Value Factories

Prior to Ice 3.7, an application needed to register an object factory with the Ice run time for two use cases:

1. to successfully unmarshal an instance of a Slice class that defined operations, or
2. to supply a custom implementation of a Slice class, regardless of whether that class defined operations.

Since Ice 3.7 deprecates classes with operations, we now refer to instances of Slice classes as *values* and the Ice run time provides a new (but similar) API for managing value factories. Generally speaking, applications will rarely need to use this API, with use case #2 above now being the primary motivation.

The following Slice definitions comprise the value factory API:

Slice

```
module Ice
{
    local interface ValueFactory
    {
        Value create(string type);
    }

    local interface ValueFactoryManager
    {
        void add(ValueFactory factory, string type);
        ValueFactory find(string type);
    }

    local interface Communicator
    {
        ValueFactoryManager getValueFactoryManager();
        // ...
    }
}
```

An application-defined value factory must provide an implementation of the `ValueFactory` interface. Its `create` operation receives the Slice type ID corresponding to the Slice class that the Ice run time is attempting to unmarshal. The `create` implementation can return nil if it's unable to instantiate the type or doesn't recognize the type, otherwise the factory must return an instance of the requested type or a type derived from the requested type.

The Ice run time supplies a default implementation of the `ValueFactoryManager` interface, although an application can optionally substitute its own implementation during communicator initialization. You can obtain the value factory manager by calling `getValueFactoryManager` on the communicator object. The manager's `add` operation registers a factory for a particular Slice type ID, or you can pass an empty string as the type and Ice will use that factory as the default in cases where no other factory was registered for a type. The `add` operation raises `AlreadyRegisteredException` if another factory has already been registered for the specified type.

Finally, the manager's `find` operation returns the factory registered for a type, or nil if no match was found.

Please refer to the relevant language mapping chapters for instructions on using the value factory API in your programming language.

See Also

- Classes
- Type IDs