



UML to ProtoBuf Mapping Guidelines

TR-544 v1.0-info

February 22, 2018



Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Any marks and brands contained herein are the property of their respective owners.

Open Networking Foundation
1000 El Camino Real, Suite 100, Menlo Park, CA 94025
www.opennetworking.org

©2018 Open Networking Foundation. All rights reserved.

Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation, in the United States and/or in other countries. All other brands, products, or service names are or may be trademarks or service marks of, and are used to identify, products or services of their respective owners.

Important note

This Technical Recommendations has been approved by the OIMT Project TST but has not been approved by the ONF board. This Technical Recommendation has been approved under the ONF publishing guidelines for 'Informational' publications that allow Project technical steering teams (TSTs) to authorize publication of Informational documents. The designation of '-info' at the end of the document ID also reflects that the project team (not the ONF board) approved this TR.

Content

1	Introduction	5
2	References	5
3	Abbreviations	5
4	Overview	5
4.1	Documentation Overview	5
5	UML- Protobuf Mapping Guidelines	6
5.1	Mapping of Classes	7
5.1.1	Mapping of Inheritance	8
5.2	Mapping of Attributes	11
5.3	Mapping of Data Types	14
5.3.1	Generic Mapping of Primitive Data Types	14
5.3.2	Generic Mapping of Complex Data Types	15
5.3.3	Mapping of Common Primitive and Complex Data Types	18
5.3.4	Mapping of Enumeration Types	21
5.4	Mapping of Relationships	22
5.4.1	Mapping of Associations	22
5.4.2	Mapping of Association references	22
5.4.3	Mapping of Association containment	24
5.5	Mapping of Interfaces (grouping of operations)	27
5.6	Mapping of Operations	28
5.7	Mapping of Operation Parameters	30
5.8	Mapping of Notifications	32
5.9	Mapping of UML Packages	34
5.10	Retention of index numbers	36
5.10.1	Field Allocation Table	36
5.10.2	Change History Table	37
5.11	Defined Types (onf-types.proto)	37
5.12	CustomOptions (onf-descriptor.proto)	41
5.13	Hand Generated TAPI Example	44
5.13.1	TapiCommon.proto	45
5.13.2	TapiConnectivity.proto	47
5.14	ProtoBuf Notes	48
5.14.1	pros and cons to upgrade protobuf v3	48
5.14.2	Replacing 'extensions' in proto3	48
5.14.3	Timestamp	48
5.14.4	Proto3 Storing UUID	48
5.14.5	Representing Polymorphism	49
5.14.6	int vs sint vs uint	50

5.14.7 solutions to resolve enum field naming restriction	51
5.14.8 Can protobuf service method return primitive type?	51
5.14.9 Protobuf RPC Service method without parameters	51
5.14.10 Why required and optional is removed in Protocol Buffers 3.....	51
5.14.11 protobuf message holding reference to another message	52
5.14.12 Protocol buffer: does changing field name break the message?.....	52
5.14.13 Schema evolution in Avro, Protocol Buffers and Thrift	52
5.14.14 finally, a better protobuf ?	52
5.15 Installing ProtoBuf on Windows	53

1 Introduction

This Technical Recommendation defines the guidelines for a mapping from a protocol-neutral UML information model to the ProtoBuf schema language. The UML information model has to be defined based on the UML Modeling Guidelines defined in [1]. The ProtoBuf language is defined in [2].

2 References

- [1] ONF TR-514 “UML Modeling Guidelines 1.1”
(https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/UML_Modeling_Guidelines_Version_1-1.pdf)
- [2] ProtoBuf V2 Language Specification
<https://developers.google.com/protocol-buffers/docs/reference/overview>

3 Abbreviations

DS	Data Schema
IM	Information Model
NA	Not Applicable
ro	read only
rw	read write
UML	Unified Modeling Language

4 Overview

4.1 Documentation Overview

This document is part of a series of Technical Recommendations. The location of this document within the documentation architecture is shown in Figure 4.1 below:

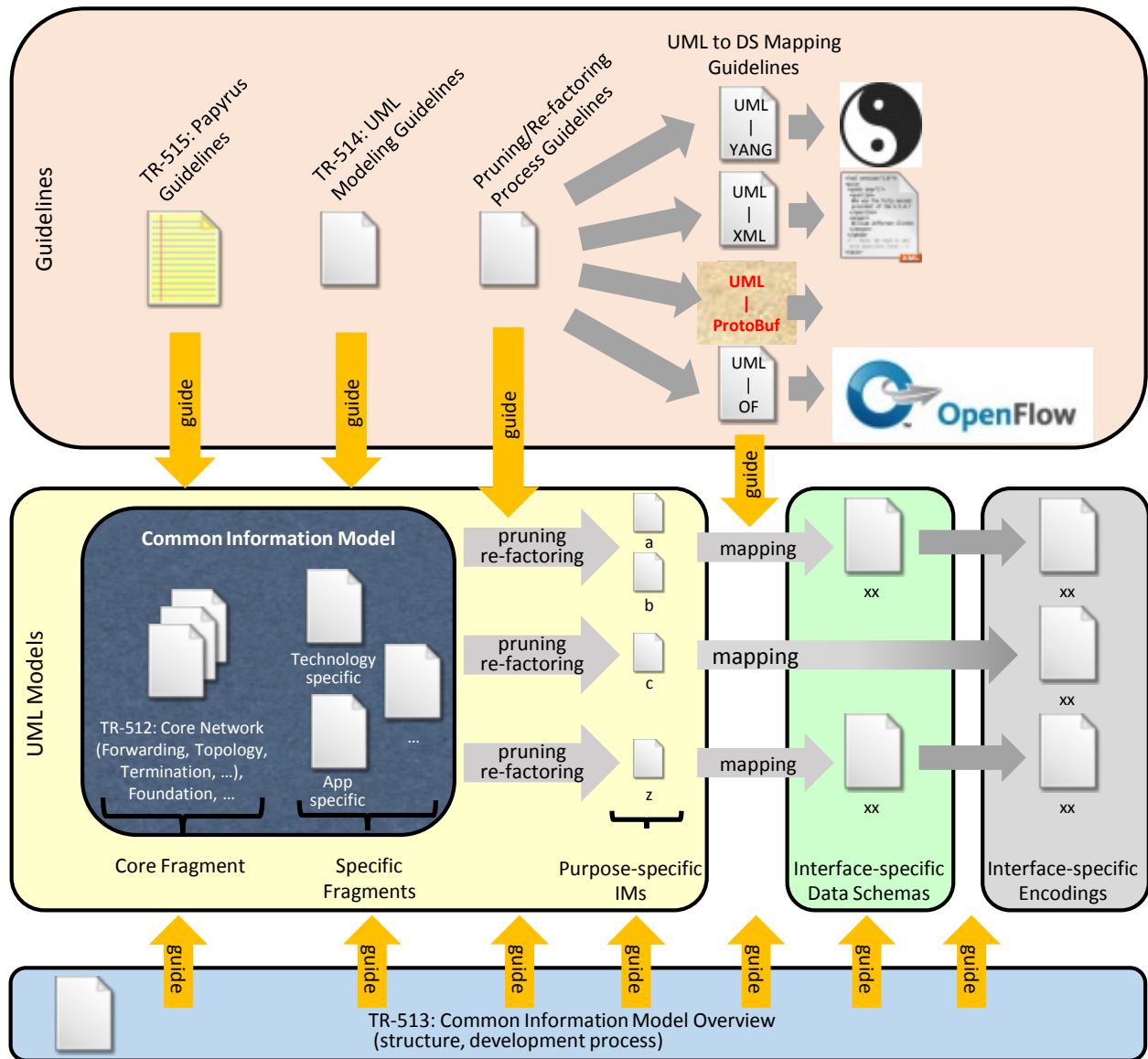


Figure 4.1: ONF Specification Architecture

5 UML- Protobuf Mapping Guidelines

The UML- Protobuf mapping rules are defined in table format and are structured based on the UML artifacts defined in [1]. Example mappings are shown below the mapping tables.

Open issues are either marked in yellow and/or by comments.

Note that at this stage we will generate Version 2 ProtoBuf syntax files (as requested by ONF CORD and ONOS) and will attempt to make it as compliant with version 3 as possible (not use any language options removed in version 3).

To ensure forward compatibility the following is **suggested** :

- 1) initially generate proto3 syntax
- 2) then add an option to generate proto3 or proto2 syntax (apart from changing the syntax statement, the only change needed should be to add “optional” before every field type that isn’t marked “repeated”

Note also that the intention is to build a generic set of mapping rules and hence a generic generator. The generator should work for any UML model in Papyrus and should also be useful for other standards bodies to use for their models.

Note also that when generating the options, suppression of any options with a “NA” value should be suppressed, to avoid clutter in the result.

