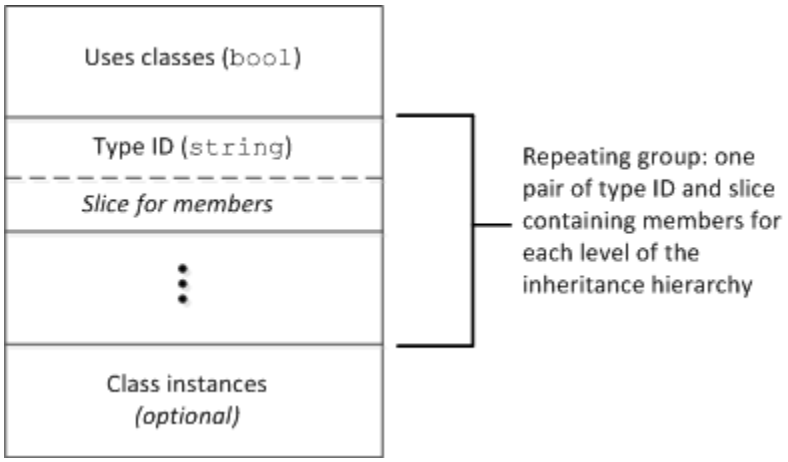
Data Encoding for Exceptions

Exception Encoding version 1.0

An exception is marshaled as shown below:



Marshaling format for exceptions.

Every exception instance is preceded by a single byte that indicates whether the exception uses class members: the byte value is `1` if any of the exception members are classes (or if any of the exception members, recursively, contain class members) and `0`, otherwise.

Following the header byte, the exception is marshaled as a sequence of pairs: the first member of each pair is the type ID for an exception slice, and the second member of the pair is a slice containing the marshaled members of that slice. The sequence of pairs is marshaled in derived-to-base order, with the most-derived slice first, and ending with the least-derived slice. Within each slice, data members are marshaled as for structures: in the order in which they are defined in the Slice definition.

Following the sequence of pairs, any class instances that are used by the members of the exception are marshaled. This final part is optional: it is present only if the header byte is `1`.

To illustrate the marshaling, consider the following exception hierarchy:

Slice

```
exception Base {
    int baseInt;
    string baseString;
};

exception Derived extends Base {
    bool derivedBool;
    string derivedString;
    double derivedDouble;
};
```

Assume that the exception members are initialized to the values shown below:

Member	Type	Value	Marshaled size (in bytes)
`baseInt`	`int`	`99`	4
`baseString`	`string`	`"Hello"`	6
`derivedBool`	`bool`	`true`	1
`derivedString`	`string`	`"World!"`	7
`derivedDouble`	`double`	`3.14`	8

Member values of an exception of type `Derived`.

From the above table, we can see that the total size of the members of `Base` is 10 bytes, and the total size of the members of `Derived` is 16 bytes. None of the exception members are classes. An instance of this exception has the on-the-wire representation shown in the next table. (The size, type, and byte offset of the marshaled representation is indicated for each component.)

Marshaled value	Size in bytes	Type	Byte offset
`0` *(no class members)*	1	`bool`	0
`"::Derived"` *(type ID)*	10	`string`	1
`20` *(byte count for slice)*	4	`int`	11
`1` (*`derivedBool`*)	1	`bool`	15
`"World!"` (*`derivedString`*)	7	`string`	16
`3.14` (*`derivedDouble`*)	8	`double`	23
`"::Base"` *(type ID)*	7	`string`	31
`14` *(byte count for slice)*	4	`int`	38
`99` (*`baseInt`*)	4	`int`	42
`"Hello"` (*`baseString`*)	6	`string`	46

Marshaled representation of the exception.

Note that the size of each string is one larger than the actual string length. This is because each string is preceded by a count of its number of bytes, as directed by the encoding for strings.

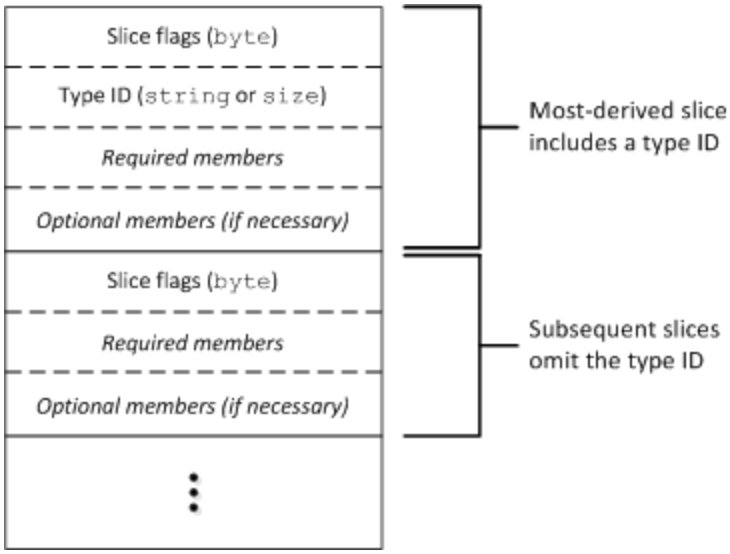
The receiver of this sequence of values uses the header byte to decide whether it eventually must unmarshal any class instances contained in the exception (none in this example) and then examines the first type ID (`::Derived`). If the receiver recognizes that type ID, it can unmarshal the contents of the first slice, followed by the remaining slices; otherwise, the receiver reads the byte count that follows the unknown type (20) and then skips 20-4 bytes in the input stream, which is the start of the type ID for the second slice (`::Base`). If the receiver does not recognize that type ID either, it again reads the byte count following the type ID (14), skips 14-4 bytes, and attempts to read another type ID. (This can happen only if client and server have been compiled with mismatched Slice definitions that disagree in the exception specification of an operation.) In this case, the receiver will eventually encounter an unmarshaling error, which it can report with a `MarshalException`.

