The InputStream Interface in Java

An `InputStream` is created using the following function:

Java

```
package Ice;

public class Util {
    public static InputStream
    createInputStream(Communicator communicator, byte[] data);
}
```

The `InputStream` interface is shown below.

Java

```
package Ice;

public interface InputStream {
    Communicator communicator();

    void sliceObjects(boolean slice);

    boolean readBool();
    boolean[] readBoolSeq();

    byte readByte();
    byte[] readByteSeq();

    short readShort();
    short[] readShortSeq();

    int readInt();
    int[] readIntSeq();

    long readLong();
    long[] readLongSeq();

    float readFloat();
    float[] readFloatSeq();

    double readDouble();
    double[] readDoubleSeq();

    String readString();
    String[] readStringSeq();

    int readSize();
    int readAndCheckSeqSize(int minSizeWireSize);

    ObjectPrx readProxy();

    void readObject(ReadObjectCallback cb);

    String readTypeId();

    void throwException() throws UserException;

    void startSlice();
    void endSlice();
    void skipSlice();

    void startEncapsulation();
    void endEncapsulation();
    void skipEncapsulation();

    void readPendingObjects();

    java.io.Serializable readSerializable();

    void rewind();

    void destroy();
}
```

Member functions are provided for extracting all of the primitive types, as well as sequences of primitive types; these are self-explanatory. The remaining member functions have the following semantics:

- `void sliceObjects(boolean slice)` Determines the behavior of the stream when extracting Ice objects. An Ice object is "sliced" when a factory cannot be found for a Slice type ID, resulting in the creation of an object of a less-derived type. Slicing is typically disabled when the application expects all object factories to be present, in which case the exception `NoObjectFactoryException` is raised. The default behavior is to allow slicing.

- `int readSize()`
  The Ice encoding has a compact representation to indicate size. This function extracts a size and returns it as an integer.

- `int readAndCheckSeqSize(int minWireSize)`
  Like `readSize`, this function reads a size and returns it, but also verifies that there is enough data remaining in the unmarshaling buffer to successfully unmarshal the elements of the sequence. The `minWireSize` parameter indicates the smallest possible on-the-wire representation of a single sequence element. If the unmarshaling buffer contains insufficient data to unmarshal the sequence, the function throws `UnmarshalOutOfBoundsException`.

- `Ice.ObjectPrx readProxy()`
  This function returns an instance of the base proxy type, `ObjectPrx`. The Slice compiler optionally generates helper functions to extract proxies of user-defined types.

- `void readObject(ReadObjectCallback cb)`
  The Ice encoding for class instances requires extraction to occur in stages. The `readObject` function accepts a callback object of type `ReadObjectCallback`, whose definition is shown below:

  Java

  ```
  package Ice;

  public interface ReadObjectCallback {
      void invoke(Ice.Object obj);
  }
  ```

  When the object instance is available, the callback object's `invoke` member function is called. The application must call `readPendingObjects` to ensure that all instances are properly extracted. Note that applications rarely need to invoke this member function directly; the helper functions generated by the Slice compiler are easier to use.

- `String readTypeId()`
  A table of Slice type IDs is used to save space when encoding Ice objects. This function returns the type ID at the stream's current position.

- `void throwException() throws UserException`
  This function extracts a user exception from the stream and throws it. If the stored exception is of an unknown type, the function attempts to extract and throw a less-derived exception. If that also fails, an `UnmarshalOutOfBoundsException` is thrown.

- `void startSlice()`
  `void endSlice()`
  `void skipSlice()`
  Start, end, and skip a slice of member data, respectively. These functions are used when manually extracting the slices of an Ice object or user exception.

- `void startEncapsulation()`
  `void endEncapsulation()`
  `void skipEncapsulation()`
  Start, end, and skip an encapsulation, respectively.

- `void readPendingObjects()`
  An application must call this function after all other data has been extracted, but only if Ice objects were encoded. This function extracts the state of Ice objects and invokes their corresponding callback objects (see `readObject`).

- `java.io.Serializable readSerializable()`
  Reads a serializable Java object from the stream.

- `void destroy()`
  Applications must call this function in order to reclaim resources.

Here is a simple example that demonstrates how to extract a boolean and a sequence of strings from a stream:

Java

```
byte[] data = ...
Ice.InputStream in =
    Ice.Util.createInputStream(communicator, data);
try {
```

```
    boolean b = in.readBool();
    String[] seq = in.readStringSeq();
} finally {
    in.destroy();
}
```

See Also

- Basic Data Encoding
- Data Encoding for Classes
- Data Encoding for Exceptions
- Serializable Objects in Java
- Stream Helper Functions in Java