Phenopacket-schema Documentation

Release 2.0

Global Alliance for Genomics and Health

Aug 25, 2021

Contents

1 Introduction to Phenopackets **3**
- 1.1 What is a Phenopacket? . . . 3
- 1.2 Requirement Levels . . . 4
- 1.3 Ontologies . . . 5
- 1.4 A short introduction to protobuf . . . 5
- 1.5 FHIR Implementation Guide . . . 8

2 Phenopacket Schema **11**
- 2.1 Version 1.0 . . . 11
- 2.2 Version 2.0 . . . 12

3 Top-Level Elements **15**
- 3.1 Phenopacket . . . 15
- 3.2 Family . . . 17
- 3.3 Cohort . . . 19

4 Phenopacket building blocks **21**
- 4.1 Age . . . 21
- 4.2 AgeRange . . . 22
- 4.3 Biosample . . . 22
- 4.4 ComplexValue . . . 26
- 4.5 TypedQuantity . . . 27
- 4.6 Disease . . . 27
- 4.7 DoseInterval . . . 29
- 4.8 DrugType . . . 31
- 4.9 Evidence . . . 31
- 4.10 ExternalReference . . . 32
- 4.11 File . . . 33
- 4.12 GeneDescriptor . . . 34
- 4.13 GenomicInterpretation . . . 36
- 4.14 GestationalAge . . . 38
- 4.15 Individual . . . 38
- 4.16 Interpretation . . . 41
- 4.17 KaryotypicSex . . . 45
- 4.18 Measurement . . . 45
- 4.19 MedicalAction . . . 48
- 4.20 MetaData . . . 49

4.21 OntologyClass . . . 50
4.22 Pedigree . . . 51
4.23 PhenotypicFeature . . . 53
4.24 Procedure . . . 55
4.25 Quantity . . . 56
4.26 RadiationTherapy . . . 58
4.27 ReferenceRange . . . 59
4.28 Resource . . . 59
4.29 Sex . . . 62
4.30 TherapeuticRegimen . . . 63
4.31 TimeElement . . . 64
4.32 TimeInterval . . . 65
4.33 Timestamp . . . 66
4.34 Treatment . . . 67
4.35 Update . . . 69
4.36 Value . . . 70
4.37 VariationDescriptor . . . 71
4.38 VariantInterpretation . . . 76
4.39 VitalStatus . . . 78

5 Recommended Ontologies **81**
5.1 Diseases . . . 81
5.2 Phenotypic features . . . 81
5.3 Anatomy . . . 82
5.4 Genes . . . 82
5.5 Units of Measurement . . . 82
5.6 Genotypes . . . 82
5.7 Assays . . . 83
5.8 Medications . . . 83
5.9 The National Cancer Institute's Thesaurus . . . 83
5.10 Experimental Factor Ontology . . . 84

6 Working with Phenopackets **85**
6.1 Example code . . . 85
6.2 Security disclaimer . . . 94

7 Examples **95**
7.1 A complete example: Rare Disease . . . 95
7.2 A complete example: Oncology . . . 98
7.3 A complete example: COVID-19 . . . 103
7.4 Rare Disease . . . 109
7.5 A complete example in Java: Oncology . . . 113

The goal of the phenopacket-schema is to define the phenotypic description of a patient/sample in the context of rare disease, common/complex disease, or cancer. The schema as well as source code in Java, C++, and Python is available from the phenopacket-schema GitHub repository.

Version 1 of phenopackets was approved by GA4GH in October, 2019. Based on initial experiences and feedback from multiple sources, and discussions in the GA4GH Clin/Pheno Workstream and Phenopackets Subgroups, version 1 has been extended to include better representation of the time course of disease, treatment, and cancer-related data. The current document refers to the version 2 of the Phenopackets schema. Version 2 is currently being finalized by the Global Alliance for Genomics and Health (GA4GH) Clinical & Phenotypic Data Capture workstream.

To see the documentation for version 1, which was approved by GA4GH in 2019, use this link.

CHAPTER 1

Introduction to Phenopackets

1.1 What is a Phenopacket?

The Phenopacket Schema represents an open standard for sharing disease and phenotype information to improve our ability to understand, diagnose, and treat both rare and common diseases. A Phenopacket links detailed phenotype descriptions with disease, patient, and genetic information, enabling clinicians, biologists, and disease and drug researchers to build more complete models of disease (see *Disease* for the distinction between disease and phenotypic feature). The standard is designed to encourage wide adoption and synergy between the people, organizations and systems that comprise the joint effort to address human disease and biological understanding.

While great strides have been made in exchange formats for sequence and variation data (e.g. Variant Call Format) and the GA4GH Variation Representation Specification), complementary standards for phenotypes and environment are urgently needed. For individuals with rare and undiagnosed diseases, broad adoption and appropriate utilization of these standards could improve the speed and accuracy of diagnosis by promoting quicker, more comprehensive, and accurate exchange of information relevant for research and medical care. The development of a clinical phenotype data exchange standard is both necessary and timely. It is necessary because study sizes of well over 100,000 patients are thought to be required to effectively assess the role of rare variation in common disease or to discover the genomic basis for a substantial portion of Mendelian diseases. It is timely because studies of this power are now becoming financially and technologically tractable.

Phenotypic abnormalities of individuals are currently described in diverse places in diverse formats: publications, databases, health records, and even in social media. We propose that these descriptions a) contain a minimum set of fields and b) get transmitted alongside genomic sequence data, such as in VCF, between clinics, authors, journals, and data repositories. The structure of the data in the exchange standard will be optimized for integration from these distributed contexts. The implementation of such a system will allow the sharing of phenotype data prospectively, as well as retrospectively. Increasing the volume of computable data across a diversity of systems will support large-scale computational disease analysis using the combined genotype and phenotype data.

The terms 'disease' and 'phenotype' are often conflated. The Phenopackets schema uses `phenotypic feature` to refer to a phenotypic feature, such as Arachnodactyly, that is the component of a disease, such as Marfan syndrome. The Phenopacket proposed here is designed to support deep phenotyping, a process wherein individual components of each phenotype are observed and documented. The phenotypes may be constitutional or those related to a sample (such as from a biopsy).

1.2 Requirement Levels

The schema is formally defined using protobuf3. In protobuf3, all elements are optional, and so there is no mechanism within protobuf to declare that a certain field is required. The Phenopacket schema does require some fields to be present and in some cases additionally requires that these fields have a certain format (syntax) or intended meaning (semantics). Software that uses Phenopackets should check the validity of the data with other means. We provide a Java implementation called Phenopacket Validator that tests Phenopackets (and related messages including Family, Cohort, and Biosample messages) for validity. Application code may additionally check for application-specific criteria.

1.2.1 Hierarchical requirements

The requirement levels that are shown for the various elements of the Phenopacket only apply if the element is used. For instance, the *Quantity* shows that the `unit` and `value` fields are required (the multiplicity is exactly 1 and the word REQUIRED is shown in the description). In contrast the field `reference_range` is optional (the multiplicity may be 0 or 1 and neither REQUIRED nor RECOMMENDED is used in the description). The requirements only apply if a *Quantity* is used in a Phenopacket. For instance, Phenopackets that do not contain *Measurement* or *Treatment* elements do not contain *Quantity* elements, and so the requirements for the fields of *Quantity* do not apply.

1.2.2 Multiplicity

The explanations for the various elements of the Phenopacket show the required multiplcities.

- `0..1`: The element may be absent (0) or present (1), i.e., the element is optional. Elements with multiplicity `0..1` may be marked RECOMMENDED, otherwise they are OPTIONAL.
- `1..1`: The element must be present (1), i.e., the element is REQUIRED
- `0..*`: There may be from zero to an arbitrary number of elements, i.e., a potentially empty list
- `1..*`: There may be from one to an arbitrary number of elements, i.e., a list that must not be empty

1.2.3 Levels

The Phenopacket schema uses three requirement levels. The required/recommended/optional designations are phenopacket-specific extensions used in the schema only (not code) and are not supported by protobuf.

Required

If a field is required, its presence is an absolute requirement of the specification, failing which the entire phenopacket is regarded as malformed. This corresponds to the key words `MUST`, `REQUIRED`, and `SHALL` in RFC2119.

Validation software must emit an error if a required field is missing. We note that natively protobuf messages never return a null pointer, and so if a field is missing it will be an empty string, a zero, or default instance depending on the datatype. Therefore, in practice, validation software does not need to check for null pointers.

Recommended

A field is not absolutely required, or there are valid reasons in particular circumstances that the field does not apply to the intended use case of the Phenopacket. This corresponds to the key words `SHOULD` and `RECOMMENDED` in RFC2119. For example, a variant may be associated with an id that can be useful to have but is not necessary for an analysis. The variant NM_000138.4:c.*2024A>G is associated with the id rs558488257.

Validation software may emit a warning if a recommended field is missing.

Optional

A field is truly optional. This category can be applied to fields that are only useful for a certain type of data. For instance, the *Biosample* field of the *Phenopacket* message is only used for Phenopackets that have an associated biosample(s).

The general-purpose validator must not emit a warning about these fields whether or not they are present. It may be appropriate for application-specific validators to emit a warning or even an error if a certain optional field is not present.

1.3 Ontologies

A terminology is a set of prefered or official terms in a domain. One of the most important terminologies for information retrieval in the medical domain is the Medical Subject Headings (MeSH), which is used for indexing and searching Medline.

Ontologies differ from terminologies in that ontologies define relationships between concepts in a way that allows computational logical reasoning. A short introduction is available in this recent review.

While both terminologies and ontologies can be used, the phenopacket schema recommends the use of specific ontologies in order to maximize utility of performing algorithmic analysis over medically relevant abnormalities.

The Human Phenotype Ontology (HPO) describes patient phenotypic features and was originally designed for genomic diagnostics, translational research, genomic matchmaking, and systems biology applications in the field of rare disease and other fields of medicine. It is increasinly used within clinical applications today for precision medicine.

The HPO is developed in the context of the Monarch Initiative, an international team of computer scientists, clinicians, and biologists in the United States, Europe, and Australia; HPO is being translated into multiple languages to support international interoperability. Due to its extensive phenotypic coverage beyond other terminologies, HPO has recently been integrated into the Unified Medical Language System (UMLS) to support deep phenotyping in a variety of mainstream health care IT systems.

The National Cancer institute's Thesaurus (NCIt) is used for cancer biosamples, and is the de facto standard for cancer knowledge representation and regulatory submission.

The Mondo Disease Ontology (Mondo) is an ontology that harmonizes disease definitions and identifiers in a systematic and computational manner across many resources, and is therefore the recommneded disease ontology for use in Phenopackets to maximize interoperability.

Other terminologies and ontologies may be used in the Phenopackets Schema, such as the International Classification of Diseases (ICD) or the Systematized Nomenclature of Medicine (SNOMED-CT), or numerous ontologies from the Open Biomedical Ontologies Library (OBO).

