Operations on Object

The `Object` interface has a number of operations. We cannot define type `Object` in Slice because `Object` is a keyword; regardless, here is what (part of) the definition of `Object` would look like if it were legal:

Slice

```
sequence<string> StrSeq;

interface Object {                          // "Pseudo" Slice!
    idempotent void   ice_ping();
    idempotent bool   ice_isA(string typeID);
    idempotent string ice_id();
    idempotent StrSeq ice_ids();
    // ...
};
```

Note that, apart from the illegal use of the keyword `Object` as the interface name, the operation names all contain the `ice_` prefix. This prefix is reserved for use by Ice and cannot clash with a user-defined operation. This means that all Slice interfaces can inherit from `Object` without name clashes. We discuss these built-in operations below.

On this page:

`ice_ping`

All interfaces support the `ice_ping` operation. That operation is useful for debugging because it provides a basic reachability test for an object: if the object exists and a message can successfully be dispatched to the object, `ice_ping` simply returns without error. If the object cannot be reached or does not exist, `ice_ping` throws a run-time exception that provides the reason for the failure.

`ice_isA`

The `ice_isA` operation accepts a type identifier (such as the identifier returned by `ice_id`) and tests whether the target object supports the specified type, returning `true` if it does. You can use this operation to check whether a target object supports a particular type. For example, referring to the diagram Implicit Inheritance from Object once more, assume that you are holding a proxy to a target object of type `AlarmClock`. The table below illustrates the result of calling `ice_isA` on that proxy with various arguments. (We assume that all types in the Implicit inheritance from Object diagram are defined in a module `Times`):

Argument	Result
`::Ice::Object`	`true`
`::Times::Clock`	`true`
`::Times::AlarmClock`	`true`
`::Times::Radio`	`false`
`::Times::RadioClock`	`false`

Calling `ice_isA` on a proxy denoting an object of type AlarmClock.

As expected, `ice_isA` returns true for `::Times::Clock` and `::Times::AlarmClock` and also returns true for `::Ice::Object` (because all interfaces support that type). Obviously, an `AlarmClock` supports neither the `Radio` nor the `RadioClock` interfaces, so `ice_isA` returns false for these types.

`ice_id`

The `ice_id` operation returns the type ID of the most-derived type of an interface.

`ice_ids`

The `ice_ids` operation returns a sequence of type IDs that contains all of the type IDs supported by an interface. For example, for the RadioClock interface in Implicit inheritance from Object, `ice_ids` returns a sequence containing the type IDs `::Ice::Object`, `::Times::Clock`, `::Times::AlarmClock`, `::Times::Radio`, and `::Times::RadioClock`.

See Also

- Type IDs
- Interface Inheritance
- Implicit inheritance from Object