5.1 Mapping of Classes

Table 5.1: Class Mapping
(Mappings required by currently used UML artifacts)

Class → message		
UML Artifact	ProtoBuf Schema Artifact	Comments
	option(uml_message_type) = ENTITY;	
documentation “Applied comments” (carried in XMI as “ownedComment”)	option (uml_message_description) = “My Description”;	Multiple “applied comments” defined in UML, need to be collapsed into a single description option statement.
Class Name	message name	
attributes	fields	See section 5.2
Generalization Class	option (uml_message_extends) = “GlobalClass”;	
abstract	option (uml_is_abstract) = false;	Don’t generate if false
isLeaf	option (uml_is_leaf) = true;	Don’t generate if false
InterfaceModel_Profile::objectCreationNotification [YES/NO/NA]	option (uml_object_creation_notification) = NO;	don’t generate NA option
InterfaceModel_Profile::objectDeletionNotification [YES/NO/NA]	option (uml_object_deletion_notification) = NO;	don’t generate NA option

Class → message		
UML Artifact	ProtoBuf Schema Artifact	Comments
OpenModel_Profile::«Reference»	option (uml_message_reference) = "From RFC1234";	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	option (uml_message_lifecycle_state) = DEPRECATED;	
Proxy Class. XOR OpenModel_Profile::«Choice»	???	
OpenModelClass::support	option (uml_message_support) = MANDATORY;	
OpenModelClass::condition	option (uml_message_condition) = "Always";	

Table 5.2: Class Mapping Example

<pre> «OpenModelClass» Tapi::Topology ├─ _linkRefList : Tapi::Link [*] ├─ _nodeRefList : Tapi::Node [*] ├─ <Generalization> GlobalClass ├─ <Substitution> Substitution └─ layerProtocolName : Tapi::LayerProtocolName [1..* </pre>	<pre> message Topology { option (uml_messagedescription) = "The ForwardingDomain (FD) object class models... "; option(uml_message_extends) = "GlobalClass"; option(uml_message_type) = "ENTITY"; GlobalClass globalClass = 1; repeated Link _linkRefList = 2; repeated Node _nodeRefList = 3; repeated LayerProtocolName layerProtocolName = 4; } </pre>
--	---

5.1.1 Mapping of Inheritance

ProtoBuf 3 has no support for polymorphism / inheritance. Version 2 had an “extend” option, but this was problematic (because of the explicit field numbering) and has been removed in version 3.

There are a number of possible ways of representing inheritance (polymorphism) in ProtoBuf :

1. reference **parent** definitions inside the **child** messages
2. reference **parent** definitions inside the **child** messages and **nest** the definitions
3. reference the **child** definitions in the **parent** using “anyone” keyword

- reference the **child** definitions in the **parent** using anyone and also **nest** the definitions

Options 3 and 4 invert the dependencies and hence can't be used for inheritance across modules.

Option 2 won't work across modules, preventing a consistent representation.

So option 1 is the recommended option.

We will use this simple example.

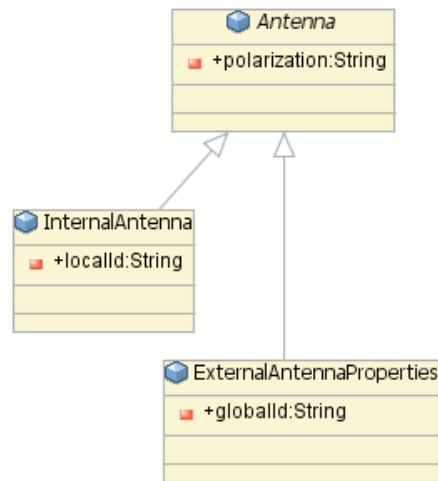


Figure 2

To simplify the code and make the mapping clear, we will not show any options statements.

```

import "onf/protobuf/onf-types.proto";

message Antenna {
  string polarization = 1;
}

message InternalAntenna {
  Antenna antenna = 1;
  Uuid localId = 2;
}

message ExternalAntennaProperties{
  Antenna antenna = 1;
  Uuid globalId = 2;
}
  
```

The parent class is just included as a field with the name being the parent type converted to lowerCamelCase (Antenna → antenna).

So when an `InternalAntenna` message is sent, it will include an `Antenna` message.

An option is added to show if any fields represent inheritance.

```
option (uml_message_extends) = "Antenna";
```

Because we could have a case where a class both subclasses from and composes a parent class (composite pattern) we will also mark the field that represents the subclassing with `(uml_field_extends) = true`.

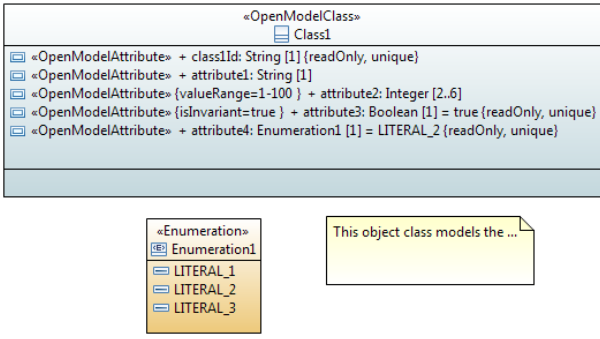
5.2 Mapping of Attributes

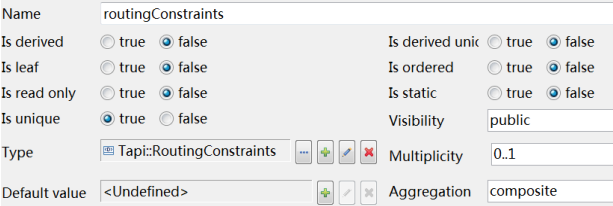
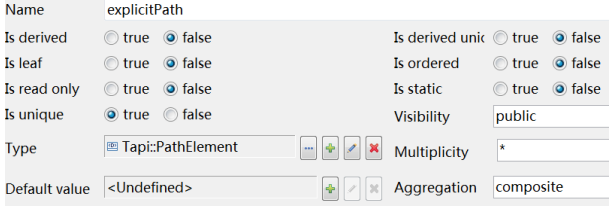
Table 5.3: Attribute Mapping
(Mappings required by currently used UML artifacts)

Attribute → field		
UML Artifact	ProtoBuf field Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	(uml_field_description) = “My Description”	Multiple “applied comments” defined in UML, need to be collapsed into a single description option statement.
Attribute name	name	
Attribute type= Common Primitive Types	When Multiplicity ≤ 1: mapped-primitive-type field-name = fieldNumber; When Multiplicity > 1: repeated mapped-primitive-type field-name = fieldNumber;	Common PrimitiveTypes as per section 5.3.3
Attribute type= Complex Data Type	When Multiplicity ≤ 1: message-type field-name = fieldNumber; When Multiplicity > 1: repeated message-type field-name = fieldNumber;	
Multiplicity (carried in XMI as lowerValue and upperValue)	lowerValue => (uml_min_items) = “1” upperValue => (uml_max_items) = “*”	
isOrdered	(uml_is_ordered_collection) = true	only generate for multiplicity > 1
isUnique	(uml_is_unique_collection) = false	only generate for multiplicity > 1
defaultValue	(uml_default_value) = “value”	If a default value exists and it is the desired value, the parameter does not have to be explicitly configured by the user. When the value of “defaultValue” is “NA”, the tool ignores it and doesn’t print out “default” substatement.

Attribute → field		
UML Artifact	ProtoBuf field Artifact	Comments
valueRange	For integer and number typed attributes: --> (uml_min_exclusive_value) = "value" (uml_max_exclusive_value) = "value"	When the value of "valueRange" is "null", "NA", "See data type", the tool ignores it and doesn't print out "range" substatement.
OpenModelAttribute.support	(uml_field_support) = MANDATORY	
OpenModelAttribute::partOfObjectKey >0	(uml_part_of_object_key) = 2	Don't generate if "partOfObjectKey"=0
OpenModelAttribute::isInvariant	(uml_is_invariant) = false	
OpenModelAttribute::unsigned	(uml_is_unsigned) = false	Don't generate if false
OpenModelAttribute::counter	(uml_counter_type) = COUNTER	Don't generate NA option
InterfaceModelAttribute::unit	(uml_units) = "metres"	
InterfaceModelAttribute::writeAllowed	(uml_write_allowed) = UPDATE_ONLY	
InterfaceModelAttribute::attributeValueChangeNotification	(uml_field_value_change_notification) = YES	
InterfaceModel_Profile::bitLength	(uml_bit_length) = LENGTH_8_BIT	Don't generate NA option
InterfaceModel_Profile::encoding	(uml_string_encoding) = BASE_64	Don't generate NA option
OpenModel_Profile::«PassedByReference»	in the generated structure	
OpenModel_Profile::«Reference»	(uml_field_reference) = "From RFC1234"	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	(uml_field_lifecycle_state) = PRELIMINARY	
OpenModelAttribute::support	(uml_field_support) = MANDATORY	
OpenModelAttribute::condition	(uml_field_condition) = "Always"	

Table 5.4: Attribute Type Mapping Example

<p style="text-align: center;">Attribute type= Common Primitive Types:</p>  <p>(note readOnly no longer used. Also for classId the multiplicity is 1, so “unique” is not applicable.)</p>	<pre> enum Enumeration1 { option (uml_added_prefix) = "ENUMERATION1_"; ENUMERATION1_LITERAL_1 = 0; ENUMERATION1_LITERAL_2 = 1; ENUMERATION1_LITERAL_3 = 2; } message Class1 { option (uml_message_description) = "This class models the ..."; string classId = 1 [(uml_min_items) = "1", (uml_max_items) = "1"]; string attribute1 = 2 [(uml_min_items) = "1", (uml_max_items) = "1"]; repeated int32 attribute2 = 3 [(uml_min_items) = "2", (uml_max_items) = "6", (uml_min_exclusive_value) = "1", (uml_max_exclusive_value) = "100"]; bool attribute3 = 4 [(uml_min_items) = "1", (uml_max_items) = "1", (uml_default_value) = "true", (uml_is_invariant) = true]; Enumeration1 attribute4 = 5 [(uml_min_items) = "1", (uml_max_items) = "1", (uml_default_value) = "ENUMERATION1_LITERAL_2"]; } </pre>
---	--