If an exception contains class members, these members are marshaled following the exception slices as described in the class encoding.

Exception Encoding version 1.1

An exception is marshaled as a collection of slices whose order matches the inheritance hierarchy, with the most-derived type appearing first. The selected encoding format affects the content of each slice. The final slice, representing the least-derived type, has its *last slice* bit set to true.

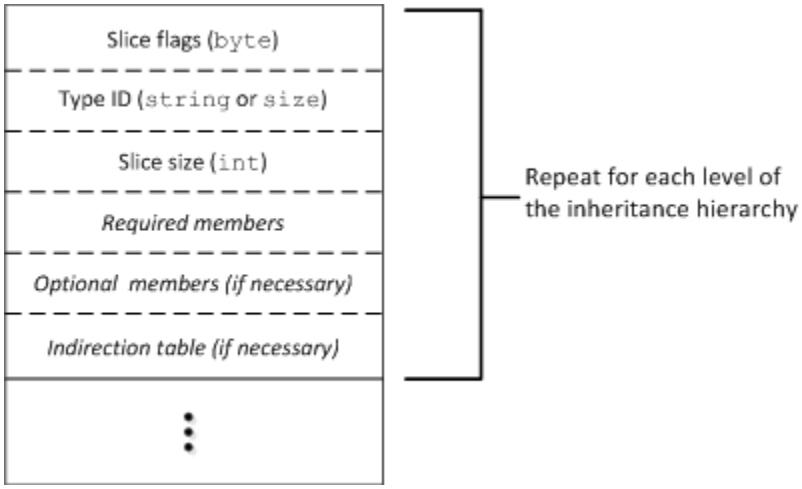
An exception in the compact format is marshaled as follows:



Compact format for exceptions.

The leading byte of each slice is a set of bit flags that specifies the features of the slice. The compact format includes a type ID in the initial (most-derived) slice but omits the type ID from all subsequent slices.

The sliced format includes a type ID in every slice, along with a slice size and an optional indirection table:



Sliced format for exceptions.

To illustrate the marshaling, consider the following exception hierarchy:

Slice

```
exception Base {
    int baseInt;
    string baseString;
};

exception Derived extends Base {
    bool derivedBool;
    string derivedString;
    double derivedDouble;
};
```

Assume that the exception members are initialized to the values shown below:

Member	Type	Value	Marshaled size (in bytes)
`baseInt`	`int`	`99`	4
`baseString`	`string`	`"Hello"`	6
`derivedBool`	`bool`	`true`	1
`derivedString`	`string`	`"World!"`	7
`derivedDouble`	`double`	`3.14`	8

Member values of an exception of type `Derived`.

From the above table, we can see that the total size of the members of `Base` is 10 bytes, and the total size of the members of `Derived` is 16 bytes. None of the exception members are classes. An instance of this exception using the sliced format has the on-the-wire representation shown in the next table. (The size, type, and byte offset of the marshaled representation is indicated for each component.)

Marshaled value	Size in bytes	Type	Byte offset
`18` *(flags: type ID is a string, slice size is present)*	1	`byte`	0
`"::Derived"` *(type ID)*	10	`string`	1

`20` *(byte count for slice)*	4	`int`	11
`1` *(`derivedBool`)*	1	`bool`	15
`"World!"` *(`derivedString`)*	7	`string`	16
`3.14` *(`derivedDouble`)*	8	`double`	23
`50` *(flags: type ID is a string, slice size is present, last slice)*	1	`byte`	31
`"::Base"` *(type ID)*	7	`string`	32
`14` *(byte count for slice)*	4	`int`	39
`99` *(`baseInt`)*	4	`int`	43
`"Hello"` *(`baseString`)*	6	`string`	47

Marshaled representation of the exception using the sliced format.

Note that the size of each string is one larger than the actual string length. This is because each string is preceded by a count of its number of bytes, as directed by the encoding for strings.

Repeating this exercise using the compact format produces the following encoding:

Marshaled value	Size in bytes	Type	Byte offset
`2` *(flags: type ID is a string)*	1	`byte`	0
`"::Derived"` *(type ID)*	10	`string`	1
`1` *(`derivedBool`)*	1	`bool`	11
`"World!"` *(`derivedString`)*	7	`string`	12
`3.14` *(`derivedDouble`)*	8	`double`	19
`32` *(flags: last slice)*	1	`byte`	27
`99` *(`baseInt`)*	4	`int`	28
`"Hello"` *(`baseString`)*	6	`string`	32

Marshaled representation of the exception using the compact format.

When using the compact format, the receiver *must* know the most-derived type: the only type ID included in the encoding is that of the most-derived type. Furthermore, the lack of slice sizes means the receiver cannot skip a slice without knowing how to decode its contents.

See Also

- Type IDs
- Basic Data Encoding
- Data Encoding for Classes