Data Encoding for Interfaces

Interfaces can be marshaled by value. For an interface marshaled by value (as opposed to a class instance derived from that interface), only the type ID of the most-derived interface is encoded. We use the following Slice definitions in our discussion:

Slice

```
interface Base { /* ... */ };

interface Derived extends Base { /* ... */ };

interface Example {
    void doSomething(Base b);
};
```

We assume that the client passes a class instance to `doSomething` that does not have a Slice definition (but implements `Derived`).

Interface Encoding version 1.0

The on-the-wire representation of the interface parameter using encoding version 1.0 is as follows:

Marshaled Value	Size in Bytes	Type	Byte offset
`1` *(identity)*	4	`int`	0
`0` *(marker for class type ID)*	1	`bool`	4
`"::Derived"` *(class type ID)*	10	`string`	5
`4` *(byte count for slice)*	4	`int`	15
`0` *(marker for class type ID)*	1	`bool`	19
`"::Ice::Object"` *(class type ID)*	14	`string`	20
`5` *(byte count for slice)*	4	`int`	34
`0` *(number of dictionary entries)*	1	`size`	38

Interface Encoding version 1.1

With encoding version 1.1, the interface parameter is encoded as if it was a class with no data members. Here is the on-the-wire representation using the sliced format:

Marshaled value	Size in bytes	Type	Byte offset
`1` *(instance marker)*	1	`size`	0
`17` *(slice flags: string type ID, size is present)*	1	`byte`	1
`"::Derived"` *(type ID - assigned index 1)*	10	`string`	2
`4` *(byte count for slice)*	4	`int`	12

The compact format omits the byte count for the slice:

Marshaled value	Size in bytes	Type	Byte offset
`1` *(instance marker)*	1	`size`	0
`1` *(slice flags: string type ID)*	1	`byte`	1
`"::Derived"` *(type ID - assigned index 1)*	10	`string`	2

See Also

- Passing Interfaces by Value