Ruby Mapping for Sequences

Previous

Next

On this page:

Mapping Slice Sequences to Ruby Arrays

A Slice sequence maps to a Ruby array; the only exception is a sequence of bytes, which maps to a string. The use of a Ruby array means that the mapping does not generate a separate named type for a Slice sequence. It also means that you can take advantage of all the array functionality provided by Ruby. For example:

Slice

```
sequence<Fruit> FruitPlatter;
```

We can use the `FruitPlatter` sequence as shown below:

Ruby

```
platter = [ Fruit::Apple, Fruit::Pear ]
platter.push(Fruit::Orange)
```

The Ice run time validates the elements of a sequence to ensure that they are compatible with the declared type; a `TypeError` exception is raised if an incompatible type is encountered.

Back to Top ^

Mapping for Byte Sequences in Ruby

A Ruby string can contain arbitrary 8-bit binary data, therefore it is a more efficient representation of a byte sequence than a Ruby array in both memory utilization and throughput performance.

When receiving a byte sequence (as the result of an operation, as an out parameter, or as a member of a data structure), the value is always represented as a string. When sending a byte sequence as an operation parameter or data member, the Ice run time accepts both a string and an array of integers as legal values. For example, consider the following Slice definitions:

Slice

```
// Slice
sequence<byte> Data;

interface I
{
    void sendData(Data d);
    Data getData();
}
```

The interpreter session below uses these Slice definitions to demonstrate the mapping for a sequence of bytes:

Ruby

```
> proxy = ...
> proxy.sendData("\0\1\2\3")   # Send as a string
> proxy.sendData([0, 1, 2, 3]) # Send as an array
> d = proxy.getData()
> d.class
=> String
> d
=> "\000\001\002\003"
```

The two invocations of `sendData` are equivalent; however, the second invocation incurs additional overhead as the Ice run time must validate the type and range of each array element.

Back to Top ^

See Also

- Sequences
- Ruby Mapping for Identifiers
- Ruby Mapping for Modules
- Ruby Mapping for Built-In Types
- Ruby Mapping for Enumerations
- Ruby Mapping for Structures
- Ruby Mapping for Dictionaries
- Ruby Mapping for Constants
- Ruby Mapping for Exceptions
- Ruby Mapping for Interfaces
- Ruby Mapping for Operations

Previous Next