Workgroup: Internet Engineering Task Force
Internet-Draft: draft-royer-bits-in-xdr-00
Published: 23 March 2025
Intended Status: Informational
Expires: 24 September 2025
Author: DM. Royer
RiverExplorer LLC

Bits in XDR

Abstract

This is an extension to the XDR specification to allow for bits to be described and sent.

With protocols that have a large number of boolean values the existing standard requires each to be individually packed into a 32-bit value.

This addition does not alter any existing XDR data streams or effect existing implementations.

- This specification describes how to pack a bit-boolean and short bit-width data values into the 32-bit XDR block chunks.
- And this specification describes how to describe them by extending "The XDR Language Specification" to include bits.
- And a new namespace declaration type is specified to aid in the reduction of name collisions in large projects.

While in draft status, a new Open Source XDR generation tool is being developed [xdrgen].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
2. The XDR Bit Language Specification 3
3. Packing Bits 6
  3.1. Packing bit-boolean 6
  3.2. Packing bit-values 7
    3.2.1. Packing wider than 32-bit-values or spanning blocks 8
4. Reducing Namespace Collision in Generated Code 10
5. IANA Considerations 13
6. Security Considerations 13
7. Normative References 13
8. Informative References 13
Appendix A. Appendix 1 Full XDR Language Grammar 13
Acknowledgments 14
Contributors 14
Author's Address 14

1. Introduction

Definitions:

- A value with 2 or more bits is called a bit-value.

  A bit-value value can be signed or unsigned. They often represent a set of possible conditions and not a numeric value and would be unsigned. And in other usages, the bits could represent a positive or negative value.
- A single bit value is called a bit-boolean. A bit-boolean is a single bit representing a true or false value. A bit-boolean by itself has no sign. Up to 32 bit-boolean fit in a 32-bit XDR block.
- A scope is a way of extending a name of a item to uniquely identify it. As an example, file A variable 'a' and file B variable 'a' can be difficult to uniquely identify in code. By adding a 'namespace A' and a second 'namespace B', they can then be identified with 'A:a' and 'B:a'.

All multi bit width values are placed into network byte order the same as their 32-bit or wider values as described in XDR [RFC4506].

2. The XDR Bit Language Specification

With an large amount of bits and data packets it is easier to have name collisions between generated object. For this reason a new 'namespace' declaration' type is specified in Section 4

A new RFC-XDR type-specifier of 'bitobject' is added to the one shown in Section 6.3 of [RFC4506]. Resulting in 'type-specifier' becoming:

```
type-specifier:
           [ "unsigned" ] "int"
         | [ "unsigned" ] "hyper"
         | "float"
         | "double"
         | "quadruple"
         | "bool"
         | enum-type-spec
         | struct-type-spec
         | union-type-spec
         | identifier
         | bitobject
```

Figure 1: Extended type-specifier ABNF

The properties of a bitobject are:

- A bitobject only consists of one or more of:
  - A signed integer value: "sbits"

  - An unsigned integer value: "ubits"
  - A boolean "bit"

- Floating point numbers would be transmitted as a float value as already described in [RFC4506] as they have a sign, exponent and a mantissa. No floating value is defined in a bitobject.
- Unused bits are set to zero (0).
- All bit widths that exceed 32-bits would be placed into two or more bitobject values. With the ones containing the most significant bits sent first and the one with the least signification bits sent last.
- A bitobject does not need to define 32-bits of data. The undefined bits are at the most significant end of the 32-bits object and and are set to zero.
- All [RFC4506] type-specifier objects are at least 32-bits wide which means that "sbits", "ubits", or "bit" can never occupy a 32-bit XDR block with a [RFC4506] type-specifier.
- A 32-bit wide "sbits" is the same as a [RFC4506], Section 4.1 signed integer. Except when they span 32-bit blocks. See Section 3.2.1.
- A 32-bit wide "ubits" is the same as a [RFC4506], Section 4.2 unsigned integer. Except when they span 32-bit blocks. See Section 3.2.1.

```
bitobject:
      "{"
         ( width-declaration ";" )
         ( width-declaration ";" )*
      "}"

width-declaration:
      "bit"   identifier
      "sbits" identifier ":" %d
      "ubits" identifier ":" %d
```

Figure 2: bitobject ABNF

Figure 3 is one example of a bitobject. that is 32-bits wide.

```
bitobject AssemblyLineStatus
{
  bit   LightOn;
  ubits Status:3;
  ubits SwitchPosition:4;
  sbits Rotation:10;
  bit   Active;
  ubits UnitsPerMinute:8;
  ubits UnitID:5;
};
```

