Local Types

In order to access certain features of the Ice run time, you must use APIs that are provided by libraries. However, instead of defining an API that is specific to each implementation language, Ice defines its APIs in Slice using the `local` keyword. The advantage of defining APIs in Slice is that a single definition suffices to define the API for all possible implementation languages. The actual language-specific API is then generated by the Slice compiler for each implementation language. Types that are provided by Ice libraries are defined using the Slice `local` keyword.

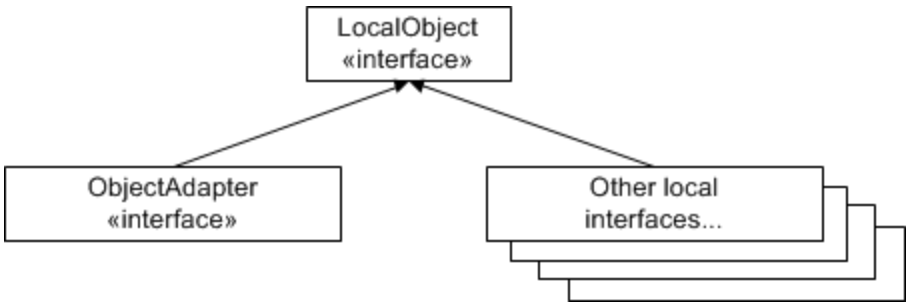
For example:

Slice

```
module Ice {
    local interface ObjectAdapter {
        // ...
    };
};
```

Any Slice definition (not just interfaces) can have a `local` modifier. If the `local` modifier is present, the Slice compiler does not generate marshaling code for the corresponding type. This means that a local type can *never* be accessed remotely because it cannot be transmitted between client and server. (The Slice compiler prevents use of `local` types in non-`local` contexts.)

In addition, local interfaces and local classes do *not* inherit from `Ice::Object`. Instead, local interfaces and classes have their own, completely separate inheritance hierarchy. At the root of this hierarchy is the type `Ice::LocalObject`, as shown:



Inheritance from `LocalObject`.

Because local interfaces form a completely separate inheritance hierarchy, you cannot pass a local interface where a non-local interface is expected, and vice-versa.

You rarely need to define local types for your own applications — the `local` keyword exists mainly to allow definition of APIs for the Ice run time. (Because local objects cannot be invoked remotely, there is little point for an application to define local objects; it might as well define ordinary programming-language objects instead.) However, there is one exception to this rule: servant locators must be implemented as local objects.

See Also

- Servant Locators