libs._mlir.ir.Module) → None

Dumps a debug representation of the object to stderr.

property operation

Accesses the module as an operation

static parse(*asm: str*, *context: mlir.ir.Context = None*) → object

Parses a module's assembly format from a string.

Returns a new MlirModule or raises an MLIRError if the parsing fails.

See also: https://mlir.llvm.org/docs/LangRef/

class iree.compiler.ir.**NamedAttribute**

property attr
The underlying generic attribute of the NamedAttribute binding

property name
The name of the NamedAttribute binding

class iree.compiler.ir.**NoneType**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.NoneType*
Create a none type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**OpAttributeMap**

class iree.compiler.ir.**OpOperand**

property operand_number

property owner

class iree.compiler.ir.**OpOperandIterator**

class iree.compiler.ir.**OpOperandList**

class iree.compiler.ir.**OpResult**

static isinstance(*other_value:* iree.compiler._mlir_libs._mlir.ir.Value) → bool

property owner

property result_number

class iree.compiler.ir.**OpResultList**

property types

class iree.compiler.ir.**OpView**

classmethod build_generic()
(cls: object, results: list = None, operands: list = None, attributes: Optional[dict] = None, successors: Optional[List[mlir::python::PyBlock]] = None, regions: Optional[int] = None, loc: mlir.ir.Location = None, ip: object = None) -> object

Builds a specific, generated OpView based on class level attributes.

property context
Context that owns the Operation

property operation

classmethod parse()
(cls: object, source: str, *, source_name: str = ‘’, context: mlir.ir.Context = None) -> object

Parses a specific, generated OpView based on class level attributes

class iree.compiler.ir.**OpaqueAttr**

property data
Returns the data for the Opaqued attributes as a string

property dialect_namespace
Returns the dialect namespace for the Opaque attribute as a string

static get(*dialect_namespace: str*, *buffer: buffer*, *type:* iree.compiler._mlir_libs._mlir.ir.Type, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.OpaqueAttr*
Gets an Opaque attribute.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**OpaqueType**

property data
Returns the data for the Opaque type as a string.

property dialect_namespace
Returns the dialect namespace for the Opaque type as a string.

static get(*dialect_namespace: str*, *buffer: str*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.OpaqueType*
Create an unregistered (opaque) dialect type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Operation**

clone(*self:* iree.compiler._mlir_libs._mlir.ir.Operation, *ip: object = None*) → object

property context
Context that owns the Operation

static create(*name: str*, *results: Optional[List[mlir::python::PyType]] = None*, *operands: Optional[List[mlir::python::PyValue]] = None*, *attributes: Optional[dict] = None*, *successors: Optional[List[mlir::python::PyBlock]] = None*, *regions: int = 0*, *loc: mlir.ir.Location = None*, *ip: object = None*) → object
Creates a new operation.

Parameters

- **name** – Operation name (e.g. "dialect.operation").
- **results** – Sequence of Type representing op result types.
- **attributes** – Dict of str:Attribute.
- **successors** – List of Block for the operation's successors.
- **regions** – Number of regions to create.
- **location** – A Location object (defaults to resolve from context manager).
- **ip** – An InsertionPoint (defaults to resolve from context manager or set to False to disable insertion, even with an insertion point set in the context manager).

Returns A new "detached" Operation object. Detached operations can be added to blocks, which causes them to become "attached."

erase(*self:* iree.compiler._mlir_libs._mlir.ir.Operation) → None

property name

property opview

property parent

static parse(*source: str*, *, *source_name: str = ''*, *context: mlir.ir.Context = None*) → object
Parses an operation. Supports both text assembly format and binary bytecode format.