1.4 A short introduction to protobuf

Phenopackets schema uses protobuf, an exchange format developed in 2008 by Google. We refer readers to the excellent Wikipedia page on Protobuf and to Google's documentation for details. This page intends to get curious readers who are unfamiliar with protobuf up to speed with the main aspects of this technology, but it is not necessary to understand protobuf to use the phenopacket schema.

Google initially developed Protocol Buffers (protobuf) for internal use, but now has provided a code generator for multiple languages under an open source license. In this documentation, we will demonstrate use of phenopacket-schema with Java, but all of the features are available in any of the languages that protobuf works with including C++ and Python.

The major advantages of protobuf are that it is language-neutral, faster than many other schema languages such as XML and JSON, and can be simpler to use because of features such as automatic validation of data objects.

Protobuf forsees that data structures (so-called **messages**) are defined in a definition file (with the suffix .proto) and compiled to generate code that is invoked by the sender or recipient of the data to encode/decode the data.

1.4.1 Installing protobuf

The following exercise is not necessary to use phenopacket-schema, but is intended to build intuition for how protobuf works. We first need to install protobuf (Note that these instructions are for this tutorial only. The maven system will automatically pull in protobuf for phenopackets schema). We show one simple way of installing protobuf on a linux system in the following.

1. Download the source code from the protobuf GitHub page. Most users should download the latest tar.gz archive from the Release page. Extract the code.
2. Install the code as follows (to do so, you will need the packages autoconf, automake, libtool, curl, make, g++, unzip).

```
./configure
make
make check
sudo make install
sudo ldconfig # refresh shared library cache.
```

You now should check if installation was successful

```
$ protoc --version
libprotoc 3.8.0
```

1.4.2 An example schema

protobuf represents data as messages whose fields are indicated and aliased with a number and tag. Fields can be required, optional, or repeated. The following message describes a dog. The name is represented as a string, and the field is indicated with the number 1. The weight of the dog is represented as an integer. The toys field can store multiple items represented as a string. Note that in protobuf3, it is not possible to define a field as required.

```
syntax = "proto3";

message dog {
  required string name = 1;
  int32 weight = 2;
  repeated string toys = 4;
  }
```

We can compile this code with the following command

```
$ protoc -I=. --java_out=. dog.proto
```

This will generate a Java file called `Dog.java` with code to create, import, and export protobuf data. For example, the weight field is represented as follows.

```
public static final int WEIGHT_FIELD_NUMBER = 2;
private int weight_;
public int getWeight() {
```

(continues on next page)

(continued from previous page)

```
  return weight_;
}
```

It is highly recommended to peruse the complete Java file, but we will leave that as an exercise for the reader.

1.4.3 Using the generated code

We can now easily use a generated code to create Java instance of the Dog class. We will not provide a complete maven tutorial here, but the key things that need to be done to get this to work are the following.

1. set up a maven-typical directory structure such as:

```
src
--main
----java
------org
--------example
----proto
```

Add the following to the dependencies

```
<dependency>
  <groupId>com.google.protobuf</groupId>
  <artifactId>protobuf-java</artifactId>
  <version>3.5.1</version>
</dependency>
```

and add the following to the plugin section

```
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <version>0.5.1</version>
  <extensions>true</extensions>
  <configuration>
    <protocExecutable>/usr/local/bin/protoc</protocExecutable>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
        <goal>test-compile</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

This is the simplest configuration of the xolstice plugin; see the documentation for further information. We have assumed that protoc is installed in /usr/local/bin in the above, and the path may need to be adjusted on your system.

Add the protobuf definition to the proto directory. Add a class such as *Main.java* in the /src/main/java/org/example directory (package: org.example). For simplicity, the following code snippets could be written in the main method

```
String name = "Fido";
int weight = 5;
String toy1="bone";
```

(continues on next page)

(continued from previous page)

```
String toy2="ball";

Dog.dog fido = Dog.dog.newBuilder()
             .setName(name).
             setWeight(weight).
             addToys(toy1).
             addToys(toy2).
             build();

 System.out.println(fido.getName() + "; weight: " + fido.getWeight() + "kg;  favorite␣
↪toys: "
     + fido.getToysList().stream().collect(Collectors.joining("; ")));
```

The code can be compiled with

```
$ mvn clean package
```

If we run the demo app, it should output the following.

```
Fido; weight: 5kg;  favorite toys: bone; ball``.
```

Serialization

The following code snippet serializes the Java object fido and writes the serialized message to disk, then reads the message and displays it.

```
try {
    // serialize
    String filePath="fido.pb";
    FileOutputStream fos = new FileOutputStream(filePath);
    fido.writeTo(fos);
    // deserialize
    Dog.dog deserialized
            = Dog.dog.newBuilder()
            .mergeFrom(new FileInputStream(filePath)).build();

    System.out.println("deserialized: "+deserialized.getName() + "; weight: " +␣
↪deserialized.getWeight() + "kg;  favorite toys: "
            + deserialized.getToysList().stream().collect(Collectors.joining("; ")));

} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

The code should output the following.

```
deserialized: Fido; weight: 5kg;  favorite toys: bone; ball
```

We hope that this brief introduction was useful and refer to Google's documentation for more details.

1.5 FHIR Implementation Guide

Phenopacket on FHIR is a Fast Healthcare Interoperability Resources (FHIR) Implementation Guide (IG) based on the Phenopacket standard. It is meant to be conceptually equivalent but not directly interoperable. The guide is hosted

at:

http://phenopackets.org/core-ig/master/

A list of the FHIR artifacts defined as part of this implementation guide can be found at http://phenopackets.org/core-ig/master/artifacts.html

Briefly, the implementation guide includes definitions for the properties by which a RESTFUL server can be queried (Search Parameters), the constraints on FHIR resources for systems (Resource Profiles) and on FHIR data types (Extension Definitions) conforming to this implementation guide, and the existent sets of codes (Value Sets) and new Code Systems used by systems that conform to the Phenopacket FHIR IG.

As well, with the help of the community, we are driving efforts to capture an analysis of the domain modeled in the Phenopacket schema, additional use cases and modeling requirements, how to address interoperability with the FHIR standard, and other relevant content. This Domain Analysis Document (DAD) is constantly evolving, and all members of the Phenopacket standard community are welcome to contribute. The DAD can be found at https://phenopackets-analysis.readthedocs.io/en/latest/

CHAPTER 2

Phenopacket Schema

The goal of the phenopacket-schema is to define a machine-readable phenotypic description of a patient/sample in the context of rare disease, common/complex disease, or cancer. It aims to provide sufficient and shareable information of the data outside of the EHR (Electronic Health Record) with the aim of enabling capturing of sufficient structured data at the point of care by a clinician or clinical geneticist for sharing with other labs or computational analysis of the data in clinical or research environments.

This work has been produced as part of the GA4GH Clinical Phenotype Data Capture Workstream and is designed to be compatible with GA4GH metadata-schemas.

The phenopacket schema defines a common, limited set of data types which may be composed into more specialised types for data sharing between resources using an agreed upon common schema.

This common schema has been used to define the 'Phenopacket' which is a catch-all collection of data types, specifically focused on representing disease data both initial data capture and analysis. The phenopacket schema is designed to be both human and machine-readable, and to inter-operate with standards being developed in organizations such as in the ISO TC215 committee and the HL7 Fast Healthcare Interoperability Resources Specification (aka FHIR®).

The structure of the schema is defined in protobuf. You can find out more in the section *A short introduction to protobuf*.

2.1 Version 1.0

The diagram below shows an overview of the schema elements.

Fig. 1: Overview of v1.0 of the schema. Lines between elements indicate composition. Note that the OntologyClass and TimeElement links have been omitted for legibility. The colour scheme represents: base classes, interpretation classes, genomic classes, pedigree classes, top-level classes

Fig. 2: Detailed view of v1.0 of the schema. Lines between elements indicate composition. Note that the OntologyClass links have been omitted for legibility.

2.2 Version 2.0

Fig. 3: Overview of v2.0 of the schema. Lines between elements indicate composition. Note that the OntologyClass and TimeElement links have been omitted for legibility. The colour scheme represents: base classes, interpretation classes, measurement classes, genomic/vrs classes, pedigree classes, top-level classes, medical-action classes

Fig. 4: Detailed view of v2.0 of the schema. Lines between elements indicate composition. Note that the OntologyClass and TimeElement links have been omitted for legibility.

Version 2.0 includes significant changes and additions to the model to enable better representation of cancer and common disease, as well as catering for the original use-case for rare-disease.

2.2.1 Additions

The following elements and their sub-elements were added to the 2.0 schema. Other additional fields have been added throughout the schema.

Measurements

Added a new *Measurement* message for capturing quantitative, ordinal (e.g., absent/present), or categorical measurements. This element is available as a repeated field in the *Phenopacket* and *Biosample* top-level elements.

Medical actions

The *MedicalAction* was added to capture medications, procedures, other actions taken for clinical management. This element is available as a repeated field in the *Phenopacket*.

Time element

The *TimeElement* was added to collect the various ways of expressing time or age throughout the schema. In general where there was an *onset* or *start* time, a *resolution* or *end* *TimeElement* has been added.

VRS / VRSATILE

The *GeneDescriptor* and *VariationDescriptor* replace the v1.0 `Gene` and `Variant` messages. The new messages are based on the VRS and VRSATILE schemas defined by the GA4GH GKS group

2.2.2 Non-breaking Changes

The .proto files in the schema have been re-organised into more self-contained logical groups extracted from the *base.proto* file. These files are all organised into a *v2* package which lives alongside the *v1* package. For some language bindings it may be required to fix import paths for code created with the previous version to compile against the latest release, but otherwise code using v1.0 of the schema should work identically.

2.2.3 Breaking Changes

Time in Individual, Biosample, Disease, Phenotypic Feature

The *TimeElement* replaces the onset *oneof* in *PhenotypicFeature* and *Disease*, the *time_of_collection* field in *Biosample*. The *Individual age* field has been replaced with a *time_at_encounter TimeElement* and *Biosample individual_age_at_collection* has been replaced with a *time_of_collection TimeElement*. *PhenotypicFeature* 'negated' field was renamed to 'excluded' to be in line with *Disease* when indicating an absent phenotype.

Gene and Variant contexts

In *Phenopacket* and *Biosample* the *genes* and *variants* fields have been removed. In the case of the *Phenopacket* these have been replaced with the updated *Interpretation*.

Interpretation

The v2.0 *Interpretation* is now a sub-element of a *phenopacket*, rather than an enclosing element. The change allows for better semantics on the `Gene` (now replaced by *GeneDescriptor*) and `Variant` (now replaced by *VariationDescriptor*) types and their relationship to an *Individual* or *Biosample* in the context of a *Diagnosis* based on a *GenomicInterpretation*.