<p>Attribute type= Complex Data Type (Multiplicity\leq1): (association end)</p>  <p>Name: routingConstraints</p> <p>Is derived: <input type="radio"/> true <input checked="" type="radio"/> false</p> <p>Is leaf: <input type="radio"/> true <input checked="" type="radio"/> false</p> <p>Is read only: <input type="radio"/> true <input checked="" type="radio"/> false</p> <p>Is unique: <input checked="" type="radio"/> true <input type="radio"/> false</p> <p>Type: Tapi::RoutingConstraints</p> <p>Multiplicity: 0..1</p> <p>Default value: <Undefined></p> <p>Aggregation: composite</p>	<p>RoutingConstraints routingConstraints = 1;</p>
<p>Attribute type= Complex Data Type (Multiplicity$>$1): (association end)</p>  <p>Name: explicitPath</p> <p>Is derived: <input type="radio"/> true <input checked="" type="radio"/> false</p> <p>Is leaf: <input type="radio"/> true <input checked="" type="radio"/> false</p> <p>Is read only: <input type="radio"/> true <input checked="" type="radio"/> false</p> <p>Is unique: <input checked="" type="radio"/> true <input type="radio"/> false</p> <p>Type: Tapi::PathElement</p> <p>Multiplicity: *</p> <p>Default value: <Undefined></p> <p>Aggregation: composite</p>	<p>repeated PathElement explicitPath = 1;</p>

5.3 Mapping of Data Types

Various kinds of data types are defined in UML:

- Primitive Data Types (not further structured; e.g., Integer, String, Boolean)
- Complex Data Types (containing attributes; e.g., Host which combines ipAddress and domainName)
- Enumerations

They are used as the type definition of attributes and parameters.

5.3.1 Generic Mapping of Primitive Data Types

Currently the primitive datatypes are represented in a static file called onf-types.proto.

We need to determine if it makes sense to generate this file from the profile or not.

5.3.2 Generic Mapping of Complex Data Types

Table 5.5: Complex Data Type Mapping

Complex Data Type → message		
UML Artifact	ProtoBuf Message Artifact	Comments
	option(uml_message_type) = DATATYPE;	
documentation “Applied comments” (carried in XMI as “ownedComment”)	option (uml_message_description) = “My Description”;	Multiple “applied comments” defined in UML, need to be collapsed into a single description option statement.
Data Type Name	message name	
attributes	fields	See section 5.2
XOR OpenModel_Profile::«Choice»	???	
OpenModel_Profile::«Reference»	option (uml_field_reference) = “From RFC1234”;	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	option (uml_message_lifecycle_state) = DEPRECATED;	

Table 5.6: Complex Data Type Mapping Example

<pre> classDiagram class Capacity { colorAware : Boolean [0..1] committedBurstSize : Integer [0..1] committedInformationRate : Integer couplingFlag : Boolean [0..1] packetBwProfileType : BandwidthProfileType peakBurstSize : Integer [0..1] peakInformationRate : Integer [0..1] totalSize : Integer } </pre>	<pre> message Capacity { option (description) = "Information on capacity of a particular TopologicalEntity."; string committedInformationRate = 1; int64 peakBurstSize = 2; int64 totalSize = 3 [(uml_field_description) = "Total capacity of the TopologicalEntity in MB/s"]; int64 committedBurstSize = 4; BandwidthProfileType packetBwProfileType = 5; int64 peakInformationRate = 6; bool couplingFlag = 7 [(uml_default_value) = false]; bool colorAware = 8 [(uml_default_value) = false]; } </pre>
--	--

Note that we assume that all datatypes are concrete and that there is no subclassing.

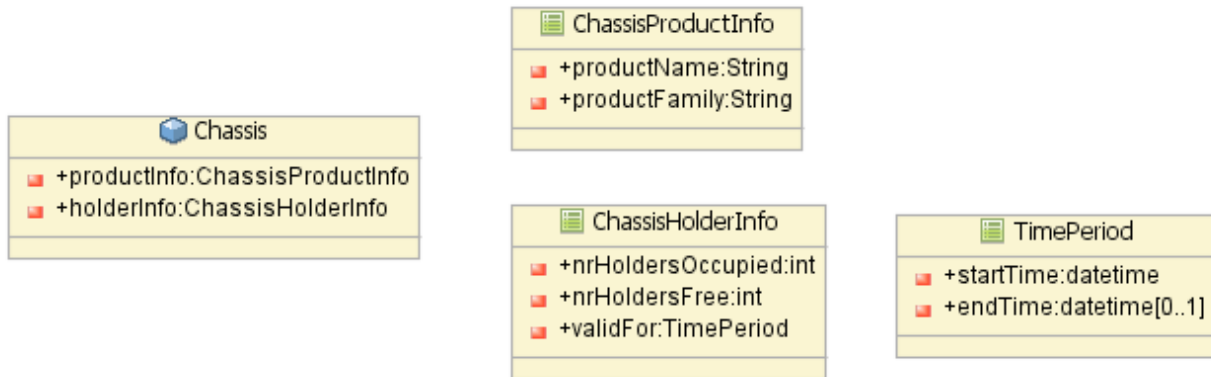


Figure 3

```

import "google/protobuf/timestamp.proto";

message Chassis {
    ChassisProductInfo productInfo = 1;
    ChassisHolderInfo holderInfo = 2;
}

message ChassisProductInfo {
    string productName = 1;
    string productFamily = 2;
}

message ChassisHolderInfo {
    int32 nrHoldersOccupied = 1;
    int32 nrHoldersFree = 2;
    TimePeriod validFor = 3;
}

message TimePeriod {
    google.protobuf.Timestamp startTime = 1;
    google.protobuf.Timestamp endTime = 2;
}
  
```

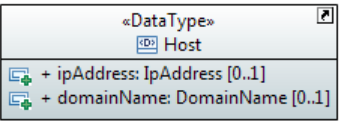
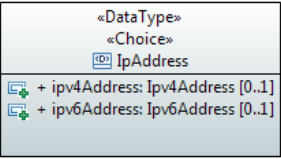
5.3.3 Mapping of Common Primitive and Complex Data Types

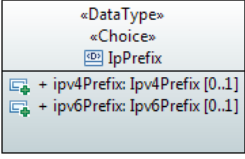
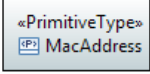
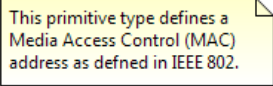
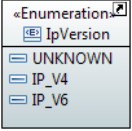
A list of generic UML data types is defined in a “CommonDataTypes” Model Library. This library is imported to every UML model to make these data types available for the model designer.

Note that protobuf also defines wrapped primitives in wrappers.proto.

Table 5.7: Common Primitive and Complex Data Type Mapping

UML CommonDataTypes → ProtoBuf Schema Types		
UML Artifact	ProtoBuf Schema Artifact(type/format)	Comments
Boolean	bool	
String	string	
«LENGTH_8_BIT» Integer	onf.protobuf.Int8	defined in onf-types.proto
«LENGTH_16_BIT» Integer	onf.protobuf.Int16	defined in onf-types.proto
«LENGTH_32_BIT» Integer	int32	Signed 32 bits
«LENGTH_64_BIT» Integer	int64	Signed 64 bits
Integer		
«UNSIGNED, LENGTH_8_BIT» Integer	onf.protobuf.Uint8	defined in onf-types.proto
«UNSIGNED, LENGTH_16_BIT» Integer	onf.protobuf.Uint16	defined in onf-types.proto
«UNSIGNED, LENGTH_32_BIT» Integer	uint32	
«UNSIGNED, LENGTH_64_BIT» Integer	uint64	
	fixed64	
	fixed32	
	bytes	byte array
	sfixed32	zig zag encoded
	sfixed64	zig zag encoded
	sint32	zig zag encoded
	sint64	zig zag encoded
Real (Not used so far. See also float and double below.)		