class iree.compiler.ir.**OperationIterator**

class iree.compiler.ir.**OperationList**

class iree.compiler.ir.**RankedTensorType**

property encoding

static get(*shape: List[int]*, *element_type:* iree.compiler._mlir_libs._mlir.ir.Type, *encoding: Optional[*iree.compiler._mlir_libs._mlir.ir.Attribute*] = None*, *loc: mlir.ir.Location = None*) → *iree.compiler._mlir_libs._mlir.ir.RankedTensorType*
Create a ranked tensor type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Region**

property blocks
Returns a forward-optimized sequence of blocks.

property owner
Returns the operation owning this region.

class iree.compiler.ir.**RegionIterator**

class iree.compiler.ir.**RegionSequence**

class iree.compiler.ir.**ShapedType**

property element_type
Returns the element type of the shaped type.

get_dim_size(*self:* iree.compiler._mlir_libs._mlir.ir.ShapedType, *dim: int*) → int
Returns the dim-th dimension of the given ranked shaped type.

static get_dynamic_size() → int
Returns the value used to indicate dynamic dimensions in shaped types.

static get_dynamic_stride_or_offset() → int
Returns the value used to indicate dynamic strides or offsets in shaped types.

property has_rank
Returns whether the given shaped type is ranked.

property has_static_shape
Returns whether the given shaped type has a static shape.

is_dynamic_dim(*self:* iree.compiler._mlir_libs._mlir.ir.ShapedType, *dim: int*) → bool
Returns whether the dim-th dimension of the given shaped type is dynamic.

static is_dynamic_size(*dim_size: int*) → bool
Returns whether the given dimension size indicates a dynamic dimension.

is_dynamic_stride_or_offset(*self:* iree.compiler._mlir_libs._mlir.ir.ShapedType, *dim_size: int*) → bool
Returns whether the given value is used as a placeholder for dynamic strides and offsets in shaped types.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

property rank
Returns the rank of the given ranked shaped type.

property shape
Returns the shape of the ranked shaped type as a list of integers.

class iree.compiler.ir.**StridedLayoutAttr**

static get(*offset: int*, *strides: List[int]*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.StridedLayoutAttr*
Gets a strided layout attribute.

static get_fully_dynamic(*rank: int*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.StridedLayoutAttr*
Gets a strided layout attribute with dynamic offset and strides of a given rank.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property offset
Returns the value of the float point attribute

property strides
Returns the value of the float point attribute

property type

class iree.compiler.ir.**StringAttr**

static get(*value: str*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.StringAttr*
Gets a uniqued string attribute

static get_typed(*type:* iree.compiler._mlir_libs._mlir.ir.Type, *value: str*) → *iree.compiler._mlir_libs._mlir.ir.StringAttr*
Gets a uniqued string attribute associated to a type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

property value
Returns the value of the string attribute

class iree.compiler.ir.**SymbolTable**

erase(*self:* iree.compiler._mlir_libs._mlir.ir.SymbolTable, *operation: iree.compiler._mlir_libs._mlir.ir._OperationBase*) → None

static get_symbol_name(*symbol: iree.compiler._mlir_libs._mlir.ir._OperationBase*) → *iree.compiler._mlir_libs._mlir.ir.Attribute*

static get_visibility(*symbol: iree.compiler._mlir_libs._mlir.ir._OperationBase*) → *iree.compiler._mlir_libs._mlir.ir.Attribute*

insert(*self:* iree.compiler._mlir_libs._mlir.ir.SymbolTable, *operation: iree.compiler._mlir_libs._mlir.ir._OperationBase*) → *iree.compiler._mlir_libs._mlir.ir.Attribute*

static replace_all_symbol_uses(*old_symbol: str*, *new_symbol: str*, *from_op: iree.compiler._mlir_libs._mlir.ir._OperationBase*) → None

static set_symbol_name(*symbol: iree.compiler._mlir_libs._mlir.ir._OperationBase*, *name: str*) → None

static set_visibility(*symbol: iree.compiler._mlir_libs._mlir.ir._OperationBase*, *visibility: str*) → None

static walk_symbol_tables(*from_op: iree.compiler._mlir_libs._mlir.ir._OperationBase*, *all_sym_uses_visible: bool*, *callback: object*) → None

class iree.compiler.ir.**TupleType**

static get_tuple(*elements: list*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.TupleType*