CHAPTER 3

Top-Level Elements

The phenopacket schema features top-level elements that make use of *Phenopacket building blocks* to structure the information.

3.1 Phenopacket

A Phenopacket is an anonymous phenotypic description of an individual or biosample with potential genes of interest and/or diagnoses. It can be used for multiple use cases. For instance, it can be used to describe the phenotypic findings observed in an individual with a disease that is being studied or for an individual in whom the diagnosis is being sought. The phenopacket can contain information about genetic findings that are causative of the disease, or alternatively it can contain a reference to a VCF file if exome sequencing is being performed as a part of the differential diagnostic process. A Phenopacket can also be used to describe the constitutional phenotypic findings of an individual with cancer (a *Biosample* should be used to describe the phenotypic abnormalities directly associated with an extirpated or biopsied tumor).

Table 1: Definition of the `Phenopacket` element

Field	Type	Multiplicity	Definition
id	string	1..1	arbitrary identifier. REQUIRED.
subject	*Individual*	0..1	The proband. RECOMMENDED.
phenotypic_features	List of *PhenotypicFeature*	0..*	Phenotypic features observed in the proband. RECOMMENDED.
measurements	List of *Measurement*	0..*	Measurements performed in the proband
biosamples	List of *Biosample*	0..*	samples (e.g., biopsies), if any
interpretations	List of *Interpretation*	0..*	Interpretations related to this phenopacket
diseases	List of *Disease*	0..*	Disease(s) diagnosed in the proband
medical_actions	List of *MedicalAction*	0..*	Medical actions performed
files	List of *File*	0..*	list of files related to the subject, e.g. VCF or other high-throughput sequencing files
meta_data	*MetaData*	1..1	Information about ontologies and references used in the phenopacket. REQUIRED.

3.1.1 Examples

TODO link to several longer examples.

3.1.2 Explanations

id

The id is an identifier specific for this phenopacket. The syntax of the identifier is application specific.

subject

This is typically the individual human (or another organism) that the Phenopacket is describing. In many cases, the individual will be a patient or proband of the study. See *Individual* for further information.

phenotypic_features

This is a list of phenotypic findings observed in the subject. See *PhenotypicFeature* for further information.

measurements

A list of measurements performed in the patient. In contrast to *PhenotypicFeature*, which relies on an *OntologyClass* to specify the observation, the *Measurement* can be used to report quanititative as well as ordinal or categorical measurements.

biosamples

This field describes samples that have been derived from the patient who is the object of the Phenopacket. or a collection of biosamples in isolation. See *Biosample* for further information.

interpretations

An optional list of *Interpretation* related to the phenopacket. These elements are intended to represent interpretations of disease or phenotypic findings based on genomic findings and must relate either to a genetic or genomic investigation of organismal origin (e.g., germline DNA derived from a blood sample) or from a *Biosample*.

diseases

This is a field for disease identifiers and can be used for listing either diagnosed or suspected conditions. The resources using these fields should define what this represents in their context. See *Disease* for further information.

medical_actions

A list of treatments or other medical actions performed for the person represented by this phenopacket. See *MedicalAction* for details.

files

This element contains a list of pointers to relevant file(s) for the *subject*. For example, the results of a high-throughput sequencing experiment. See *File* for further information.

meta_data

This element contains structured definitions of the resources and ontologies used within the phenopacket. It is expected that every valid Phenopacket contains a metaData element. See *MetaData* for further information.

3.2 Family

Phenotype, sample and pedigree data required for a genomic diagnosis. This element is equivalent to the Genomics England InterpretationRequestRD.

In many cases, genetic diagnostics of Mendelian and other disease is performed on a family basis in order to check for cosegregation of candidate variants with a disease. Usually, one family member is designated as the `proband`, for instance, a child affected by a genetic disease might be the proband in a family. Genomic diagnostics might involve genome sequencing of the child and his or her parents. In this case, the `Family` element would include a Phenopacket for the child (`proband` element). If the parents themselves display phenotypic findings relevant to the analysis, then Phenopackets are included for them (using the `relatives` element). If the parents do not display any relevant phenotypic findings, then it is not necessary to include Phenopacket elements for them. Instead, their status as unaffected can be recorded with the *Pedigree* element.

3.2.1 Data model

Table 2: Definition of the `Family` element

Field	Type	Multiplicity	Description
id	string	1..1	application-specific identifier. REQUIRED.
proband	*Phenopacket*	1..1	The proband (index patient) in the family. REQUIRED.
relatives	*Phenopacket* (list)	0..*	list of Phenopackets for family members other than the proband
pedigree	*Pedigree*	1..1	representation of the pedigree. REQUIRED.
files	*File* (list)	0..*	list of files related to the whole family, e.g. multi-sample high-throughput sequencing files
meta_data	*MetaData*	1..1	Metadata about ontologies and references used in this message. REQUIRED.

3.2.2 Explanations

id

An identifier specific for this family.

proband

The individual representing the focus of this packet - e.g. the proband in rare disease cases or cancer patient. See *Individual* for further information.

relatives

Individuals related in some way to the patient. For instance, the individuals may be genetically related or may be members of a cohort. If this field is used, then it is expected that a pedigree will be included for genetically related individuals for use cases such as genomic diagnostics. If a phenopacket is being used to describe one member of a cohort, then in general one phenopacket will be created for each of the individuals in the cohort. If this field is used, then it is expected that a pedigree will be included for genetically related individuals for use cases such as genomic diagnostics. If all that is required is to record affected/not-affected status in a family, it is possible to use the pedigree element only.

pedigree

The pedigree defining the relations between the proband and their relatives. This element contains information compatible with the information in a PED file. Pedigree.individual_id MUST map to the PhenoPacket.Individual.id. See *Pedigree* for further information.

files

This element contains a list of pointers to relevant file(s) for the family as a whole. The file(s) MUST refer to the entire family. Otherwise individual files MUST be contained within their appropriate scope. e.g. within a `Phenopacket` for germline samples of an individual or within the scope of the `Phenopacket.Biosample` in the case of data derived from that biosample.

In the case of multi-sample high-throughput sequencing files the sample identifiers in the hight-throughput sequencing file MUST each map to a `Pedigree.individual_id` referenced in the `pedigree` field, in order that linkage analysis can be performed on the sample.

See *File* for further information.

meta_data

This element contains structured definitions of the resources and ontologies used within the phenopacket. It is expected that every valid Phenopacket contains a metaData element. See *MetaData* for further information.

3.3 Cohort

This element describes a group of individuals related in some phenotypic or genotypic aspect. For instance, a cohort can consist of individuals all of whom have been diagnosed with a certain disease or have been found to have a certain phenotypic feature.

We recommend using the *Family* element to describe families being investigated for the presence of a Mendelian disease.

The GA4GH is in the process of creating a Computable Cohort Working Group to define how a cohort is generated. It is the intention that the results of this definition should be collected into a *Cohort* message.

3.3.1 Data model

Table 3: Definition of the `Cohort` element

Field	Type	Multiplicity	Description
id	string	1..1	An arbitrary identifier for the cohort. REQUIRED
description	string	0..1	text description of the cohort
members	*Phenopacket*	1..*	Phenopackets that represent members of the cohort. REQUIRED
files	*File*	0..*	list of files related to the whole cohort, e.g. multi-sample high-throughput sequencing files
meta_data	*MetaData*	1..1	Metadata related to the ontologies and references used in this message. REQUIRED

3.3.2 Explanations

id

The id is an identifier specific for this cohort. The syntax of the identifier is application specific.

description

Any information relevant to the study can be added here as free text.

members

One *Phenopacket* is included for each member of the cohort.

files

This element contains a list of pointers to relevant file(s) for the cohort. The file(s) MUST refer to the entire cohort. Otherwise individual files MUST be contained within their appropriate scope. e.g. within a `Phenopacket` for germline samples of an individual or within the scope of the `Phenopacket.Biosample` in the case of data derived from that biosample.

See *File* for further information.

meta_data

This element contains structured definitions of the resources and ontologies used within the phenopacket. It is expected that every valid Phenopacket contains a metaData element. See *MetaData* for further information.

CHAPTER 4

Phenopacket building blocks

The phenopacket standard consists of several protobuf messages each of which contains information about a certain topic such as phenotype, variant, pedigree, and so on. One message can contain other messages, which allows a rich representation of data. For instance, the Phenopacket message contains messages of type Individual, PhenotypicFeature, Biosample, and so on. Individual messages can therefore be regarded as building blocks that are combined to create larger structures. It would also be straightforward to include the Phenopackets schema into larger schema for particular use cases. Follow the links to read more information about individual building blocks.

4.1 Age

The Age element allows the age of the subject to be encoded in several different ways that support different use cases. Age is encoded as ISO8601 duration.

4.1.1 Data model

Field	Type	Multiplicity	Description
iso8601duration	string	1..1	An ISO8601 string represent age

If the Age message is used, the `iso8601duration` value must be present.

4.1.2 Example

```
age:
  iso8601duration: "P25Y3M2D"
```

The string element (string age=1) should be used for ISO8601 durations (e.g., P40Y10M05D). For many use cases, less precise strings will be preferred for privacy reasons, e.g., P40Y.

age

It is possible to represent age using a string that should be formatted according to ISO8601 Duration.

4.2 AgeRange

The AgeRange element is intended to be used when the age of a subject is represented by a bin, e.g., 5-10 years. Bins such as this are used in some situations in order to protect the privacy of study participants, whose age is then represented by bins such as 45-49 years, 50-54 years, 55-59 years, and so on.

4.2.1 Data model

Field	Type	Multiplicity	Description
start	*Age*	1..1	An Age message
end	*Age*	1..1	An Age message

4.2.2 Example

For instance, to represent the bin 45-49 years, one could use an Age element with **P45Y** as the start element of the AgeRange element, and an Age element with **P49Y** as the end element. An AgeRange.end SHALL occur after AgeRange.start.

```
ageRange:
  start:
      iso8601duration: "P45Y"
  end:
      iso8601duration: "P49Y"
```

4.3 Biosample

A Biosample refers to a unit of biological material from which the substrate molecules (e.g. genomic DNA, RNA, proteins) for molecular analyses (e.g. sequencing, array hybridisation, mass-spectrometry) are extracted. Examples would be a tissue biopsy, a single cell from a culture for single cell genome sequencing or a protein fraction from a gradient centrifugation. Several instances (e.g. technical replicates) or types of experiments (e.g. genomic array as well as RNA-seq experiments) may refer to the same Biosample.