UML CommonDataTypes → ProtoBuf Schema Types		
UML Artifact	ProtoBuf Schema Artifact(type/format)	Comments
«LENGTH_32_BIT» Real (float)	float	
«LENGTH_64_BIT» Real (double)	double	
Unlimited Natural		currently not used
«COUNTER, LENGTH_32_BIT» Integer	onf.protobuf.Counter32	defined in onf-types.proto
«COUNTER, LENGTH_64_BIT» Integer	onf.protobuf.Counter64	defined in onf-types.proto
«GAUGE, LENGTH_32_BIT» Integer	onf.protobuf.Gauge32	defined in onf-types.proto
«GAUGE, LENGTH_64_BIT» Integer	onf.protobuf.Gauge64	defined in onf-types.proto
«ZERO_COUNTER, LENGTH_32_BIT» Integer	onf.protobuf.ZeroCounter32	defined in onf-types.proto
DateTime	google.protobuf.Timestamp	As defined by date-time - RFC3339 Requires an import statement import "google/protobuf/timestamp.proto";
Domain Name and URI related Types		
DomainName	onf.protobuf.DomainName	defined in onf-types.proto
	onf.protobuf.Host	defined in onf-types.proto
Uri	onf.protobuf.Uri	defined in onf-types.proto
Address related Types		
	onf.protobuf.IpAddress	defined in onf-types.proto
Ipv4Address	onf.protobuf.Ipv4Address	defined in onf-types.proto
Ipv6Address	onf.protobuf.Ipv6Address	defined in onf-types.proto
	onf.protobuf.IpAddressNoZone	defined in onf-types.proto
Ipv4AddressNoZone	onf.protobuf.	defined in onf-types.proto

UML CommonDataTypes → ProtoBuf Schema Types		
UML Artifact	ProtoBuf Schema Artifact(type/format)	Comments
	Ipv4AddressNoZone	
Ipv6AddressNoZone	onf.protobuf.Ipv6AddressNoZone	defined in onf-types.proto
	onf.protobuf.IpPrefix	defined in onf-types.proto
Ipv4Prefix	onf.protobuf.Ipv4Prefix	defined in onf-types.proto
Ipv6Prefix	onf.protobuf.Ipv6Prefix	defined in onf-types.proto
MacAddress  	onf.protobuf.MacAddress	defined in onf-types.proto
Protocol Field related Types		
Dscp	onf.protobuf.Dscp	defined in onf-types.proto
	onf.protobuf.IpVersionEnum	defined in onf-types.proto
IpV6FlowLabel		
PortNumber	onf.protobuf.PortNumber	defined in onf-types.proto
String related Types		
DottedQuad	onf.protobuf.DottedQuad	defined in onf-types.proto
«OctetEncoded» String		
HexString «HexEncoded» String		
«Base64Encoded» String		
Uuid	onf.protobuf.common.Uuid	defined in onf-types.proto
	google.protobuf.Duration	Requires an import statement

UML CommonDataTypes → ProtoBuf Schema Types		
UML Artifact	ProtoBuf Schema Artifact(type/format)	Comments
		import "google/protobuf/duration.proto";

5.3.4 Mapping of Enumeration Types

ProtoBuf allows ‘localized’ Enums inside of messages, but we will just generate ‘global’ Enums. “The first enum value must be zero in proto3.”

Note that in ProtoBuf, the default value of a field of type Enum is the literal with value = 0, therefore it is considered good practice to make the default value UNKNOWN or NA or similar.

Note that Enum literals need to be globally unique in ProtoBuf while they only need to be unique within the Enum in UML. To allow for this, the generator will need to add a unique prefix to every literal. To help in knowing what the prefix is, we will add an option that records the prefix. The prefix should be the Enum name (with the suffix Enum removed if it exists) and then the remaining name should be converted from camel case to uppercase with underscores and have a trailing underscore added. For example, an Enum named “PortDirectionEnum” would generate a prefix of "PORT_DIRECTION_". Note also that any default values in the model will need to be mapped to include the prefix.

Table 5.8: Enumeration Type Mapping
(Mappings required by currently used UML artifacts)

Fixed Enumeration Type → “enum” statement		
UML Artifact	ProtoBuf Schema Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	option (uml_enum_description) = “My Description”;	Multiple “applied comments” defined in UML, need to be collapsed into a single description option statement.
literal name	enumField within enum	
«Reference»	option (uml_enum_reference) = “From RFC1234”;	
«Example»	Ignore Example elements and all composed parts	
lifecycleState	option (uml_enum_lifecycle_state) = DEPRECATED;	

5.4 Mapping of Relationships

5.4.1 Mapping of Associations

All associations (i.e., pointers, composition aggregations and shared aggregations) are per default passed by reference (i.e., contain only the reference (name, identifier, address) to the referred instance(s) when being transferred across the interface); **except** the «StrictComposite» and «ExtendedComposite» associations which are always passed by value (i.e., contain the complete information of the instance(s) when being transferred across the interface).

This lead to the following 3 kinds of association scenarios:

1. Pointers, composition aggregations and shared aggregations which are passed by reference
2. «StrictComposite» associations which are passed by value
3. «ExtendedComposite» associations which can also be somehow treated as passed by value.

5.4.2 Mapping of Association references

ProtoBuf has no concept of reference, so we need to rely on the identifiers defined in our model. Coupled with the lack of polymorphism / inheritance support, this creates an issue that means that strongly typed identifiers are not possible.

The only solution is to have a single explicit identifier of the one type, with the one name. The only sensible identifier is to use a UUID, so every Entity needs to have or inherit an identifier called “id” of type Uuid.

The alternate key (partOfObjectKey) CAN'T be used in a reference.

Now we can map polymorphic references to ProtoBuf.

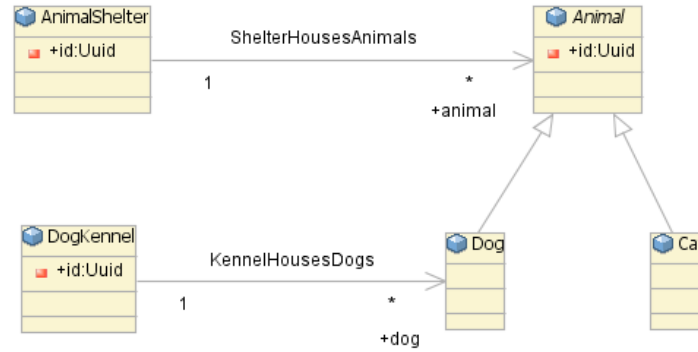


Figure 4

```

import "onf/protobuf/onf-types.proto";
message Animal {
  option (uml_is_abstract) = true;
  Uuid id = 1;
  // ... other attributes
}

message Cat {
  option (uml_message_extends) = "Animal";
  Animal animal = 1 [
    (uml_field_extends) = true
  ];
  // ... other attributes
}

message Dog {
  option (uml_message_extends) = "Animal";
  Animal animal = 1 [
    (uml_field_extends) = true
  ];
  // ... other attributes
}

// shelter for all animals
message AnimalShelter {
  // shelter references many animals via their identifiers
  repeated Uuid animalId = 1 [
    (uml_references) = "Animal"
  ];
  // ... other attributes
}

// kennel for dogs only
// not possible to restrict, so add (another) option to show the valid type
message Dogkennel {
  // kennel references many Dogs via their identifiers
  repeated Uuid dogId = 1 [
    (uml_references) = "Dog"
  ];
  // ... other attributes
}

```

To show that the association match is done via id, we have added the suffix **Id** to the field name. Also note the “`uml_references`” option that slows us to check that the referenced messages are of the correct type.

5.4.3 Mapping of Association containment

For containment, we will use the same representation as for inheritance.

We will use this simple example.

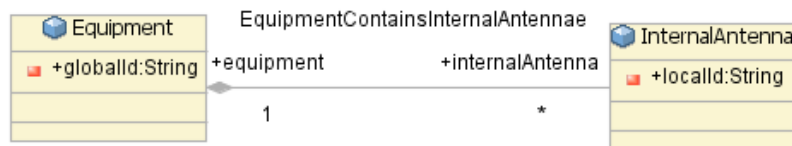


Figure 5

```

import "onf/protobuf/onf-types.proto";

message Equipment {
  Uuid globalId = 1;
  repeated InternalAntenna _internalAntenna = 2;
}

message InternalAntenna {
  Uuid localId = 2;
}
  
```

So when an `Equipment` message is sent, it will include an array of `Antenna` messages.

Note that the representation is independent of the composing association navigabilities.

Note that this also works for self joins.

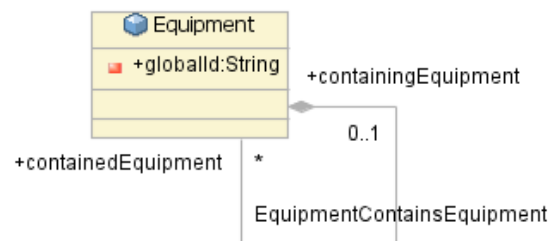


Figure 6

```

message Equipment {
  string globalId = 1;
  repeated Equipment containedEquipment = 2;
}
  
```


It also works with inheritance – InternalAntenna is composed by Equipment and also extends Antenna.

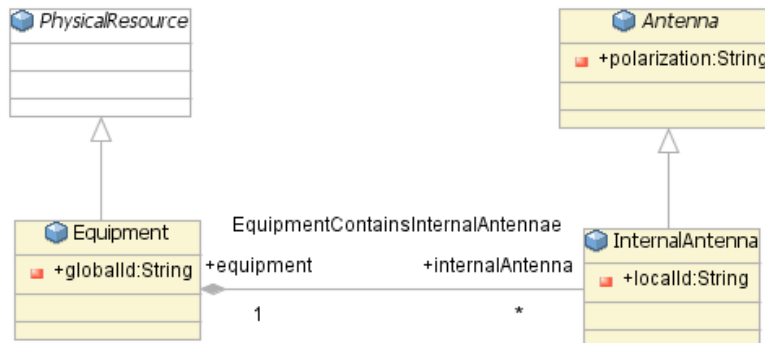


Figure 7

```

import "onf/protobuf/onf-types.proto";

message Equipment {
  PhysicalResource physicalResource = 1 [
    (uml_field_extends) = true
  ];
  Uuid globalId = 2;
  repeated InternalAntenna _internalAntenna = 3;
}

message Antenna {
  string polarization = 1;
}

message InternalAntenna {
  Antenna antenna = 1 [
    (uml_field_extends) = true
  ];
  Uuid localId = 2;
}
  
```

In the examples below, we will not show the fields for any of the attributes to help focus on the generated association ends.