Figure 3: Multiple Bits Example

Example Figure 4 uses 11 bit-boolean values and would be transmitted in one 32-bit block.

```
bitobject EmailStatus
{
  bit Seen:1;
  bit Answered:1;
  bit Flagged:1
  bit Deleted:1
  bit Draft:1
  bit Recent:1
  bit Forwarded:1
  bit Ignored:1
  bit Watched:1
  bit Shared:1
  bit ReadOnly:1
};
```

Figure 4: Flags Example

Example Figure 5 contains two 42 bit-boolean values and would be transmitted in three 32-bit blocks.

```
bitobject Trajectory
{
  ubits Velocity:42;
  sbits VectorX:14;
  sbits VectorY:14;
  sbits VectorZ:14;
};
```

Figure 5: Wider than 32 bit example:

In some cases, like in Figure 3 "AssemblyLineStatus" and in Figure 5 "Trajectory", the bits could represent the output of a hardware device where the bits are defined by a manufacturer.

And in other cases the bits could be logical software flags that have a predefined bit-position in a bit stream as exampled in Figure 4.

3. Packing Bits

The bit-boolean and bit-value objects are processed from the top to the bottom as shown in their bitobject XDR language definition.

The top most value would be packed into the least significant bits. The second from the top value would be placed next to it, and so on.

bit-values are converted to network byte order and then bit packed.

And a caution to the implementors for signed bit-value data. Many computer languages will convert a narrower bit value into a wider bit value and move the sign bit to the most significant position. So when preparing a signed bit-value, be sure to clamp the value and adjust the sign to the correct bit position before packing the bits. This would apply to "sbits" and not "ubits".

3.1. Packing bit-boolean

This is how the XDR Language bitobject "EmailStatus" in Figure 4 would be packed. "EmailStatus" is a bitobject that only contains bit-boolean values.

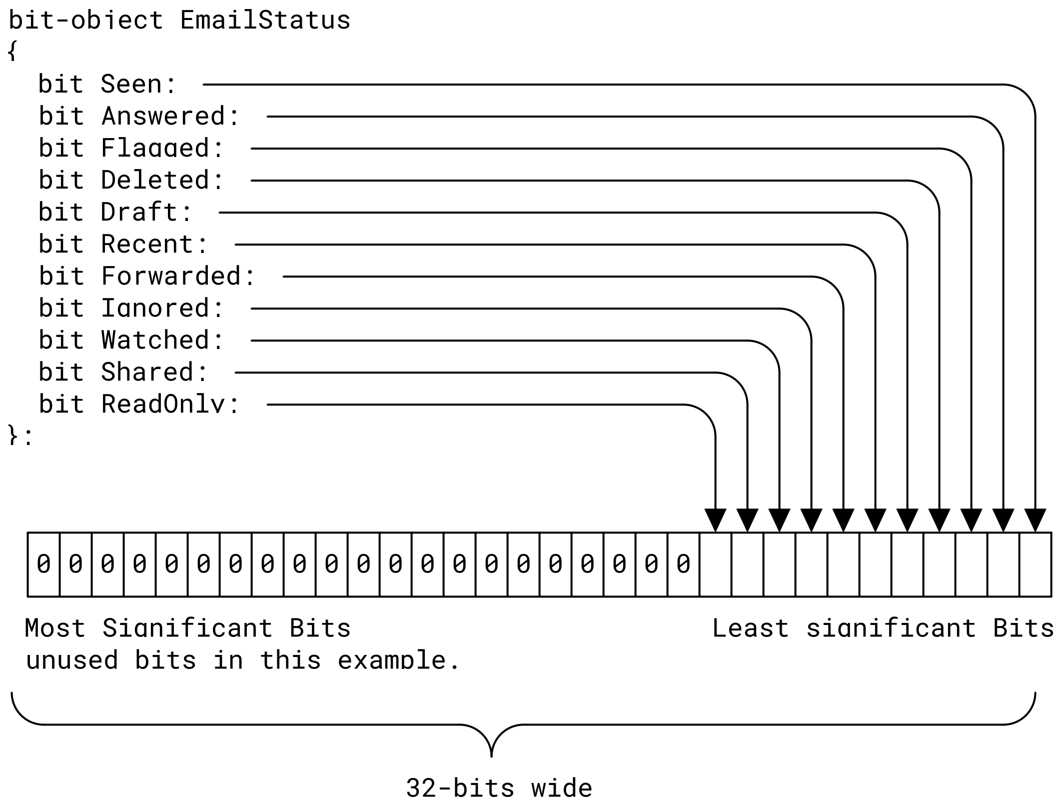

Figure 6: How Figure 4 would be packed.