4.3.1 Data model

Field	Type	Multiplicity	Description
id	string	1..1	Arbitrary identifier. REQUIRED.
individual_id	string	0..1	Arbitrary identifier. RECOMMENDED.
derived_from_id	string	0..1	id of the biosample from which the current biosample was derived (if applicable)
description	string	0..1	arbitrary text
sampled_tissue	*OntologyClass*	0..1	Tissue from which the sample was taken
sample_type	*OntologyClass*	0..1	type of material, e.g., RNA, DNA, Cultured cells
phenotypic_features	*PhenotypicFeature* (List)	0..*	List of phenotypic abnormalities of the sample. RECOMMENDED.
measurements	*Measurement* (List)	0..*	List of measurements of the sample
taxonomy	*OntologyClass*	0..1	Species of the sampled individual
time_of_collection	*TimeElement*	0..1	Age of the proband at the time the sample was taken. RECOMMENDED.
histological_diagnosis	*OntologyClass*	0..1	Disease diagnosis that was inferred from the histological examination. RECOMMENDED.
tumor_progression	*OntologyClass*	0..1	Indicates primary, metastatic, recurrent. RECOMMENDED.
tumor_grade	*OntologyClass*	0..1	Term representing the tumor grade
pathological_stage	*OntologyClass*	0..1	Pathological stage, if applicable. RECOMMENDED.
pathological_tnm_finding	*OntologyClass* (List)	0..*	Pathological TNM findings, if applicable. RECOMMENDED.
diagnostic_markers	*OntologyClass* (List)	0..*	Clinically relevant biomarkers. RECOMMENDED.
procedure	*Procedure*	0..1	The procedure used to extract the biosample. RECOMMENDED.
files	*File* (List)	0..*	list of files related to the biosample, e.g. VCF or other high-throughput sequencing files
material_sample	*OntologyClass*	0..1	Status of specimen (tumor tissue, normal control, etc.). RECOMMENDED.
sample_processing	*OntologyClass*	0..1	how the specimen was processed
sample_storage	*OntologyClass*	0..1	how the specimen was stored

4.3.2 Example

The staging system most often used for bladder cancer is the American Joint Committee on Cancer (AJCC) TNM system. The overall stage is assigned based on the T, N, and M categories (Cancer stage grouping). For instance, stage II (pathological staging) is defined as T2a or T2b, N0, and M0, meaning the cancer has spread into the wall of the bladder.

```
biosample:
  id: "sample1"
  individualId: "patient1"
  description: "Additional information can go here"
  sampledTissue:
      id: "UBERON_0001256"
```

(continues on next page)

(continued from previous page)

```
    label: "wall of urinary bladder"
histologicalDiagnosis:
    id: "NCIT:C39853"
    label: "Infiltrating Urothelial Carcinoma"
tumorProgression:
    id: "NCIT:C84509"
    label: "Primary Malignant Neoplasm"
tumorGrade:
    id: "NCIT:C36136"
    label: "Grade 2 Lesion"
procedure:
    code:
        id: "NCIT:C5189"
        label: "Radical Cystoprostatectomy"
files:
    - uri: "file:///data/genomes/urothelial_ca_wgs.vcf.gz"
    individualToFileIdentifiers:
        patient1: "NA12345"
    fileAttributes:
        description: "Urothelial carcinoma sample"
        htsFormat: "VCF"
        genomeAssembly: "GRCh38"
materialSample:
    id: "EFO:0009655"
    label: "abnormal sample"
timeOfCollection:
    age:
        iso8601duration: "P52Y2M"
pathologicalStage:
    id: "NCIT:C28054"
    label: "Stage II"
pathologicalTnmFinding:
- id: "NCIT:C48726"
    label: "T2b Stage Finding"
- id: "NCIT:C48705"
    label: "N0 Stage Finding"
- id: "NCIT:C48699"
    label: "M0 Stage Finding"
```

4.3.3 Explanations

id

The Biosample id. This is unique in the context of the server instance.

individual_id

The id of the *Individual* this biosample was derived from. It is recommended, but not necessary to provide this information here if the Biosample is being transmitted as a part of a *Phenopacket*.

derived_from_id

The id of the parent biosample this biosample was derived from.

description

The biosample's description. This attribute contains human readable text. The "description" attributes should not contain any structured data.

sampled_tissue

On *OntologyClass* describing the tissue from which the specimen was collected. We recommend the use of UBERON. The PDX MI mapping is `Specimen tumor tissue`.

sample_type

RNA, DNA, Cultured cells. We recommend use of EFO term to describe the sample, for instance, genomic DNA (EFO:0008479).

phenotypic_features

The phenotypic characteristics of the BioSample, for example histological findings of a biopsy. See *PhenotypicFeature* for further information.

measurements

Measurements (usually quantitative) performed on the sample. See *Measurement* for further information.

taxonomy

For resources where there may be more than one organism being studied it is advisable to indicate the taxonomic identifier of that organism, to its most specific level. We advise using the codes from the NCBI Taxonomy resource. For instance, NCBITaxon:9606 is human (homo sapiens sapiens) and or NCBITaxon:9615 is dog.

individual_age_at_collection

An age object describing the age of the individual this biosample was derived from at the time of collection. The Age object allows the encoding of the age either as ISO8601 duration or time interval (preferred), or as ontology term object. See *TimeElement* for further information.

histological_diagnosis

This is the pathologist's diagnosis and may often represent a refinement of the clinical diagnosis (which could be reported in the *Phenopacket* that contains this Biosample). Normal samples would be tagged with the term "NCIT:C38757", "Negative Finding". See *OntologyClass* for further information.

tumor_progression

This field can be used to indicate if a specimen is from the primary tumor, a metastasis or a recurrence. There are multiple ways of representing this using ontology terms, and the terms chosen should have a specific meaning that is application specific.

For example a term from the following NCIT terms from the Neoplasm by Special Category can be chosen.

- Primary Neoplasm
- Metastatic Neoplasm
- Recurrent Neoplasm

tumor_grade

This should be a child term of NCIT:C28076 (Disease Grade Qualifier) or equivalent. See the tumor grade fact sheet.

diagnostic_markers

Clinically relevant bio markers. Most of the assays such as immunohistochemistry (IHC) are covered by the NCIT under the sub-hierarchy NCIT:C25294 (Laboratory Procedure), e.g. NCIT:C68748 (HER2/Neu Positive), NCIT:C131711 (Human Papillomavirus-18 Positive).

procedure

The clinical procedure performed on the subject in order to extract the biosample. See *Procedure* for further information.

files

This element contains a list of pointers to relevant file(s) for the biosample. For example, the results of a high-throughput sequencing experiment. See *File* for further information.

material_sample

This element can be used to specify the status of the sample. For instance, a status may be used as a normal control, often in combination with another sample that is thought to contain a pathological finding. We recommend use of ontology terms such as

- reference sample (EFO:0009654).
- abnormal sample (EFO:0009655).

sample_processing

The technique used to process the sample.

sample_storage

How the sample was stored.

4.4 ComplexValue

This element is intended for complex measurements, such as blood pressure where more than one component quantity is required to describe the measurement.

4.4.1 Data model

Field	Type	Multiplicity	Description
typed_quantities	*TypedQuantity*	1..*	list of quantities required to fully describe the complex value. REQUIRED.

4.5 TypedQuantity

The Complex Value element consists of a list of `TypedQuantity` elements.

4.5.1 Data model

Field	Type	Multiplicity	Description
type	*OntologyClass*	1..1	OntologyClass to describe the type of the measurement. REQUIRED.
quantity	*Quantity*	1..1	Quantity denoting the outcome of the measurement. REQUIRED.

4.5.2 Example

The following example shows a ComplexQuantity message for diastolic blood pressure. The intended use case for a ComplexQuantity message is as a component of a *Measurement* message that contains two or more components (e.g., systolic and diastolic blood pressure).

```
complexValue:
    typedQuantities:
    - type:
        id: "NCIT:C25298"
        label: "Systolic Blood Pressure"
      quantity:
        unit:
            id: "NCIT:C49670"
            label: "Millimeter of Mercury"
      value: 120.0
    - type:
        id: "NCIT:C25299"
        label: "Diastolic Blood Pressure"
      quantity:
        unit:
            id: "NCIT:C49670"
            label: "Millimeter of Mercury"
        value: 70.0
```

4.6 Disease

The word *phenotype* is used with many different meanings, including "the observable traits of an organism". In medicine, the word can be used with at least two different meanings. It is used to refer to some **observed** deviation from normal morphology, physiology, or behavior. In contrast, the *disease* is a diagnosis, i.e., an inference or hypothesis

about the cause underlying the observed phenotypic abnormalities. Occasionally, physicians use the word phenotype to refer to a disease, but we do not use this meaning here.

4.6.1 Data model

Field	Type	Multiplicity	Description
term	*OntologyClass*	1..1	An ontology class that represents the disease. REQUIRED.
excluded	boolean	0..1	Flag to indicate whether the disease was observed or not.
onset	*TimeElement*	0..1	an element representing the age of onset of the disease
resolution	*TimeElement*	0..1	an element representing the age of resolution (abatement) of the disease
disease_stage	*OntologyClass* (List)	0..*	List of terms representing the disease stage e.g. AJCC stage group.
clinical_tnm_finding	*OntologyClass* (List)	0..*	List of terms representing the tumor TNM score
primary_site	*OntologyClass*	0..1	the primary site of disease
laterality	*OntologyClass*	0..1	laterality (left or right) of the primary site of sites if applicable

4.6.2 Example

```
disease:
  term:
      id: "OMIM:164400"
      label: "Spinocerebellar ataxia 1"
  onset:
      age:
          iso8601duration: "P38Y7M"
```

See *A complete example: Oncology* for a usage of the Disease element that includes information about tumor staging.

4.6.3 Explanations

term

In the phenopacket schema, the disease element denotes the diagnosis by means of an ontology class. For rare diseases, we recommend using a term from Online Mendelian Inheritance in Man (OMIM) (e.g., OMIM:101600), Orphanet (e.g., Orphanet:710), or MONDO (e.g., MONDO:0007043). There are many other ontologies and terminologies that can be used including Disease Ontology, SNOMED, and ICD. For cancers, we recommend using terms from domain-specific ontologies, such as NCIthesaurus (e.g., NCIT:C9049).

excluded

Flag to indicate whether the disease was observed or not. Default is 'false', in other words the disease was observed. Therefore it is only required in cases to indicate that the disease was looked for, but found to be absent. More formally, this modifier indicates the logical negation of the OntologyClass used in the 'term' field. *CAUTION* It is imperative to check this field for correct interpretation of the disease!

onset

The `onset` element provides three possibilities of describing the onset of the disease. It is also possible to denote the onset of individual phenotypic features of disease in the Phenopacket element. If an ontology class is used to refer to the age of onset of the disease, we recommend using a term from the HPO onset hierarchy.

resolution

An element representing the age of resolution (abatement, recovery from) of the disease.

disease_stage

This attribute is used to describe the stage of disease. If the disease is a cancer, this attribute describes the extent of cancer development, typically including an AJCC stage group (i.e., Stage 0, I-IV), though other staging systems are used for some cancers. See staging. The list of elements constituting this attribute should be derived from child terms of NCIT:C28108 (Disease Stage Qualifier) or equivalent hierarchy from another ontology.

clinical_tnm_finding

This attribute can be used if the phenopacket is describing cancer. TNM findings score the progression of cancer with respect to the originating tumor (T), spread to lymph nodes (N), and presence of metastases (M). These findings are commonly reported for tumors, and support the stage classifications stored in the *disease_stage* attribute. See staging. The list of elements constituting this attribute should be derived from child terms of NCIT:C48232 (Cancer TNM Finding) or equivalent hierarchy from another ontology.

primary_site

The term used to describe the primary site of disease. Terms from the NCIT or UBERON.

laterality

The term used to indicate laterality of diagnosis, if applicable.

