Aliases

An alias is an alternative name for an existing QL entity.

Once you've defined an alias, you can use that new name to refer to the entity in the current module's namespace.

Defining an alias

You can define an alias in the body of any module. To do this, you should specify:

1. The keyword `module`, `class`, or `predicate` to define an alias for a module, type, or non-member predicate respectively.
2. The name of the alias. This should be a valid name for that kind of entity. For example, a valid predicate alias starts with a lowercase letter.
3. A reference to the QL entity. This includes the original name of the entity and, for predicates, the arity of the predicate.

You can also annotate an alias. See the list of annotations available for aliases.

Note that these annotations apply to the name introduced by the alias (and not the underlying QL entity itself). For example, an alias can have different visibility to the name that it aliases.

Module aliases

Use the following syntax to define an alias for a module:

```
module ModAlias = ModuleName;
```

For example, if you create a new module `NewVersion` that is an updated version of `OldVersion`, you could deprecate the name `OldVersion` as follows:

```
deprecated module OldVersion = NewVersion;
```

That way both names resolve to the same module, but if you use the name `OldVersion`, a deprecation warning is displayed.

Type aliases

Use the following syntax to define an alias for a type:

```
class TypeAlias = TypeName;
```

Note that `class` is just a keyword. You can define an alias for any type—namely, primitive types, database types and user-defined classes.

For example, you can use an alias to abbreviate the name of the primitive type `boolean` to `Bool`:

```
class Bool = boolean;
```

Or, to use a class `OneTwo` defined in a module `M` in `OneTwoThreeLib.qll`, you could create an alias to use the shorter name `OT` instead:

```
import OneTwoThreeLib

class OT = M::OneTwo;

...

from OT ot
select ot
```

Predicate aliases

Use the following syntax to define an alias for a non-member predicate:

```
predicate PredAlias = PredicateName/Arity;
```

This works for predicates with or without result.

For example, suppose you frequently use the following predicate, which calculates the successor of a positive integer less than ten:

```
int getSuccessor(int i) {
  result = i + 1 and
  i in [1 .. 9]
}
```

You can use an alias to abbreviate the name to `succ`:

```
predicate succ = getSuccessor/1;
```

As an example of a predicate without result, suppose you have a predicate that holds for any positive integer less than ten:

```
predicate isSmall(int i) {
  i in [1 .. 9]
}
```

You could give the predicate a more descriptive name as follows:

```
predicate lessThanTen = isSmall/1;
```