Create a tuple type

get_type(*self:* iree.compiler._mlir_libs._mlir.ir.TupleType, *pos: int*) → *iree.compiler._mlir_libs._mlir.ir.Type*

Returns the pos-th type in the tuple type.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

property num_types

Returns the number of types contained in a tuple.

class iree.compiler.ir.**Type**

property context

Context that owns the Type

dump(*self:* iree.compiler._mlir_libs._mlir.ir.Type) → None

Dumps a debug representation of the object to stderr.

static parse(*asm: str*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.Type*

Parses the assembly form of a type.

Returns a Type object or raises an MLIRError if the type cannot be parsed.

See also: https://mlir.llvm.org/docs/LangRef/#type-system

class iree.compiler.ir.**TypeAttr**

static get(*value:* iree.compiler._mlir_libs._mlir.ir.Type, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.TypeAttr*

Gets a uniqued Type attribute

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

property value

class iree.compiler.ir.**UnitAttr**

static get(*context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.ir.UnitAttr*

Create a Unit attribute.

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Attribute) → bool

property type

class iree.compiler.ir.**UnrankedMemRefType**

static get(*element_type:* iree.compiler._mlir_libs._mlir.ir.Type, *memory_space:* iree.compiler._mlir_libs._mlir.ir.Attribute, *loc: mlir.ir.Location = None*) → *iree.compiler._mlir_libs._mlir.ir.UnrankedMemRefType*
Create a unranked memref type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

property memory_space
Returns the memory space of the given Unranked MemRef type.

class iree.compiler.ir.**UnrankedTensorType**

static get(*element_type:* iree.compiler._mlir_libs._mlir.ir.Type, *loc: mlir.ir.Location = None*) → *iree.compiler._mlir_libs._mlir.ir.UnrankedTensorType*
Create a unranked tensor type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

class iree.compiler.ir.**Value**

property context
Context in which the value lives.

dump(*self:* iree.compiler._mlir_libs._mlir.ir.Value) → None
Dumps a debug representation of the object to stderr.

property owner

property type

property uses

class iree.compiler.ir.**VectorType**

static get(*shape: List[int]*, *elementType:* iree.compiler._mlir_libs._mlir.ir.Type, *loc: mlir.ir.Location = None*) → *iree.compiler._mlir_libs._mlir.ir.VectorType*
Create a vector type

static isinstance(*other:* iree.compiler._mlir_libs._mlir.ir.Type) → bool

iree.compiler.ir.**register_attribute_builder**(*kind*)

2.7.2 *iree.compiler.passmanager* module

class iree.compiler.passmanager.**PassManager**

add(*self:* iree.compiler._mlir_libs._mlir.passmanager.PassManager, *pipeline: str*) → None
Add textual pipeline elements to the pass manager. Throws a ValueError if the pipeline can't be parsed.

enable_ir_printing(*self:* iree.compiler._mlir_libs._mlir.passmanager.PassManager) → None
Enable mlir-print-ir-after-all.

enable_verifier(*self:* iree.compiler._mlir_libs._mlir.passmanager.PassManager, *enable: bool*) → None
Enable / disable verify-each.

static parse(*pipeline: str*, *context: mlir.ir.Context = None*) → *iree.compiler._mlir_libs._mlir.passmanager.PassManager*

Parse a textual pass-pipeline and return a top-level PassManager that can be applied on a Module. Throw a ValueError if the pipeline can't be parsed

run(*self:* iree.compiler._mlir_libs._mlir.passmanager.PassManager, *operation: iree.compiler._mlir_libs._mlir.ir._OperationBase*) → None

Run the pass manager on the provided operation, raising an MLIRError on failure.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z