4.7 DoseInterval

This element represents a block of time in which the dosage of a medication was constant. For example, to represent a period of 30 mg twice a day for an interval of 10 days, we would use a *Quantity* element to represent the individual *30 mg* dose, and *OntologyClass* element to represent *twice a day*, and an rstinterval element to represent the 10-day interval.

4.7.1 Data model

Table 1: Definition of the `DoseInterval` element

Field	Type	Multiplicity	Description
quantity	*Quantity*	1..1	Amount administered in one dose. REQUIRED.
schedule_frequency	*OntologyClass*	1..1	how often doses are administered per day (or other indicated duration). REQUIRED.
interval	*TimeInterval*	1..1	The specific interval over which the dosage was administered in the indicated quantity. REQUIRED.

4.7.2 Example

The following message represents a dose interval from March 15, 2020 to March 25, 2020, in which a constant dose of 30 mg was given twice a day.

```
doseInterval:
  quantity:
      unit:
          id: "UO:0000022"
          label: "milligram"
      value: 30.0
  scheduleFrequency:
      id: "NCIT:C64496"
      label: "Twice Daily"
  interval:
      start: "2020-03-15T13:00:00Z"
      end: "2020-03-25T09:00:00Z"
```

4.7.3 Explanations

quantity

The amount of an individual dose (See *Quantity*).

schedule_frequency

This element specifies the number of instances within a specific time period. It is intended to have the same meaning as the NCIT Schedule Frequency class.

interval

The time interval over which the specified dosage is given. See *TimeInterval* for information about privacy concerns.

4.8 DrugType

Drugs can be administered in different contexts. This element does not intend to capture information about the administration route (e.g., by mouth or intravenous), but rather about the setting - inpatient, outpatient, or related to a (generally one-time) procedure.

DrugType is an enumeration.

Table 2: Values of the `DrugType` element

Item	Value
UNKNOWN_DRUG_TYPE	0
PRESCRIPTION	1
EHR_MEDICATION_LIST	2
ADMINISTRATION_RELATED_TO_PROCEDURE	3

This element does not intend to capture the medical reason for administering a treatment. Instead, it records the context in which the medication is administered. It is assumed that medications recorded on the medication list of an electronic health record (EHR) are likely to have been actually administered as prescribed. If a prescription is given for outpatient use, it is less likely that the medication will be taken exactly as prescribed (Osterberg L, Adherence to medication. N Engl J Med. 2005). Finally, medications may be given to conduct a medical procedure such as a local anesthetic given before a skin biopsy or a sedative given to perform a bronchoscopy.

4.9 Evidence

This element intends to represent the evidence for an assertion such as an observation of a *PhenotypicFeature*. We recommend the use of terms from the Evidence & Conclusion Ontology (ECO)

4.9.1 Data model

Table 3: Definition the `Evidence` element

Field	Type	Multiplicity	Description
evidence_code	*OntologyClass*	1..1	An ontology class that represents the evidence type. REQUIRED.
reference	*ExternalReference*	0..1	Representation of the source of the evidence

4.9.2 Example

```
evidence:
    evidenceCode:
        id: "ECO:0006017"
        label: "author statement from published clinical study used in manual␣
↪assertion"
    reference:
        id: "PMID:30962759"
        description: "Recurrent Erythema Nodosum in a Child with a SHOC2 Gene Mutation
↪"
```

4.9.3 Explanations

evidence_code

For example, in order to describe the evidence for a phenotypic observation that is derived from a publication, one might use the ECO term *author statement from published clinical study used in manual assertion* (ECO:0006017) and record a PubMed id in the reference field (See *ExternalReference*).

reference

An *ExternalReference* is used to store a reference to the publication or other source that supports the evidence. Not all types of evidence will have an external reference, and therefore this field is optional.

4.10 ExternalReference

This element encodes information about an external reference. One typical use case for this elements is to provide a reference to a published article by showing its PubMed identifier as a part of an *Evidence* element.

4.10.1 Data model

Table 4: Definition of the `ExternalReference` element

Field	Type	Multiplicity	Description
id	string	0..1	An application specific identifier. RECOMMENDED.
reference	string	0..1	An application specific identifier. RECOMMENDED.
description	string	0..1	An application specific description

4.10.2 Example

```
externalReference:
    id: "PMID:30962759"
    description: "Recurrent Erythema Nodosum in a Child with a SHOC2 Gene Mutation"

externalReference:
    id: "PMID:30962759"
    reference: "https://pubmed.ncbi.nlm.nih.gov/30962759"
    description: "Recurrent Erythema Nodosum in a Child with a SHOC2 Gene Mutation"
```

4.10.3 Explanations

id

The syntax of the identifier is application specific. It is RECOMMENDED that this is a *CURIE* that uniquely identifies the evidence source, e.g. **ISBN:978-3-16-148410-0** or **PMID:123456**. However, it could be a URL/URI, or other relevant identifier.

It is RECOMMENDED to use a *CURIE* identifier. If one is used, it is RECOMMENDED that the corresponding *Resource* be provided in the *MetaData* element. For the above example, one would provide an *Resource* for PubMed (see the *MetaData* for this example).

reference

It is RECOMMENDED that a full or partial URL/URI is provided for systems to resolve an external reference, especially in the absence of a CURIE identifier.

description

An optional free text description of the evidence. In the example above, the title of a published article is shown.

4.11 File

The File message allows a Phenopacket to link the structured phenotypic data it contains to external files which can be used to inform analyses. For example the file could link to sequencing data in VCF format as well as other data types.

Given that `File` elements are listed in various locations such as the `Phenopacket`, `Biosample`, `Family` etc. which can in turn be nested, individual files MUST be contained within their appropriate scope. For example within a `Phenopacket` for germline samples of an individual or within the scope of the `Phenopacket.Biosample` in the case of genomic data derived from sequencing or other characterisation of that biosample. Aggregate data types such as `Cohort` and `Family` MUST contain aggregate file data i.e. merged/multi-sample VCF at the level of the Family/Cohort, but each member Phenopacket can contain its own locally-scope files pertaining to that individual/biosample(s).

4.11.1 File

This message is used for information about a file.

Field	Type	Multiplicity	Description
uri	string	1..1	A valid URI e.g. file://data/file1.vcf.gz or https://opensnp.org/data/60.23andme-exome-vcf.231?1341012444. REQUIRED.
individual_to_file_identifiers	a map of string key: value	0..1	The mapping between the Individual.id or Biosample.id to any identifier in the file. RECOMMENDED.
file_attributes	a map of string key: value	0..1	A map of attributes pertaining to the file or its contents.

Example

The message below shows how a compressed VCF file should be specified. It indicates the sample identifier *NA12345* in the VCF file is equivalent to the Phenopacket individual identifier *patient23456*. The *fileAttributes* indicate that the file was called against the GRCh38 genome assembly, indicates the *fileFormat* is VCF and also provides a human-readable *description*.

```
file:
  uri: "file://data/genomes/germline_wgs.vcf.gz"
  individualToFileIdentifiers:
    patient23456: "NA12345"
```

(continues on next page)

(continued from previous page)

```
  fileAttributes:
    genomeAssembly: "GRCh38"
    fileFormat: "vcf"
    description: "Matched normal germline sample"
```

4.11.2 uri

URI for the file e.g. *file://data/genomes/file1.vcf.gz* or *https://opensnp.org/data/60.23andme-exome-vcf.231?1341012444*.

4.11.3 individual_to_file_identifiers

A map of identifiers mapping an individual referred to in the Phenopacket to an identifier used in the file. The key values must correspond to the Individual::id for the individuals in the message or Biosample::id for biosamples, the values must map to identifiers in the file.

4.11.4 file_attributes

A map of attributes which a resource might want to share about the contents of a file. For example a file containing genomic coordinates (e.g. VCF, BED) SHOULD contain an entry with the key *genomicAssembly* and a value indicating the genome assembly the contents of the file was called against. We recommend using the Genome Reference Consortium nomenclature e.g. *GRCh37*, *GRCh38*.

4.12 GeneDescriptor

This element represents an identifier for a gene, using the Gene Descriptor from the VRSATILE Framework. Gene Descriptors can be used to transmit the information that the gene is thought to play a causative role in the disease phenotypes being described in cases where the exact variant cannot be transmitted, either for privacy reasons or because it is unknown.

Gene Descriptors may also be used to contextualize variants described in a *VariationDescriptor*.

4.12.1 Data model

Table 5: Definition of the `GeneDescriptor` element

Field	Type	Multiplicity	Description
value_id	string	1..1	Official identifier of the gene. REQUIRED.
symbol	string	1..1	Official gene symbol. REQUIRED.
description	string	0..1	A free-text description of the gene
alternate_ids	list of string	0..*	Alternative identifier(s) of the gene
xrefs	list of string	0..*	Related concept IDs (e.g. gene ortholog IDs) may be placed in xrefs
alternate_symbols	list of string	0..*	Alternative symbol(s) of the gene

4.12.2 Example

```
geneDescriptor:
  valueId: "HGNC:3477"
  symbol: "ETF1"
```

Optionally, with alternative identifiers:

```
geneDescriptor:
  valueId: "HGNC:3477"
  symbol: "ETF1"
  alternateIds:
  - "ensembl:ENSG00000120705"
  - "ncbigene:2107"
  - "ucsc:uc003ldc.6"
  - "OMIM:600285"
```

Using the gene descriptor to convey alternate identifiers, symbols and orthologs:

```
geneDescriptor:
  valueId: "HGNC:3477"
  symbol: "ETF1"
  alternateIds:
  - "ensembl:ENSG00000120705"
  - "ncbigene:2107"
  - "ucsc:uc003ldc.6"
  - "OMIM:600285"
  alternateSymbols:
  - "SUP45L1"
  - "ERF1"
  - "ERF"
  - "eRF1"
  - "TB3-1"
  - "RF1"
  xrefs:
  - "VGNC:97422"
  - "MGI:2385071"
  - "RGD:1305712"
  - "ensembl:ENSRNOG00000019450"
  - "ncbigene:307503"
```

4.12.3 Explanations

value_id

The id represents the accession number of comparable identifier for the gene.