Table 5.9: Association Mapping Examples

	<pre>import "onf/protobuf/onf-types.proto"; message ClassC { repeated Uuid _classDId = 1 [(uml_references) = "ClassD"]; } message ClassD { }</pre>
	<pre>message ClassC { repeated ClassD _classD = 1; } message ClassD { }</pre>
	<pre>import "onf/protobuf/onf-types.proto"; message ClassA { Uuid _classBId = 1 [(uml_references) = "ClassB"]; } message ClassB { }</pre>

5.5 Mapping of Interfaces (grouping of operations)

Table 5.10: UML Interface Mapping

UML Interface → Service		
UML Artifact	ProtoBuf Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	option (uml_service_description) = “My Description”;	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.
OpenModel_Profile::«Reference»	option (uml_service_reference) = “From RFC1234”;	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	option (uml_service_lifecycle_state) = DEPRECATED;	
OpenModelInterface::support	option (uml_service_support) = MANDATORY;	Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported.
OpenModelInterface::condition	option (uml_message_condition) = “Always”;	

5.6 Mapping of Operations

Protobuf has a very simple method syntax – every method must take exactly one message and return exactly one message. So if the method takes multiple parameters, we will need to wrap them in a message. If the method has no input parameters or returns no value, then we will need to create an empty message.

All parameters marked as in and inout are added to the request message.

All parameters marked as out and inout are added to the response message.

Table 5.11: Operation Mapping

Operation →ProtoBuf method		
UML Artifact	ProtoBuf Method Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	option (uml_method_description) = “My Description”;	Multiple “applied comments” defined in UML, need to be collapsed into a single description option statement.
pre-condition	option (uml_method_pre_condition) = “A>1”;	
post-condition	option (uml_method_post_condition) = “B<6”;	
input parameter	need to group these into a request message with option(uml_message_type) = METHOD_REQUEST; The message name should be the interface name + the method name + “Request”	
output parameter	need to group these into a response message with option(uml_message_type) = METHOD_RESPONSE; The message name should be the interface name + the method name + “Response”	

Operation →ProtoBuf method		
UML Artifact	ProtoBuf Method Artifact	Comments
operation exceptions Internal Error Unable to Comply Comm Loss Invalid Input Not Implemented Duplicate Entity Not Found Object In Use Capacity Exceeded Not In Valid State Access Denied	<code>option (uml_method_exception) = "Internal Error";</code>	
OpenModelOperation:: isOperationIdempotent	<code>option (uml_method_is_idempotent) = true;</code>	
OpenModelOperation:: isAtomic	<code>option (uml_method_is_atomic) = true;</code>	
OpenModel_Profile::«Reference»	<code>option (uml_method_reference) = "From RFC1234";</code>	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	<code>option (uml_method_lifecycle_state) = DEPRECATED;</code>	
OpenModelOperation::support	<code>option (uml_method_support) = MANDATORY;</code>	
OpenModelOperation::condition	<code>option (uml_method_condition) = "Always";</code>	

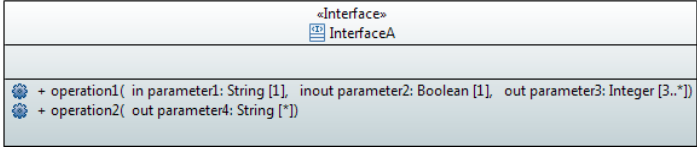
5.7 Mapping of Operation Parameters

Because the parameters are mapped to fields in a message, then the definitions as per fields apply.

Table 5.12: Parameter Mapping

Operation Parameters → “input” substatement or “output” substatement		
UML Artifact	YANG Artifact	Comments
documentation “Applied comments” (carried in XMI as “ownedComment”)	as per field	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.
direction	not needed as split into request and response messages	
type	as per field	
isOrdered		
multiplicity		
defaultValue		
OpenModelParameter::valueRange		
InterfaceModel_Profile::passedByReference		convert the field into a reference, add a suffix “Id” to the name and change the type to Uuid
OpenModel_Profile::«Reference»	as per field	
OpenModel_Profile::«Example»	as per field	
OpenModel_Profile::lifecycleState	as per field	
OpenModelParameter::support	as per field	Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported.
OpenModelParameter::condition		
XOR	??	
error notification?		
complex parameter		

Table 5.13: Interface/Operation/Parameter Mapping Example

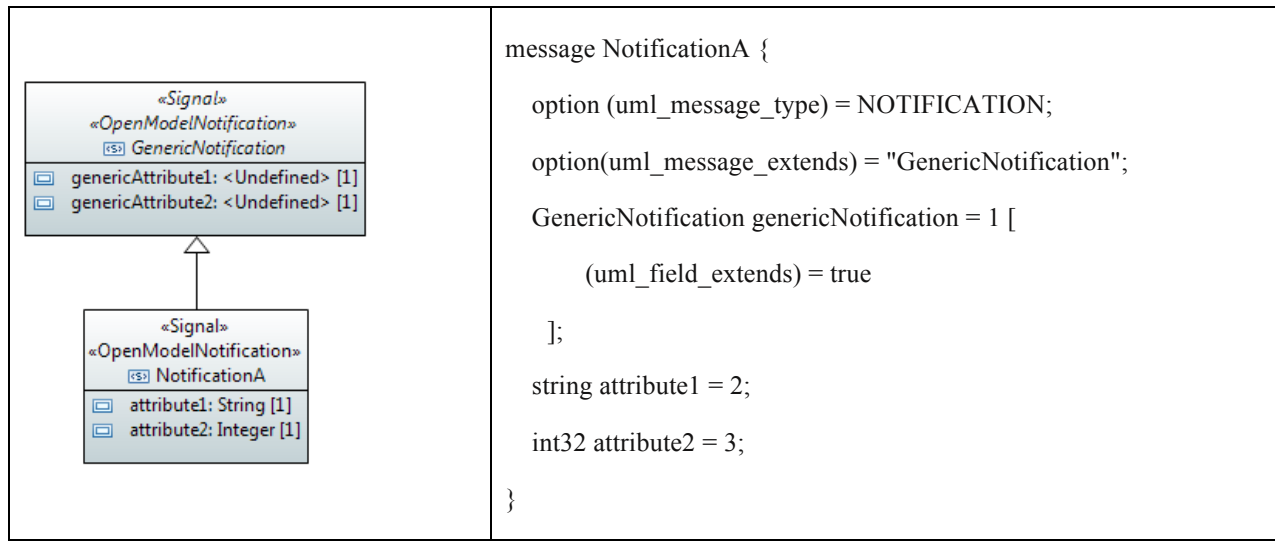
 <pre> classDiagram class InterfaceA { <<interface>> + operation1(in parameter1: String [1], inout parameter2: Boolean [1], out parameter3: Integer [3..*]) + operation2(out parameter4: String [1]) } </pre> <p>The diagram shows an interface named InterfaceA. It has two operations: operation1 and operation2. Operation1 has three parameters: parameter1 (String, 1), parameter2 (Boolean, 1), and parameter3 (Integer, 3..*). Operation2 has one parameter: parameter4 (String, 1).</p>	<pre> message InterfaceAOperation1Request { string parameter1 = 1; bool parameter2 = 2; } message InterfaceAOperation1Response { bool parameter2 = 1; int32 parameter3 = 2 [(uml_min_items) = "3", (uml_max_items) = "*"]; } message InterfaceAOperation2Request { } message InterfaceAOperation2Response { repeated string parameter4 = 1; } service InterfaceA { rpc operation1(InterfaceAOperation1Request) returns(InterfaceAOperation1Response); rpc operation2(InterfaceAOperation2Request) returns(InterfaceAOperation2Response); } </pre>
--	---

5.8 Mapping of Notifications

Table 5.14: Notification Mapping

Signal → Message		
UML Artifact	ProtoBuf Artifact	Comments
	option(uml_message_type) = NOTIFICATION;	
documentation “Applied comments” (carried in XMI as “ownedComment”)	option (uml_message_description) = “My Description”;	Multiple “applied comments” defined in UML, need to be collapsed into a single “description” substatement.
OpenModel_Profile::«Reference»	option (uml_message_reference) = “From RFC1234”;	
OpenModel_Profile::«Example»	Ignore Example elements and all composed parts	
OpenModel_Profile::lifecycleState	option (uml_message_lifecycle_state) = DEPRECATED;	
OpenModelNotification::triggerConditionList	option (uml_notification_trigger_conditions) = “a condition”;	
OpenModelNotification::support	option (uml_message_support) = MANDATORY;	Support and condition belong together. If the “support” is conditional, then the “condition” explains the conditions under which the class has to be supported.
OpenModelNotification::condition		
Proxy Class: XOR:	??	
attributes	fields	
complex attribute	fields	

Table 5.15: Notification Mapping Example



5.9 Mapping of UML Packages





The mapping tool shall generate a ProtoBuf file per UML model.

