Binary Format

We define a binary format to which messages are serialized on send and recovered on reception. For binary format using TLV (Tag, Length and Value) binary markup, we use an array of bytes with each resource serialized in a tag (a byte codifying each resource type), size, name (only on structure and resource components) and value (the actual sequence of bytes resulting from serializing the resource.

The binary representation uses a modified version of the TLV format (Tag, Length and Value) used by ASN.1 [41]. This not only supports the direct integration of binary information but also facilitates parsing, since each resource, primitive or structured, can be stepped over in a breadth-first traversal, thanks to the *Length* field (the resource size in bytes).  Resources (including messages) can be represented in three levels:

Binary, no variable names (components are referred to by their position index in the resource);

Variable names, in a dictionary that maps names to position indices;

Source text, when available. Typically, runtime messages are generated directly  in binary, without source text. This is used mostly in design-time resources.

The binary representation provides native support for binary data, has a smaller length and is faster to parse (the Length field enables breadth-first parsing).

As long as we control the serialization format, we can perform the serialization in one language and the deserialization in another.

The binary format is always the result of serializing data in each language, with a tag, the number of bytes that follow and the serialized content. Recovering the serialized data is simply testing the tag to find the data type and then using the number of bytes and the serialized content.

We start by serializing several objects to the binary format. Each will have a tag, a size (number of bytes it occupies after the tag). This is implemented as following:

* An integer is always 64 bits, with each byte serialized in sequence. The receiver will recover the integer in the same way.
* Booleans can use just two different tags. There is no size or content, since the tag says it all
* Strings use a UTF-8 encoding, since it is already byte oriented. We decided which byte of each character goes first, if it has more than one byte. Check in both C# and Java which is the best way to deal with strings, in terms of serialization. You cannot use predefined serialization, unless you know exactly how it is done. It cannot be language dependent.
* Structures. Sequence of fields, in which, for each field, you should include: name (a string) and the component proper (serialized according to its type of resource)

Objects with variables should be serialized to a compound data (composed of inner data). This means having a compound data type, with its own tag, and inner components serialized according to their own data type (composition can be recursive). This is always recoverable at the receiver, thanks to the tag.

To define a set of classes implementing each type of resource, including serialization to the binary format and recovery from that format (each resource knows how to produce its bytes and to recover from bytes with its tag). The tag permits to reconstruct completely a given resource that has been serialized