It SHOULD be a *CURIE* identifier with a prefix that is used by the official organism gene nomenclature committee. In the case of Humans, this is the HGNC, e.g. HGNC:347

symbol

This SHOULD use official gene symbol as designated by the organism gene nomenclature committee. In the case of human this is the HUGO Gene Nomenclature Committee e.g. ETF1.

description

A free-text description of the value object. This should be only used to convey information which is otherwise not possible to encode using the schema.

alternate_ids

This field can be used to provide identifiers to alternative resources where this gene is used or catalogued. For example, the NCBI and Ensemble both provide alternative identifiers for genes where they catalogue the transcripts for a gene e.g. ncbigene:2107, ensembl:ENSG00000120705 These identifiers SHOULD be represented in *CURIE* form with a corresponding *Resource*.

alternate_symbols

This field can be used to list the alternate symbols used to refer to the gene. These include the previously approved gene symbols and those used in the literature or other databases to refer to the gene.

xrefs

This field can be used to provide identifiers to alternative resources representing related, but not equivalent concepts, for example gene ortholog ids

4.13 GenomicInterpretation

This element is used as a component of the *Interpretation* element, and describes the interpretation for an individual variant or gene. Note that multiple variants or genes may support the interpretation related to one disease. See the *Interpretation* element for examples.

4.13.1 Data model

GenomicInterpretation

Table 6: Definition of the `GenomicInterpretation` element

Field	Type	Multiplicity	Description
subject_or_biosample_id	string	1..1	The id of the patient or biosample that is the subject being interpreted. REQUIRED.
interpretation_status	enum *InterpretationStatus*	1..1	status of the interpretation. REQUIRED.
call	oneof {*GeneDescriptor* | *VariantInterpretation*}	1..1	represents the interpretation. REQUIRED.

InterpretationStatus

Table 7: Definition of the `InterpretationStatus` enumeration

Name	Ordinal	Description
UNKNOWN_STATUS	0	No information is available about the status
REJECTED	1	The variant or gene reported here is interpreted *not* to be related to the diagnosis
CANDIDATE	2	The variant or gene reported here is interpreted to *possibly* be related to the diagnosis
CONTRIBUTORY	3	The variant or gene reported here is interpreted to be related to the diagnosis
CAUSATIVE	4	The variant or gene reported here is interpreted to be causative of the diagnosis

4.13.2 Example

```
genomicInterpretation:
    subjectOrBiosampleId: "subject 1"
    interpretationStatus: "CONTRIBUTORY"
    variantInterpretation:
      acmgPathogenicityClassification: "PATHOGENIC"
      variationDescriptor:
        expressions:
        - syntax: "hgvs"
          value: "NM_001848.2:c.877G>A"
        allelicState:
          id: "GENO:0000135"
          label: "heterozygous"
```

4.13.3 Explanations

subject_or_biosample_id

Each genomic interpretation is based on a genomic finding in the germline DNA of the *Individual* referenced in the phenopacket or of a *Biosample* derived from the individual. The id used here must therefore match with the Individual.id or with the Biosample.id element.

interpretation_status

This is an enumeration that describes the conclusion made about the genomic interpretation.

- UNKNOWN_STATUS: unknown
- REJECTED: the variant or gene reported here is interpreted *not* to be related to the diagnosis
- CANDIDATE: the variant or gene reported here is interpreted to *possibly* be related to the diagnosis
- CONTRIBUTORY: the variant or gene reported here is interpreted to be related to the diagnosis
- CAUSATIVE: the variant or gene reported here is interpreted to be causative of the diagnosis

In an autosomal dominant Mendelian disease, one variant is causative. In this case, one would classify it as `CAUSATIVE` and the *Interpretation* object that contains the genomic interpretation would use `SOLVED`. Similarly in the case of an autosomal recessive disease, one would classify a homozygous variant as `CAUSATIVE`. There

are several situations in which one should use `CONTRIBUTORY`. In the case of an autosomal recessive disease, two `CONTRIBUTORY` genomic interpretations would be used for compound heterozygous variants. In cancer, `CONTRIBUTORY` can be used for multiple variants, and the corresponding *Interpretation* object could classify them as `ACTIONABLE`, for instance, if a targeted treatment is available for the variant.

call

Either an *GeneDescriptor* or a *VariantInterpretation* representing the subject of the genomic interpretation.

4.14 GestationalAge

Gestational age (or menstrual age) is the time elapsed between the first day of the last normal menstrual period and the day of delivery. The first day of the last menstrual period occurs approximately 2 weeks before ovulation and approximately 3 weeks before implantation of the blastocyst. Because most women know when their last period began but not when ovulation occurred, this definition traditionally has been used when estimating the expected date of delivery. In contrast, chronological age (or postnatal age) is the time elapsed after birth.

Gestational age is conventionally expressed as completed weeks. Therefore, a 25-week, 5-day fetus is considered a 25-week fetus. Gestational age is often reported as W+D. For instance, 33 weeks and 2 days could be reported as 33+2.

4.14.1 Data model

Table 8: Definition of the `GestationalAge` element

Field	Type	Multiplicity	Description
weeks	int32	1..1	Completed weeks of gestation according to the above definition. REQUIRED.
days	int32	0..1	RECOMMENDED, If available

The following shows how the element can be used to report the gestational age of 33 weeks and 2 days.

```
gestationalAge:
    weeks: 33
    days: 2
```

The gestational age element is intended for use in phenopackets that describe prenatal clinical data.

4.15 Individual

The subject of the Phenopacket is represented by an *Individual* element. This element intends to represent an individual human or other organism. In this documentation, we explain the element using the example of a human proband in a clinical investigation.

4.15.1 Data model

Field	Type	Multiplicity	Description
id	string	1..1	An arbitrary identifier. REQUIRED.
alternate_ids	a list of *CURIE*	0..*	A list of alternative identifiers for the individual
date_of_birth	timestamp	0..1	A timestamp either exact or imprecise
time_at_last_encounter	*TimeElement*	0..1	The age or age range of the individual when last encountered. RECOMMENDED.
vital_status	*VitalStatus*	0..1	The vital status of the individual e.g. whether they are alive or the time and cause of death. RECOMMENDED.
sex	*Sex*	0..1	Observed apparent sex of the individual
karyotypic_sex	*KaryotypicSex*	0..1	The karyotypic sex of the individual
taxonomy	*OntologyClass*	0..1	an *OntologyClass* representing the species (e.g., NCBITaxon:9615)

4.15.2 Example

The following example is typical but does not make use of all of the optional fields of this element.

```
individual:
  id: "patient:0"
  dateOfBirth: "1998-01-01T00:00:00Z"
  sex: "MALE"
```

4.15.3 Explanations

id

This element is the **primary** identifier for the individual and SHOULD be used in other parts of a message when referring to this individual - for example in a *Pedigree* or *Biosample*. The contents of the element are context dependent, and will be determined by the application. For instance, if the Phenopacket is being used to represent a case study about an individual with some genetic disease, the individual may be referred to in that study by their position in the pedigree, e.g., III:2 for the second person in the third generation. In this case, id would be set to `III:2`.

If a *Pedigree* element is used, it is essential that the `individual_id` of the *Pedigree* element matches the `id` field here.

If a *Biosample* element is used, it is essential that the `individual_id` of the *Biosample* element matches the `id` field here.

All identifiers within a phenopacket pertaining to an individual SHOULD use this identifier. It is the responsibility of the sender to provide the recipient an internally consistent message. This is possible as all messages can be created dynamically be the sender using identifiers appropriate for the receiving system.

For example, a hospital may want to send a *Family* to an external lab for analysis. Here the hospital is providing an obfuscated identifier which is used to identify the individual in the *Phenopacket*, the *Pedigree* and mappings to the sample id in the *File*.

In this case the *Pedigree* is created by the sending system from whatever source they use and the identifiers should be mapped to those *Individual.id* contained in the *Family.proband* and *Family.relatives* phenopackets.

In the case of the VCF file, the sending system likely has no control or ability to change the identifiers used for the sample id and it is likely they use different identifiers. It is for this reason the *File* has a *local* mapping field *HtsFile.individual_to_sample_identifiers* where the *Individual.id* can be mapped to the sample id in that file.

example

In this example we show individual blocks which would be used as part of a singleton 'family' to illustrate the use of the internally consistent *Individual.id*. As noted above, the data may have been constructed by the sender from different sources but given they know these relationships, they should provide the receiver with a consistent view of the data both for ease of use and to limit incorrect mapping.

Thus, we would use the same id various elements.

```
individual:
  id: "patient23456"
  dateOfBirth: "1998-01-01T00:00:00Z"
  sex: "MALE"
```

Assuming that this individual was sequenced, we might have the following *File* element.

```
htsFile:
    uri: "file://data/genomes/germline_wgs.vcf.gz"
    description: "Matched normal germline sample"
    htsFormat: "VCF"
    genomeAssembly: "GRCh38"
    individualToSampleIdentifiers:
        patient23456: "NA12345"
```

We would also use `patient23456` as the `individualId` element within a *Pedigree* element.

alternate_ids

An optional list of alternative identifiers for this individual. These should be in the form of rstcurie's and hence have a corresponding :ref:'rstresource listed in the *MetaData*. These should **not** be used elsewhere in the phenopacket as this will break the assumptions required for using the `id` field as the primary identifier. This field is provided for the convenience of users who may have multiple mappings to an individual which they need to track.

date_of_birth

This element represents the date of birth of the individual as an ISO8601 UTC timestamp that is rounded down to the closest known year/month/day/hour/minute. For example:

- "2018-03-01T00:00:00Z" for someone born on an unknown day in March 2018
- "2018-01-01T00:00:00Z" for someone born on an unknown day in 2018
- empty if unknown/ not stated.

See *here* for more information about timestamps.

The element is provided for use cases within protected networks, but it many situations the element should not be used in order to protect the privacy of the individual. Instead, the `time_at_last_encounter` element should be preferred.

time_at_last_encounter

An object describing when the encounter with the patient happened or the the age of the individual at the time of collection of biospecimens or phenotypic observations reported in the current Phenopacket. It is specified using either an *TimeElement*, which can represent an time in several different ways, either precisely or within a range. For example an *Age* or an *AgeRange* element, which can represent a range of ages such as 10-14 years (age can be represented in this was to protect privacy of study participants).

vital_status

The *VitalStatus* can be used to report whether the individual is living or dead at the timepoint when the phenopacket was created (or if the status is unknown).

sex

Phenopackets make use of an enumeration to denote the phenotypic sex of an individual. See *Sex*.

karyotypic_sex

Phenopackets make use of an enumeration to denote the chromosomal sex of an individual. See *KaryotypicSex*.

taxonomy

For resources where there may be more than one organism being studied it is advisable to indicate the taxonomic identifier of that organism, to its most specific level. We advise using the codes from the NCBI Taxonomy resource. For instance, NCBITaxon:9606 is human (homo sapiens sapiens) and or NCBITaxon:9615 is dog.