According to the UML Modeling Guidelines [1], each UML model is basically structured into the following packages:

- ▢ Associations
- ▢ Diagrams
- ▢ Imports
- ▢ Interfaces
- ▢ Notifications
- ▢ ObjectClasses
- ▢ Rules
- ▢ TypeDefinitions

Figure 5.8: Pre-defined Packages in a UML Module

Table 5.16: UML Packages Mapping Example

 TypeDefinitions	<pre>// ***** // type-definitions // *****</pre>
 ObjectClasses	<pre>// ***** // object-classes // *****</pre>
 Interfaces	<pre>// ***** // interfaces // *****</pre>
 Notifications	<pre>// ***** // notifications // *****</pre>

The generator shall generate at the top of each ProtoBuf file :

- Copyright and license text
- generator and model version information
- A syntax statement for version 3
- A package statement
- Any required import statements

For example

```
// (c) 2017 Open Networking Foundation. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// TapiModule-Interfaces-CoreModelAPI generated from UML
// generator version 0.0.1
// TapiModule-Interfaces-CoreModelAPI
// model version 1.0.0
//

syntax = "proto3";
package onf.tapi.CoreModelAPI;

import "google/protobuf/timestamp.proto";
import "onf/protobuf/onf-descriptor.proto";
import "onf/protobuf/onf-types.proto";
```

5.10 Retention of index numbers.

ProtoBuf uses unique reference numbers for each field in a message.

It assumes that someone will be editing the fields by hand.

Note also that “Field numbers cannot be greater than 536,870,911” (29 bits).

From the language reference : “If you want your new buffers to be backwards-compatible, and your old buffers to be forward-compatible – and you almost certainly do want this – then there are some rules you need to follow. In the new version of the protocol buffer:

- **you must not change the tag numbers of any existing fields.**
- ~~you must not add or delete any required fields.~~ (not applicable to proto3)
- you *may* delete optional or repeated fields.
- you *may* add new optional or repeated fields **but you must use fresh tag numbers** (i.e. tag numbers that were never used in this protocol buffer, not even by deleted fields).”

Note that some field types are compatible – this means you can change a field from one of these types to another without breaking forwards or backwards compatibility. See the protobuf3 language guide for more details.

Note that any numbers that are freed by removing a field should be marked as reserved in the generated files (e.g.

```
reserved 2, 15, 9 to 11; //don't reuse these numbers
reserved "foo", "bar"; // don't reuse these names ).
```

To comply with the 2 rules in red, we will need to store the existing formal release usages.

I **suggest** storing the records in a text format so they can be hand modified if required.

Also I **suggest** storing a changelog file to help in tracking down any issues.

Records are just appended to the changelog file in date order and it isn't read by the generator.

This document deliberately doesn't choose a file format; CSV, tab separated, JSON, YAML, XML ... could all be suitable.

5.10.1 Field Allocation Table

State could be :

- USED – number in use
- RESERVED – number not in use and was never used but can't be used
- REMOVED – number not in use but was used before and can't be used

Path	Module	Artefact	Field	field Number	Type	State	model release created	model release Last Updated
org.onf.tapi	CoreModelAPI	Link	name	3	string	USED	1.1.3	1.1.3
org.onf.tapi	CoreModelAPI	LinkStateEnum	OCCUPIED	584	literal	REMOVED	1.1.0	1.1.2

5.10.2 Change History Table

Change could be :

- CREATED
- UPDATED (also list change – “type changed from string to int32”)
- DELETED

Path	Module	Artefact	Field	field Number	change	model release version	generator version	timestamp
org.onf.tapi	CoreModelAPI	Link	org.onf.tapi	3	ADDED	1.1.3	0.1.2	1-jan-2018
org.onf.tapi	CoreModelAPI	LinkStateEnum	OCCUPIED	384	REMOVED	1.1.2	0.1.3	1-jan-2018

5.11 Defined Types (onf-types.proto)

There are a number of types defined in the ONF UML profile that need to be supported.

They may be able to be generated from the profile or could be added as a hand maintained file which is copied to an output folder.

```
// (c) 2017 Open Networking Foundation. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// common types for the ONF model generated files

syntax = "proto3";

package onf.protobuf;

import "onf/protobuf/onf-descriptor.proto";

message Int8 {
    uint32 counter32 = 1 [
        (uml_is_unsigned) = false,
```

```

        (uml_bit_length) = LENGTH_8_BIT
    ];
}

message Int16 {
    uint32 counter32 = 1 [
        (uml_is_unsigned) = false,
        (uml_bit_length) = LENGTH_16_BIT
    ];
}

message Uint8 {
    uint32 counter32 = 1 [
        (uml_is_unsigned) = true,
        (uml_bit_length) = LENGTH_8_BIT
    ];
}

message Uint16 {
    uint32 counter32 = 1 [
        (uml_is_unsigned) = true,
        (uml_bit_length) = LENGTH_16_BIT
    ];
}

//=====
message Counter32 {
    uint32 counter32 = 1 [
        (uml_counter_type) = COUNTER
    ];
}

message Counter64 {
    uint64 counter64 = 1 [
        (uml_counter_type) = COUNTER
    ];
}

message Gauge32 {
    uint32 gauge32 = 1 [
        (uml_counter_type) = GAUGE
    ];
}

message Gauge64 {
    uint64 gauge64 = 1 [
        (uml_counter_type) = GAUGE
    ];
}

message ZeroCounter32 {
    uint32 counter32 = 1 [
        (uml_counter_type) = ZERO_COUNTER
    ];
}

//=====
message Uuid {
    string uuid = 1;
}

message DottedQuad {
    string dottedQuad = 1;
}

```

```

}

message MacAddress {
  string macAddress = 1;
}

message Uri {
  string uri = 1;
}

message ObjectIdentifier {
  string objectIdentifier = 1;
}

message ObjectIdentifier128 {
  string objectIdentifier128 = 1;
}

message Dscp {
  string dscp = 1;
}

message IPv6FlowLabel {
  string ipv6FlowLabel = 1;
}

message PortNumber {
  uint32 portNumber = 1;
}

enum IpVersionEnum {
  option (uml_added_prefix) = "IP_VERSION_";

  IP_VERSION_UNKNOWN = 0; //default value
  IP_VERSION_IP_V4 = 1;
  IP_VERSION_IP_V6 = 2;
}

//=====
message IpAddress {
  oneof ipAddress{
    Ipv4Address ipv4Address = 1;
    Ipv6Address ipv6Address = 2;
  }
}

message Ipv4Address {
  string ipv4Address = 1;
}

message Ipv6Address {
  string ipv6Address = 1;
}

//=====
message IpAddressNoZone {
  oneof ipAddressNoZone{
    Ipv4AddressNoZone ipv4AddressNoZone = 1;
    Ipv6AddressNoZone ipv6AddressNoZone = 2;
  }
}

message Ipv4AddressNoZone {

```

```
    string ipv4AddressNoZone = 1;
  }

message Ipv6AddressNoZone {
  string ipv6AddressNoZone = 1;
}

//=====
message IpPrefix {
  oneof ipPrefix{
    Ipv4Prefix ipv4Prefix = 1;
    Ipv6Prefix ipv6Prefix = 2;
  }
}

message Ipv4Prefix {
  string Ipv4Prefix = 1;
}

message Ipv6Prefix {
  string Ipv6Prefix = 1;
}

//=====
message DomainName{
  string domainName = 1;
}

message Host {
  IpAddress ipAddress = 1;
  DomainName domainName = 2;
}
```


5.12 CustomOptions (onf-descriptor.proto)

To enable us to generate model metadata (that is not passed on the wire, but can be accessed from the language API, we will define a number of options.

This hand maintained file will need to be added to the output by the generator.

Note that the file needs to be in proto2 format due to a quirk of the language.

```
// (c) 2017 Open Networking Foundation. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// note that the compiler requires the option names to be unique in this file
// so options like description have the option type added to the name

syntax = "proto2";

package onf.protobuf;

import "google/protobuf/descriptor.proto";

enum UmlTypeEnum {
    ENTITY = 50000;
    DATATYPE = 50001;
    INTERFACE = 50002;
    NOTIFICATION = 50003;
    METHOD_REQUEST = 50004;
    METHOD_RESPONSE = 50005;
}

enum UmlLifecycleStateEnum {
    DEPRECATED = 50000;
    EXPERIMENTAL = 50001;
    FAULTY = 50002;
    LIKELY_TO_CHANGE = 50003;
    MATURE = 50004;
    OBSOLETE = 50005;
    PRELIMINARY = 50006;
}

enum UmlWriteAllowedEnum {
    WRITE_NOT_ALLOWED = 50000;
    UPDATE_ONLY = 50001;
    CREATE_ONLY = 50002;
    CREATE_AND_UPDATE = 50003;
}

enum UmlCounterTypeEnum {
    COUNTER_NA = 50000;
    COUNTER = 50001;
    GAUGE = 50002;
}
```

```

    ZERO_COUNTER = 50003;
}

enum UmlStringEncodingEnum {
    STRING_ENCODING_NA = 50000;
    BASE_64 = 50001;
    HEX = 50002;
    OCTET = 50003;
}

enum UmlBitLengthEnum {
    BIT_LENGTH_NA = 50000;
    LENGTH_8_BIT = 50001;
    LENGTH_16_BIT = 50002;
    LENGTH_32_BIT = 50003;
    LENGTH_64_BIT = 50004;
}

enum UmlSupportQualifierEnum {
    MANDATORY = 50000;
    OPTIONAL = 50001;
    CONDITIONAL_MANDATORY = 50002;
    CONDITIONAL_OPTIONAL = 50003;
    CONDITIONAL = 50004;
}

enum UmlNotificationDefinitionEnum {
    NOTIFICATION_NA = 50000;
    NO = 50001;
    YES = 50002;
}

// =====

extend google.protobuf.FileOptions {
    optional string uml_file_description = 50001;
}

extend google.protobuf.MessageOptions {
    optional string uml_message_description = 50001;
    optional UmlLifecycleStateEnum uml_message_lifecycle_state = 50002;
    optional bool uml_is_abstract = 50003;
    optional UmlTypeEnum uml_message_type = 50004;
    optional UmlSupportQualifierEnum uml_message_support = 50005;
    optional bool uml_is_leaf = 50006;
    optional UmlNotificationDefinitionEnum uml_object_creation_notification =
50007;
    optional UmlNotificationDefinitionEnum uml_object_deletion_notification =
50008;
    optional string uml_message_reference = 50009;
    optional string uml_message_condition = 50010;
    repeated string uml_notification_trigger_conditions = 50011;
    optional string uml_message_named_by = 50012;
    optional string uml_message_extends = 50013;
}

extend google.protobuf.FieldOptions {
    optional string uml_field_description = 50001;
    optional string uml_min_items = 50002;
    optional string uml_max_items = 50003;
    optional string uml_default_value = 50004;
    optional string uml_min_exclusive_value = 50005;
    optional string uml_max_exclusive_value = 50006;
}