3.2. Packing bit-values

This example has both bit-boolean and bit-value data being packed together.

Here is how the XDR Language bitobject "AssemblyLineStatus" shown in Figure 3 would be packed:

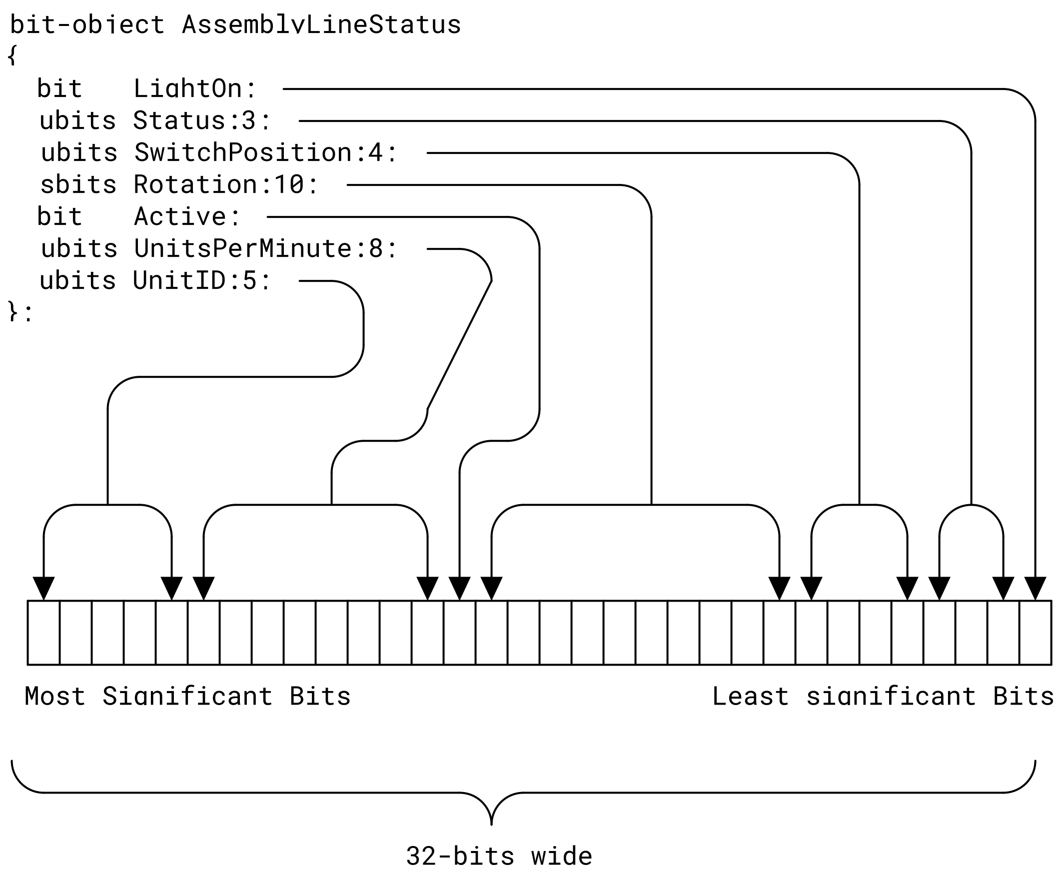

Figure 7: How Figure 3 would be packed.

3.2.1. Packing wider than 32-bit-values or spanning blocks

This shows how to pack values that are wider than 32-bits.

The values would be converted to network byte order like with [RFC4506], Section 4.5 Hyper Integer and Unsigned Hyper Integer values. With their values clamped to the width defined and sbits values having their sign at their own most significant bit position.

Here is how the XDR Language bitobject "Trajectory" shown in Figure 5 would be packed into three 32-bit XDR blocks:

```
bit-object Trajectory
{
  ubits Velocity:42;
  sbits VectorX:14;
  sbits VectorY:14;
  sbits VectorZ:14;
};
```

Figure 8: Trajectory - shown again

3.2.1.1. Most significant 32-bit block of Figure 8 "Trajectory"

- 11 unused bits set to zero (0).
- 14-bit "VectorZ:14"
- The most significant 7 bits of 14-bit "VectorY:14"