4.16 Interpretation

This message intends to represent the interpretation of a genomic analysis, such as the report from a diagnostic laboratory.

4.16.1 Data model

Field	Type	Multiplicity	Description
id	string	1..1	Arbitrary identifier. REQUIRED.
progress_status	*ProgressStatus*	1..1	The current resolution status. REQUIRED.
diagnosis	*Diagnosis*	0..*	One or more diagnoses, if made
summary	string	0..1	Additional data about this interpretation

4.16.2 Example

In this example, a case with id `CONSORTIUM:0000123456` is reported to be solved. The diagnosis is Miller syndrome, and the supporting interpretation states the involved gene. For privacy reasons, the variant was not reported, but the intended meaning is that a relevant variant in the named gene was found.

```
interpretation:
  id: "CONSORTIUM:0000123456"
  progressStatus: "SOLVED"
  diagnosis:
    disease:
      id: "OMIM:263750"
      label: "Miller syndrome"
    genomicInterpretations:
    - interpretationStatus: "CONTRIBUTORY"
      gene:
        valueId: "HGNC:2867"
        symbol: "DHODH"
```

4.16.3 Explanations

id

The id has the same interpretation as the **id** element in the *Individual* element.

ProgressStatus

The interpretation has a **ProgressStatus** that refers to the status of the attempted diagnosis.

Implementation note - this is an enumerated type, therefore the values represented below are the only legal values. The value of this type SHALL NOT be null, instead it SHALL use the 0 (zero) ordinal element as the default value, should none be specified.

Name	Ordinal	Description
UNKNOWN_PROGRESS	0	No information is available about the diagnosis
IN_PROGRESS	1	No diagnosis has been found to date but additional differential diagnostic work is in progress.
COMPLETED	2	The work on the interpretation is complete.
SOLVED	3	The interpretation is complete and also considered to be a definitive diagnosis
UNSOLVED	4	The interpretation is complete but no definitive diagnosis was found

Diagnosis

The diagnosis element is meant to refer to the disease that is inferred to be present in the individual or family being analyzed. The diagnosis can be made by means of an analysis of the phenotypic or the genomic findings or both. The element is optional because if the **resolution_status** is **UNSOLVED** then there is no diagnosis.

Data elements

Field	Type	Multiplicity	Description
disease	*OntologyClass*	1..1	The diagnosed condition. REQUIRED.
genomic_interpretations	*GenomicInterpretation*	0..*	The genomic elements assessed as being responsible for the disease or empty

4.16.4 Examples of the intended usage of the Interpretation element

Candidate genes

Research consortia may exchange information about candidate genes in which an undisclosed variant was found that was assessed to be possibly related to a disease or phenotype but for which insufficient evidence is available to be certain. The intention is often to find other researchers with similar cases in order to subsequently share detailed information in a collaborative project.

In this case, the gene should be marked as `CANDIDATE`. Here is an example of an interpretation with the hypothetical gene YFG42.

```
interpretation:
  id: "CONSORTIUM:0000123456"
  progressStatus: "SOLVED"
  diagnosis:
    disease:
      id: "OMIM:263750"
      label: "Miller syndrome"
    genomicInterpretations:
    - interpretationStatus: "CONTRIBUTORY"
      gene:
        valueId: "HGNC:2867"
        symbol: "DHODH"
```

Diagnostic finding in an autosomal dominant disease

The `Interpretation` element might be used in this way to report a laboratory finding in a diagnostic setting or in a published case report. The following example shows how the variant NM_000138.4(FBN1):c.6751T>A (p.Cys2251Ser) would be reported.

```
interpretation:
  id: "Arbitrary interpretation id"
  progressStatus: "SOLVED"
  diagnosis:
    disease:
      id: "OMIM:154700"
      label: "Marfan syndrome"
    genomicInterpretations:
    - subjectOrBiosampleId: "subject 1"
      interpretationStatus: "CONTRIBUTORY"
      variantInterpretation:
        acmgPathogenicityClassification: "PATHOGENIC"
        variationDescriptor:
          expressions:
          - syntax: "hgvs"
            value: "NM_000138.4(FBN1):c.6751T>A"
          allelicState:
            id: "GENO:0000135"
            label: "heterozygous"
```

The `subjectOrBiosampleId` is set to the id of the *Individual* of the enclosing phenopacket to indicate that the genomic interpretation refers to a germline variant.

Diagnostic finding in an autosomal recessive disease

For homozygous variants, the `zygosity` would be set to homozygous. The following example shows a finding of compound heterozygous variants.

```
interpretation:
  id: "Arbitrary interpretation id"
  progressStatus: "SOLVED"
  diagnosis:
    disease:
      id: "OMIM: 219700"
      label: "Cystic fibrosis"
    genomicInterpretations:
    - subjectOrBiosampleId: "subject 1"
      interpretationStatus: "CONTRIBUTORY"
      variantInterpretation:
        acmgPathogenicityClassification: "PATHOGENIC"
        variationDescriptor:
          expressions:
          - syntax: "hgvs"
            value: "NM_000492.3(CFTR):c.1477C>T (p.Gln493Ter)"
          allelicState:
            id: "GENO:0000135"
            label: "heterozygous"
    - subjectOrBiosampleId: "subject 1"
      interpretationStatus: "CONTRIBUTORY"
      variantInterpretation:
        acmgPathogenicityClassification: "PATHOGENIC"
        variationDescriptor:
          expressions:
          - syntax: "hgvs"
            value: "NM_000492.3(CFTR):c.1521_1523delCTT (p.Phe508delPhe)"
          allelicState:
            id: "GENO:0000135"
            label: "heterozygous"
```

The `subjectOrBiosampleId` is set to the id of the *Individual* of the enclosing phenopacket to indicate that the genomic interpretation refers to a germline variant.

Diagnostic finding in a cancer

Cancer cases are not generally solved by genomic analysis. Instead, the intention is often to identify actionable variants that represent potential indications for targeted therapy. In this example, a BRAF variant is interpreted as being actionable in this sense.

```
interpretation:
 id: "Arbitrary interpretation id"
 progressStatus: "COMPLETED"
 diagnosis:
   disease:
     id: "NCIT:C3224"
     label: "Melanoma"
   genomicInterpretations:
   - subjectOrBiosampleId: "biosample id"
     interpretationStatus: "CONTRIBUTORY"
     variantInterpretation:
```

(continues on next page)

(continued from previous page)

```
      acmgPathogenicityClassification: "PATHOGENIC"
      therapeuticActionability: "ACTIONABLE"
      variationDescriptor:
        expressions:
        - syntax: "hgvs"
          value: "NM_001374258.1(BRAF):c.1919T>A (p.Val640Glu)"
        allelicState:
          id: "GENO:0000135"
          label: "heterozygous"
```

The `subjectOrBiosampleId` is set to the id of the *Biosample* that is contained in the enclosing phenopacket, representing a biopsy from a melanoma sample taken from the subject of the phenopacket.

4.17 KaryotypicSex

This enumeration represents the chromosomal sex of an individual.

4.17.1 Data model

Implementation note - this is an enumerated type, therefore the values represented below are the only legal values. The value of this type SHALL NOT be null, instead it SHALL use the 0 (zero) ordinal element as the default value, should none be specified.

Name	Ordinal	Description
UNKNOWN_KARYOTYPE	0	Untyped or inconclusive karyotyping
XX	1	Female
XY	2	Male
XO	3	Single X chromosome only
XXY	4	Two X and one Y chromosome
XXX	5	Three X chromosomes
XXYY	6	Two X chromosomes and two Y chromosomes
XXXY	7	Three X chromosomes and one Y chromosome
XXXX	8	Four X chromosomes
XYY	9	One X and two Y chromosomes
OTHER_KARYOTYPE	10	None of the above types

4.18 Measurement

The measurement element is used to record individual measurements. It can capture quantitative, ordinal (e.g., absent/present), or categorical measurements.

4.18.1 Data model

Table 9: Definition of the `Quantity` element

Field	Type	Multiplicity	Description
description	string	0..1	free text.
assay	*OntologyClass*	1..1	Class that describes the assay used to produce the measurement. REQUIRED.
measurement_value	one of {*Value* | *ComplexValue*}	1..1	The result of the measurement. REQUIRED.
time_observed	*TimeElement*	0..1	Time at which measurement was performed. RECOMMENDED.
procedure	*Procedure*	0..1	Clinical procdure performed to acquire the sample used for the measurement

4.18.2 Examples

The following example shows measurement of platelet count. The result is abnormally low, but in general this element can be used to represent normal or abnormal measurements.

```
measurement:
    assay:
        id: "LOINC:26515-7"
        label: "Platelets [#/volume] in Blood"
    value:
        quantity:
            unit:
                id: "UO:0000316"
                label: "cells per microliter"
            value: 24000.0
    referenceRange:
        unit:
            id: "UO:0000316"
            label: "cells per microliter"
        low: 150000.0
        high: 450000.0
    timeObserved:
        timestamp: "2020-10-01T10:54:20.021Z"
```

The following example shows an ordinal measurement. The measurement is for nitrite in urine, and the result is positive (present).

```
measurement:
    assay:
        id: "LOINC:5802-4"
        label: "Nitrite [Presence] in Urine by Test strip"
    value:
        ontologyClass:
            id: "NCIT:C25626"
            label: "Present"
    timeObserved:
        timestamp: "2021-01-01T10:54:20.021Z"
```

This element represents a specific measurement. It may also be appropriate to represent the result of this test as a *PhenotypicFeature* using the HPO term Nitrituria. Which option to use depends on the analysis goals. The measure-

ment object is intended to represent specific measurements, and the *PhenotypicFeature* is often used to represent a conclusion that is reached on the basis of the test.

Categorical measurements, in which the outcome of the measurement is represented by one of two or more options that are not ordered, are represented in an analogous fashion.