```

```
    optional bool uml_is_ordered_collection = 50007;
    optional bool uml_is_unique_collection = 50008;
    optional bool uml_is_invariant = 50009;
    optional UmlCounterTypeEnum uml_counter_type = 50010;
    optional UmlWriteAllowedEnum uml_write_allowed = 50011;
    optional int32 uml_part_of_object_key = 50012;
    optional UmlSupportQualifierEnum uml_field_support = 50013;
    optional bool uml_is_unsigned = 50014;
    optional string uml_units = 50015;
    optional UmlNotificationDefinitionEnum uml_field_value_change_notification =
50016;
    optional UmlBitLengthEnum uml_bit_length = 50017;
    optional UmlStringEncodingEnum uml_string_encoding = 50018;
    optional string uml_field_reference = 50019;
    optional string uml_field_condition = 50020;
    optional UmlLifecycleStateEnum uml_field_lifecycle_state = 50021;
    optional string uml_references = 50022;
    optional bool uml_field_extends = 50023;
}

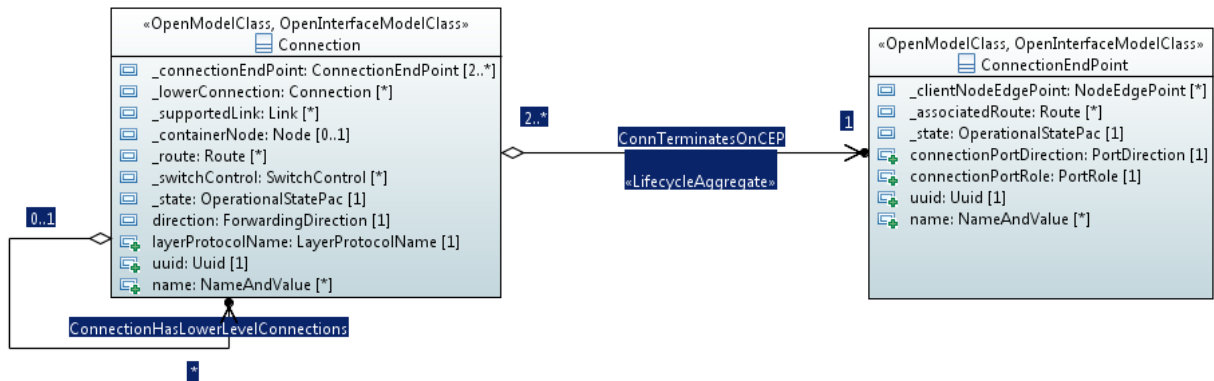
extend google.protobuf.EnumOptions {
    optional string uml_enum_description = 50001;
    optional UmlLifecycleStateEnum uml_enum_lifecycle_state = 50002;
    optional string uml_added_prefix = 50003;
}

extend google.protobuf.EnumValueOptions {
    optional string uml_literal_description = 50001;
}

extend google.protobuf.ServiceOptions {
    optional string uml_service_description = 50001;
    optional UmlSupportQualifierEnum uml_service_support = 50002;
    optional string uml_service_reference = 50003;
    optional UmlLifecycleStateEnum uml_service_lifecycle_state = 50020;
}

extend google.protobuf.MethodOptions {
    optional string uml_method_description = 50001;
    optional string uml_method_pre_condition = 50002;
    optional string uml_method_post_condition = 50003;
    optional string uml_method_is_idempotent = 50004;
    optional string uml_method_is_atomic = 50005;
    optional string uml_method_reference = 50006;
    optional UmlLifecycleStateEnum uml_method_lifecycle_state = 50007;
    optional UmlSupportQualifierEnum uml_method_support = 50008;
    optional string uml_method_exception = 50009;
    optional string uml_method_condition = 50010;
}
```

5.13 Hand Generated TAPI Example



«OpenModelClass, OpenInterfaceModelClass» Connection

Applied stereotypes:

- OpenModelClass (from OpenModel_Profile)
 - support: SupportQualifier [1] = MANDATORY
 - condition: String [0..1] = null
- OpenInterfaceModelClass (from OpenInterfaceModel_Profile)
 - objectCreationNotification: NotificationDefinition [1] = NA
 - objectDeletionNotification: NotificationDefinition [1] = NA

«OpenModelClass, OpenInterfaceModelClass» ConnectionEndPoint

Applied stereotypes:

- OpenModelClass (from OpenModel_Profile)
 - support: SupportQualifier [1] = MANDATORY
 - condition: String [0..1] = null
- OpenInterfaceModelClass (from OpenInterfaceModel_Profile)
 - objectCreationNotification: NotificationDefinition [1] = NA
 - objectDeletionNotification: NotificationDefinition [1] = NA

direction : ForwardingDirection

Applied stereotypes:

- OpenModelAttribute (from OpenModel_Profile)
 - partOfObjectKey: Integer [1] = 0
 - uniqueSet: Integer [*] = []
 - isInvariant: Boolean [1] = false
 - valueRange: String [0..1] = null
 - unsigned: Boolean [0..1] = false
 - counter: Counter [0..1] = NA
 - unit: String [0..1] = null
 - support: SupportQualifier [1] = MANDATORY
 - condition: String [0..1] = null
- OpenInterfaceModelAttribute (from OpenInterfaceModel_Profile)
 - writeAllowed: WriteAllowed [1] = CREATE_AND_UPDATE
 - attributeValueChangeNotification: NotificationDefinition [1] = NA
 - bitLength: BitLength [0..1] = NA
 - encoding: Encoding [0..1] = NA

5.13.1 TapiCommon.proto

```

// (c) 2017 Open Networking Foundation. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// TapiModule-Interfaces-CoreModelAPI generated from UML
// generator version 0.0.1
// TapiModule-Interfaces-CoreModelAPI
// model version 1.0.0
//
// Simple example to demonstrate what the generator output should look like

syntax = "proto3";

package onf.protobuf;

import "onf/protobuf/onf-descriptor.proto";
import "onf/protobuf/onf-types.proto";

// here we just list the types used in our example

enum PortRoleEnum {
  option (uml_enum_description) = "The role of an end in the context of the
function of the forwarding entity that it bounds";
  option (uml_added_prefix) = "PORT_ROLE_";

  PORT_ROLE_SYMMETRIC = 0;
  PORT_ROLE_ROOT = 1;
  PORT_ROLE_LEAF = 2;
  PORT_ROLE_TRUNK = 3;
  PORT_ROLE_UNKNOWN = 4;
}

enum PortDirectionEnum {
  option (uml_enum_description) = "The orientation of flow at the Port of a
Forwarding entity";
  option (uml_added_prefix) = "PORT_DIRECTION_";

  PORT_DIRECTION_BIDIRECTIONAL = 0 [(uml_literal_description) = "The Port has
both an INPUT flow and an OUTPUT flow defined."];
  PORT_DIRECTION_INPUT = 1 [(uml_literal_description) = "The Port only has
definition for a flow into the Forwarding entity (i.e. an ingress flow)."];
  PORT_DIRECTION_OUTPUT = 2 [(uml_literal_description) = "The Port only has
definition for a flow out of the Forwarding entity (i.e. an egress flow)."];
  PORT_DIRECTION_UNIDENTIFIED_OR_UNKNOWN = 3 [(uml_literal_description) = "Not a
normal state. The system is unable to determine the correct value."];
}

message NameAndValue {
  option (uml_message_description) = "A scoped name-value pair";
  option (uml_message_type) = DATATYPE;
}