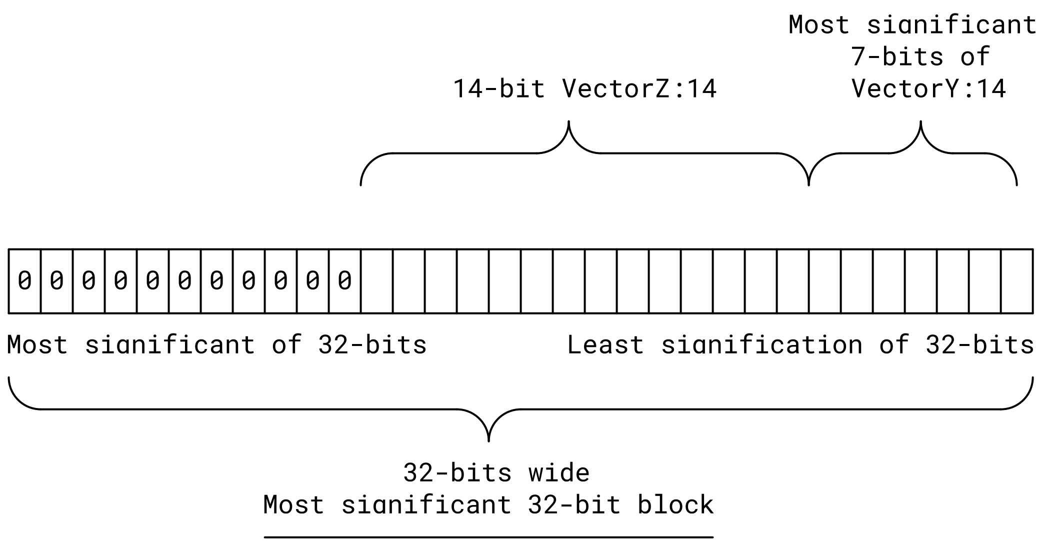

Figure 9: How Figure 8 "Trajectory" most signification 32-bit block would be packed.

3.2.1.2. How Figure 8 "Trajectory" middle signification 32-bit block would be packed.

The middle signification 32-bit block of "Trajectory" would have:

- The least significant 7 bits of 14-bit "VectorY:14.
- 14-bit "VectorX:14"
- The most significant 10 bits of 42-bit "Velocity:42"

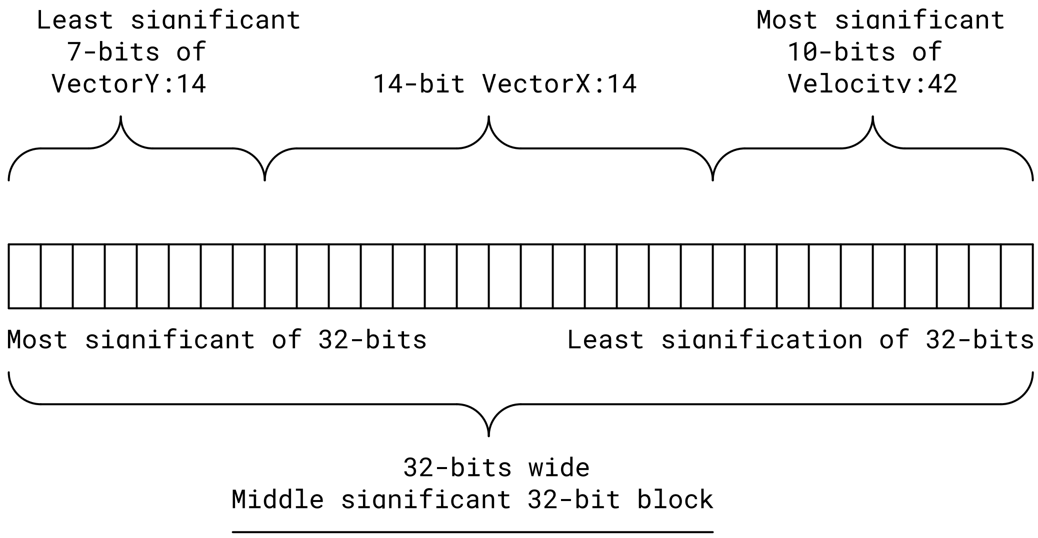

Figure 10: How Figure 8 "Trajectory" middle signification 32-bit block would be packed.

