Data Encoding for Class Type IDs

Unlike for exception type IDs, class type IDs are not simple strings. Instead, a class type ID is marshaled as a boolean followed by either a string or a size, to conserve bandwidth. To illustrate this, consider the following class hierarchy:

Slice

```
class Base {
    // ...
};

class Derived extends Base {
    // ...
};
```

The type IDs for the class slices are ::Derived and ::Base. Suppose the sender marshals three instances of ::Derived as part of a single request. (For example, two instances could be out-parameters and one instance could be the return value.)

The first instance that is sent on the wire contains the type IDs ::Derived and ::Base preceding their respective slices. Because marshaling proceeds in derived-to-base order, the first type ID that is sent is ::Derived. Every time the sender sends a type ID that it has not sent previously in the same request, it sends the boolean value false, followed by the type ID. Internally, the sender also assigns a unique positive number to each type ID. These numbers start at 1 and increment by one for each type ID that has not been marshaled previously. This means that the first type ID is encoded as the boolean value false, followed by ::Derived, and the second type ID is encoded as the boolean value false, followed by ::Base.

When the sender marshals the remaining two instances, it consults a lookup table of previously-marshaled type IDs. Because both type IDs were sent previously in the same request (or reply), the sender encodes all further occurrences of ::Derived as the value true followed by the number 1 encoded as a size, and it encodes all further occurrences of ::Base as the value true followed by the number 2 encoded as a size.

When the receiver reads a type ID, it first reads its boolean marker:

- If the boolean is false, the receiver reads a string and enters that string into a lookup table that maps integers to strings. The first new class type ID received in a request is numbered 1, the second new class type ID is numbered 2, and so on.
- If the boolean value is true, the receiver reads a number encoded as a size and uses it to retrieve the corresponding class type ID from the lookup table.

Note that this numbering scheme is re-established for each new encapsulation. (As we will see in our discussion of protocol messages, parameters, return values, and exceptions are always marshaled inside an enclosing encapsulation.) For subsequent or nested encapsulation, the numbering scheme restarts, with the first new type ID being assigned the value 1. In other words, each encapsulation uses its own independent numbering scheme for class type IDs to satisfy the constraint that encapsulations must not depend on their surrounding context.

Encoding class type IDs in this way provides significant savings in bandwidth: whenever an ID is marshaled a second and subsequent time, it is marshaled as a two-byte value (assuming no more than 254 distinct type IDs per request) instead of as a string. Because type IDs can be long, especially if you are using nested modules, the savings are considerable.

See Also

- Type IDs
- Basic Data Encoding
- Protocol Messages