```

```
    string valueName = 1 [
      (uml_field_description) = "The name of the value. The value need not have a
name.",
      (uml_min_items) = "0",
      (uml_max_items) = "1"
    ];
    string value = 2 [
      (uml_field_description) = "The value",
      (uml_min_items) = "1",
      (uml_max_items) = "1"
    ];
  }

message GlobalClass {
  option (uml_message_description) = "The TAPI GlobalComponent serves as the
super class for all TAPI entities that can be directly retrieved by their ID.
As such, these are first class entities and their ID is expected to be globally
unique. ";
  option (uml_is_abstract) = true;
  option (uml_message_type) = ENTITY;

  Uuid uuid = 1 [
    (uml_field_description) = "UUID: An identifier that is universally unique
within an identifier space, where the identifier space is itself globally
unique, and immutable. An UUID carries no semantics with respect to the purpose
or state of the entity.",
    (uml_part_of_object_key) = 1,
    (uml_is_invariant) = true,
    (uml_field_support) = MANDATORY,
    (uml_write_allowed) = CREATE_AND_UPDATE,
    (uml_min_items) = "1",
    (uml_max_items) = "1"
  ];

  repeated NameAndValue name = 2 [
    (uml_field_description) = "List of names. A property of an entity with a
value that is unique in some namespace but may change during the life of the
entity. A name carries no semantics with respect to the purpose of the
entity.",
    (uml_min_items) = "0",
    (uml_max_items) = "*"
  ];
}

message ResourceSpec {
}
```

5.13.2 TapiConnectivity.proto

```

// (c) 2017 Open Networking Foundation. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// TapiModule-Interfaces-CoreModelAPI generated from UML
// generator version 0.0.1
// TapiModule-Interfaces-CoreModelAPI model version1.0.0
//
// Simple example to demonstrate what the generator output should look like

syntax = "proto3";

package onf.protobuf;

import "onf/protobuf/onf-descriptor.proto";
import "onf/protobuf/TapiCommon.proto";

message Connection {
    option (uml_message_description) = "The ForwardingConstruct (FC) object class
models enabled potential for forwarding between two or more LTPs and like the
LTP supports any transport protocol including all circuit and packet forms.";
    option (uml_message_support) = MANDATORY;
    option (uml_message_type) = ENTITY;
    option (uml_is_abstract) = false;

    ConnectionEndPoint _connectionEndPoint = 1 [
        (uml_field_support) = MANDATORY,
        (uml_write_allowed) = CREATE_AND_UPDATE,
        (uml_min_items) = "2",
        (uml_max_items) = "*"
    ];
}

message ConnectionEndPoint {
    option (uml_message_description) = "The LogicalTerminationPoint (LTP) object
class encapsulates the termination and adaptation functions of one or more
transport layers. The structure of LTP supports all transport protocols
including circuit and packet forms.";
    option (uml_message_support) = MANDATORY;
    option (uml_message_type) = ENTITY;
    option (uml_is_abstract) = false;

    PortDirectionEnum connectionPortDirection = 1 [
        (uml_field_description) = "The orientation of defined flow at the EndPoint."
    ];

    PortRoleEnum connectionPortRole = 2 [
        (uml_field_description) = "The orientation of defined flow at the EndPoint."
    ];
}

```

5.14 ProtoBuf Notes

5.14.1 pros and cons to upgrade protobuf v3

<https://groups.google.com/forum/#!topic/protobuf/8sJ4GE1oaJI>

“You can use the latest version but still use a corpus files syntax="proto2" files.”

“If you don't use default values or extensions, converting a proto2 syntax file to proto3 is quite simple.”

5.14.2 Replacing 'extensions' in proto3

<https://groups.google.com/forum/#!topic/protobuf/49chFOx06nU>

5.14.3 Timestamp

<https://groups.google.com/forum/#!topic/protobuf/udcKGuMbCUI>

<https://groups.google.com/forum/#!topic/protobuf/fR-VMsCEtDg>

```
syntax = "proto3";

import "google/protobuf/any.proto";
import "google/protobuf/timestamp.proto";

package movie.pbuf;

// This message is a one-to-one mapping to the Movie Collection in our
// MongoDB
message Movie {
  google.protobuf.Timestamp start_time = 1;
  string movie_name = 2;
  // TODO action flag
  // TODO add more here
}

message CommandControlMsg {
  string message = 1;
  google.protobuf.Any command = 2;
}
```

5.14.4 Proto3 Storing UUID

<https://groups.google.com/forum/#!topic/protobuf/THxuTQKsz54>

5.14.5 Representing Polymorphism

<https://stackoverflow.com/questions/3018743/whats-the-right-way-to-do-polymorphism-with-protocol-buffers>

<https://groups.google.com/forum/#!msg/protobuf/wRDrA0lfowM/1KULvGzodrEJ>

“If Bar extends Foo, simply write: (modified for proto3 format)

```
message Foo {
  string aString = 1;
}

message Bar {
  Foo foo = 1;
  double aDouble = 2;
}
```

So if Foo evolves, only Foo message is modified.”

“In **proto3** the `extend` keyword has been replaced. From the docs: If you are already familiar with proto2 syntax, the `Any` type replaces extensions. But beware: `Any` is essentially a bytes blob. Most of the times it is better to use [Oneof](#):”

```
syntax = "proto3";

message Cat {
  string litterType = 1;
}

message Dog {
  string leashType = 1;
}

message Animal {
  string name = 1;

  oneof animalType {
    Cat cat = 2;
    Dog dog = 3;
  }
}
```

Note that the `repeated` keyword can't be used with `one of`, limiting its usefulness (you'd have to create a wrapper message).

<https://groups.google.com/forum/#!msg/protobuf/ojpYHqx2104/bfyAhqBxAQAJ>

“There are two composition approaches available, depending on what your needs are. Contain the common stuff:

```
message Common { // base type but can also be used
  string account = 1;
  string symbol = 2;
}

message MSG1 { // extends common, by wrapping
  Common common = 1;
  String type = 2;
}

message MSG2 { // extends common, by wrapping
  Common common = 1;
  int32 id = 2;
}
```

Or contain the variable stuff:

```
message MSG { // a MSG1 or MSG2, note need to include ALL subclasses
  string account = 1;
  string symbol = 2;

  oneof subclasses {
    MSG1 msg1 = 3;
    MSG2 msg2 = 4;
  }

  message MSG1 {
    String type = 1;
  }

  message MSG2 {
    int32 id = 1;
  }
}”
```

5.14.6 int vs sint vs uint

<https://stackoverflow.com/questions/765916/is-there-ever-a-good-time-to-use-int32-instead-of-sint32-in-google-protocol-buff>

<https://developers.google.com/protocol-buffers/docs/encoding>

5.14.7 solutions to resolve enum field naming restriction

<https://stackoverflow.com/questions/27030332/solutions-to-resolve-enum-field-naming-restriction-in-google-protobuf-due-to-c>

“The prevailing solution in existing code is option (2): give each literal name a prefix corresponding to its type. This also helps reduce collisions with macros. It's verbose, but unlike the other options suggested, it has no runtime overhead and it creates the least confusion for people reading the code.”

5.14.8 Can protobuf service method return primitive type?

<https://stackoverflow.com/questions/28876725/can-protobuf-service-method-return-primitive-type>

“No, you cannot use a primitive type as either the request or response. You must use a message type. This is important because a message type can be extended later, in case you decide you want to add a new parameter or return some additional data.”

Note that there are wrapper messages defined for all the primitive types in the protobuf file wrappers.proto.

5.14.9 Protobuf RPC Service method without parameters

<https://stackoverflow.com/questions/29687243/protobuf-rpc-service-method-without-parameters>

“You (always) have to specify an input type (and return type). If you don't want the method to take any parameters, define an empty message type, like:

```
message WhoAreYouParams {}
```

The reason this is required is so that if you later need to add an optional parameter, you can do so without breaking existing code.”

5.14.10 Why required and optional is removed in Protocol Buffers 3

<https://stackoverflow.com/questions/31801257/why-required-and-optional-is-removed-in-protocol-buffers-3>

5.14.11 protobuf message holding reference to another message

<https://stackoverflow.com/questions/17471765/protobuf-message-holding-reference-to-another-message-of-same-type>

“There is no concept of "reference" in protobuf.”

5.14.12 Protocol buffer: does changing field name break the message?

<https://stackoverflow.com/questions/45431685/protocol-buffer-does-changing-field-name-break-the-message>

“Changing a field name will not affected protobuf encoding or compatibility between applications that use proto definitions which differ only by field names.

The binary protobuf encoding is based on tag numbers, so that is what you need to preserve. ”

5.14.13 Schema evolution in Avro, Protocol Buffers and Thrift

<http://martin.kleppmann.com/2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html>

5.14.14 finally, a better protobuf ?

From the original ProtoBuf creator

<https://capnproto.org/language.html>

5.15 Installing ProtoBuf on Windows

From <https://github.com/google/protobuf/releases>

- 1) Download the latest language binding you want and extract it
- 2) Find on the page the latest compiler (search for “win32” or “linux-x86” for linux) and extract it
- 3) add the compiler bin directory to the system path environment variable
- 4) open a command prompt window and check that “protoc” runs the compiler

Note that directory `protoc-3.4.0-win32\include\google\protobuf` contains

- `any.proto`
- `api.proto`
- `descriptor.proto`
- `duration.proto`
- `empty.proto`
- `field_mask.proto`
- `source_context.proto`
- `struct.proto`
- `timestamp.proto`
- `type.proto`
- `wrappers.proto`

If any of these are used in the generated code, then the appropriate import statement must be added to the generated file, for example

```
import "google/protobuf/timestamp.proto";
```