3.2.1.3. How Figure 8 "Trajectory" least signification 32-bit block would be packed.

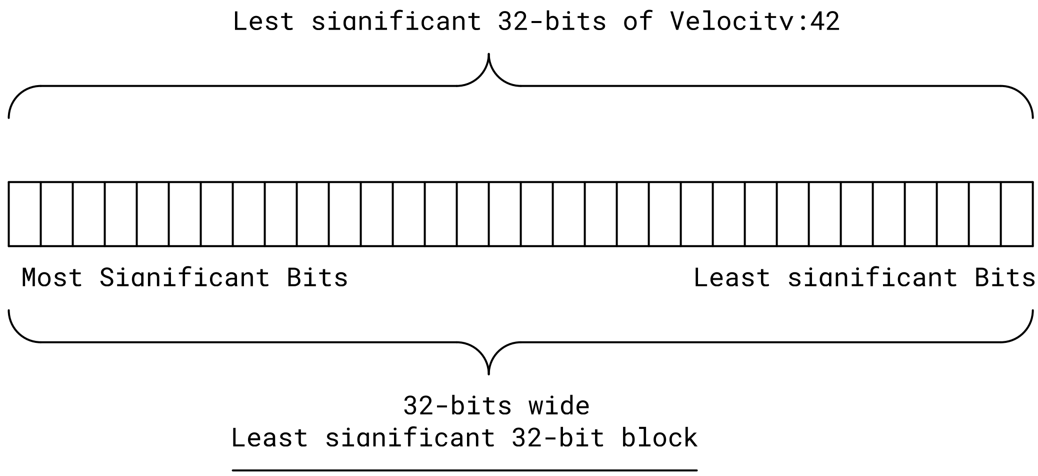

Figure 11: How Figure 8 "Trajectory" least signification 32-bit block would be packed.

4. Reducing Namespace Collision in Generated Code

When gathering specifications and definitions from multiple specifications it can be much more convenient to be able to name each identifier in a way the most resembles the original specification. However it can be confusing to uniquely identify them in generated code, and match them up to the XDR Language file that generated them.

For this reason a new 'namespace' declaration type is being defined.

```
namespace = 'namespace' identifier *( ':' identifier) ';'
```

Figure 12: namespace declaration type

A namespace may have several scope levels. Each scope name separated by a colon (:).

As each 'namespace' is encountered, all objects that follow will be in that scoped namespace. Zero or more 'namespaces' declarations may be in one XDR Language source.

Section 6.3 of [RFC4506] 'declaration' is modified to:

```
declaration:
           type-specifier identifier
         | type-specifier identifier "[" value "]"
         | type-specifier identifier "<" [ value ] ">"
         | "opaque" identifier "[" value "]"
         | "opaque" identifier "<" [ value ] ">"
         | "string" identifier "<" [ value ] ">"
         | type-specifier "*" identifier
         | "void"
         | namespace
```

Figure 13: Extended declaration ABNF

Examples:

```
namespace ietf:xdr:example_namespace;

namespace RiverExplorer:Phoenix:xdrgen;
```

Figure 14: namespace type-specifier

Multiple namespaces in one definition could look like this:

```
namespace MyCompany:LauchPad;

bitobject Status
{
  bit   OffLine;
  bit   LightOn;
  ubits Status:3;
  ubits SwitchPosition:4;
  sbits Rotation:10;
  bit   Active;
  ubits UnitsPerMinute:8;
  ubits UnitID:5;
};

namspace MyCompany:Projectile;

bitobject Status
{
  bit   OffLine;
  ubits Status:3;
  sbits Rotation:14;
  ubits Velocity:42;
  sbits VectorX:14;
  sbits VectorY:14;
  sbits VectorZ:14;
};
```

Figure 15: Namespace Example

The result would be two 'Status' data types. They are similar, and unique. Perhaps they came from different specifications. And each uniquely identifiable by their scope. And the variables produced could be referenced with:

- 'MyCompany:LaunchPad:Status'
- 'MyCompany:Projectile:Status'

This also makes it easier to identify an object in an XDR Section 4.15 of [RFC4506] union:

Multiple namespaces used in one definition could look like this:

```
enum Type = {
 LaunchPadType = 1,
 ProjectileType = 2
};

union ObjectStatus (Type WhichType)
{
  case LaunchPadType:
   MyCompany:LaunchPad:Status PadStatus;

  case ProjectilePadType:
   MyCompany:ProjectilePad:Status ProjectileStatus;

  default:
   void;
};
```

Figure 16: Namespace Example

5. IANA Considerations

This memo includes no request to IANA. [CHECK]

6. Security Considerations

This document should not affect the security of the Internet. [CHECK]

7. Normative References

[RFC4506] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, DOI 10.17487/RFC4506, May 2006, <https://www.rfc-editor.org/info/rfc4506>.

8. Informative References

[xdrgen] Royer, DM., "XDR Code Generator, Open Source", 2025, <https://github.com/RiverExplorer/Phoenix>.

Appendix A. Appendix 1 Full XDR Language Grammar

TODO

Acknowledgments

Contributors

Author's Address

Doug Royer
RiverExplorer LLC
848 N. Rainbow Blvd, Ste-1120
Las Vegas, Nevada 89107
United States of America
Phone: +1-208-806-1358
Email: DouglasRoyer@gmail.com
URI: https://DougRoyer.US