The following example presents a blood pressure measurement. The measurement of blood pressure consists of two measurements (systolic and diastolic), that are represented as a rstcomplexquantity.

```
measurement:
    assay:
      id: "CMO:0000003"
      label: "blood pressure measurement"
    complexValue:
      typedQuantities:
      - type:
          id: "NCIT:C25298"
          label: "Systolic Blood Pressure"
        quantity:
            unit:
              id: "NCIT:C49670"
              label: "Millimeter of Mercury"
            value: 125.0
      - type:
          id: "NCIT:C25299"
          label: "Diastolic Blood Pressure"
        quantity:
          unit:
            id: "NCIT:C49670"
            label: "Millimeter of Mercury"
          value: 75.0
    timeObserved:
      timestamp: "2021-01-01T10:54:20.021Z"
```

4.18.3 Explanations

description

Free-text description of the feature. Note this is not a acceptable place to document/describe t he phenotype - the type and onset etc… fields should be used for this purpose.

assay

An ontology class which describes the assay used to produce the measurement. For example “body temperature” (CMO:0000015) or “Platelets [#/volume] in Blood” (LOINC:26515-7) FHIR mapping: Observation.code

value

This element represents the result of the measurement. The measurement can be quantitative, such as LOINC:2472-9 (IgM [Mass/volume] in Serum or Plasma) or ordinal or categorical.

complex_value

This is intended to represent measurements that consist of a tightly coupled group of related quanitities. For instance, blood pressure represents a measurement of systolic and diastolic blood pressure.

time_observed

The time at which the measurement was made.

procedure

Clinical procedure performed on the subject in order to obtain the sample used for the measurement. Examples include blood draw and biopsy. If the procedure can be inferred from the measurement or if the details of the measurement are deemed unimportant (e.g., a blood glucose test is performed on a blood sample obtained with some procedure that is not specified), this element can be omitted.

4.19 MedicalAction

This element describes medications, procedures, other actions taken for clinical management. The element is a list of options.

4.19.1 Data model

Table 10: Definition of the `MedicalAction` element

Field	Type	Multiplicity	Description
action	one of {*Procedure* | *Treatment* | *RadiationTherapy* | *TherapeuticRegimen*}	1..1	One of a list of medical actions. REQUIRED.
treatment_target	*OntologyClass*	0..1	The condition or disease that this treatment was intended to address
treatment_intent	*OntologyClass*	0..1	Whether the intention of the treatment was curative, palliative...
response_to_treatment	*OntologyClass*	0..1	How the patient responded to the treatment
adverse_events	*OntologyClass* (List)	0..*	Any adverse effects experienced by the patient attributed to the treatment
treatment_termination_reason	*OntologyClass*	0..1	The reason that the treatment was stopped.

action

Each MedicalAction element refers to one of the following specific types of medical action:

- *Procedure*
- *Treatment*
- *RadiationTherapy*
- *TherapeuticRegimen*

4.20 MetaData

This element contains structured definitions of the resources and ontologies used within the phenopacket. It is considered to be a required element of a valid Phenopacket and application Q/C software should check this.

4.20.1 Data model

Table 11: Definition of the `MetaData` element

Field	Type	Multiplicity	Description
created	A Timestamp	1..1	Representation of the time when this object was created, e.g., 2019-04-01T15:10:17.808Z
created_by	string	1..1	Name of person who created the phenopacket
submitted_by	string	0..1	Name of person who submitted the phenopacket
resources	list of *Resource*	1..*	Ontologies used to create the phenopacket
updates	list of *Update*	0..*	List of updates to the phenopacket
phenopacket_schema_version	string	1..1	schema version of the current phenopacket
external_references	List of *ExternalReference*	0..*	(See text)

The *MetaData* element MUST have one *Resource* element for each ontology or terminology whose terms are used in the Phenopacket. For instance, if a MONDO term is used to specify the disease and HPO terms are used to specify the phenotypes of a patient, then the *MetaData* element MUST have one *Resource* element each for MONDO and HPO.

4.20.2 Example

```
metadata:
  created: "2019-07-21T00:25:54.662Z"
  createdBy: "Peter R."
  resources:
      - id: "hp"
      name: "human phenotype ontology"
      url: "http://purl.obolibrary.org/obo/hp.owl"
      version: "2018-03-08"
      namespacePrefix: "HP"
      iriPrefix: "hp"
      - id: "geno"
      name: "Genotype Ontology"
      url: "http://purl.obolibrary.org/obo/geno.owl"
      version: "19-03-2018"
      namespacePrefix: "GENO"
      iriPrefix: "geno"
      - id: "pubmed"
      name: "PubMed"
      url: "https://www.ncbi.nlm.nih.gov/pubmed/"
      namespacePrefix: "PMID"
  phenopacketSchemaVersion: "2.0"
  externalReferences:
      - id: "PMID:30808312"
      description: "Bao M, et al. COL6A1 mutation leading to Bethlem myopathy with␣
↪recurrent hematuria: a case report. BMC Neurol. 2019;19(1):32."
```

4.20.3 Explanations

created

This element is a ISO8601 UTC timestamp for when this phenopacket was created in ISO, e.g., "2018-03-01T00:00:00Z".

created_by

This is a string that represents an identifier for the contributor/ program. The expected syntax and semantics are application-dependent.

submitted_by

This is a string that represents an identifier for the person who submitted the phenopacket (who may not be the person who created the phenopacket).

resources

This element contains a listing of the ontologies/resources referenced in the phenopacket.

updates

This element contains a list of *Update* objects which contain information about when, what and who updated a phenopacket. This is only necessary when a phenopacket is being used as a persistent record and is being continuously updated. Resources should provide information about how this is being used.

phenopacket_schema_version

A string representing the version of the phenopacket-schema according to which a phenopacket was made. Permitted values MUST be one of *1.0.0*, *1.0* or *2.0*. Versions *1.0.0* and *1.0* are equivalent and the *1.0* string should be preferred. This version of the schema is *2.0*.

external_references

A list of *ExternalReference* (such as the PubMed id of a publication from which a phenopacket was derived).

4.21 OntologyClass

This element is used to represent classes (terms) from ontologies, and is used in many places throughout the Phenopacket standard. It is a simple, two element message that represents the identifier and the label of an ontology class.

The ID SHALL be a CURIE-style identifier e.g. HP:0100024, MP:0001284, UBERON:0001690, i.e., the primary key for the ontology class. The label should be the corresponding class name. The Phenopacket standard REQUIRES that the id and the label match in the original ontology. We note that occasionally, ontology maintainers change the primary label of a term.

4.21.1 Data model

Field	Type	Multiplicity	Description
id	string	1..1	a CURIE-style identifier e.g. HP:0001875. REQUIRED.
label	string	1..1	human-readable class name e.g. Neutropenia. REQUIRED.

4.21.2 Example

```
ontologyClass:
    id: "HP:0001875"
    label: "Neutropenia"
```

4.21.3 Explanations

id

The ID of an OntologyClass element MUST take the form of a *CURIE* format. In order that the class is resolvable, it MUST reference the namespace prefix of a *Resource* named in the *MetaData*.

label

The the human-readable label for the concept. This MUST match the ID in the ontology referenced by the namespace prefix in a *Resource* named in the *MetaData*.

4.22 Pedigree

This element is used to represent a pedigree to describe the family relationships of each sample along with their gender and phenotype (affected status). The information in this element is for use by programs for analysis of a multi-sample VCF file with exome or genome sequences of members of a family, some of whom are affected by a Mendelian disease.

The phenopacket schema has implemented a PED-compatible data-model to promote interoperability between existing PED files and PED software, but does not actually store a PED file.

See the detailed description at the PLINK website for more information about PED files.

4.22.1 Data model

Field	Type	Multiplicity	Description
persons	list of *Person*	1..*	list of family members in this pedigree. REQUIRED.

The pedigree is simply a list of Person objects. These objects reflect the elements of a PED file.

Person

The *Person* class represents a row from the PED file indicating the biological parents of the individual, their sex and their *AffectedStatus*.

Table 12: Definition of the `Person` element

Field	Type	Multiplicity	Description
family_id	string	1..1	application specific identifier. REQUIRED.
individual_id	string	1..1	application specific identifier. REQUIRED.
paternal_id	string	1..1	application specific identifier. REQUIRED.
maternal_id	string	1..1	application specific identifier. REQUIRED.
sex	*Sex*	1..1	see text. REQUIRED.
affected_status	*AffectedStatus*	1..1	see text. REQUIRED.

4.22.2 Example

Here we show a pedigree in PED format, this contains two male siblings which share an abnormal (affected) phenotype and their two normal (unaffected) parents.

Below we show the same pedigree as a phenopacket *Pedigree* in YAML format.

```yaml
pedigree:
 persons:
 - familyId: "family 1"
   individualId: "kindred 1A"
   paternalId: "FATHER"
   maternalId: "MOTHER"
   sex: "MALE"
   affectedStatus: "AFFECTED"
 - familyId: "family 1"
   individualId: "kindred 1B"
   paternalId: "FATHER"
   maternalId: "MOTHER"
   sex: "MALE"
   affectedStatus: "AFFECTED"
 - familyId: "family 1"
   individualId: "MOTHER"
   paternalId: "0"
   maternalId: "0"
   sex: "FEMALE"
   affectedStatus: "UNAFFECTED"
 - familyId: "family 1"
   individualId: "FATHER"
   paternalId: "0"
   maternalId: "0"
   sex: "MALE"
   affectedStatus: "UNAFFECTED"
```

AffectedStatus

This element is an enumeration to

Name	Description
MISSING	It is unknown if the individual has the affected phenotype
UNAFFECTED	The individual does not show the affected phenotype of the proband
AFFECTED	The individual has the affected phenotype of the proband

In a PED file, affected persons are encoded with "2", and unaffecteds by "1" (a "0" is used if no information is available). Instead, Phenopackets uses an enumeration as shown in the table.

In a PED file, the sex of individuals is encoded as a "1" for females, "2" for males, and "0" for unknown. Phenopackets uses *Sex* instead.

The message is made up of a list of `Person` elements (the Person element is defined within the Pedigree element). Each Person element is equivalent to one row of a PED file.

The family ID and the individual IDs may be made up of letters and digits, and the combination of family and individual ID should uniquely identify each person represented in the PED file. The parents of a person in the pedigree are shown with the corresponding individual IDs. Individuals whose parents are not represented in the PED file are known as founders; their parents are represented by a zero ("0") in the columns for mother and father. Finally, the sex and the affected (disease) status of the person are shown.

If a `Phenopacket` is used to represent any of the individuals listed in the `Pedigree`, then it is essential that the `individual_id` used in the pedigree matches the `id` of the `subject` of the `Phenopacket`. It is allowable for the `Pedigree` to have individuals that do not have an associated `Phenopacket`. This is useful, for instance, if the `Pedigree` is being used to store the affected/not affected status of family members being examined by exome or genome sequencing. In this case (i.e. where there are no associated phenopackets for the `Pedigree.individual_id`), it is expected that the `individual_id` elements match the sample identifiers of the exome/genome file.

4.23 PhenotypicFeature

This element is intended to be used to describe a phenotype that characterizes the subject of the Phenopacket. For medical use cases the subject will generally be a patient or a proband of a study, and the phenotypes will be abnormalities described by an ontology such as the Human Phenotype Ontology. The word phenotype is used with many different meanings including disease entity, but in this context we mean An individual phenotypic feature, observed as either present or absent (excluded), with possible onset, modifiers and frequency.

4.23.1 Data model

Field	Type	Multiplicity	Description
description	string	optional	human-readable verbiage **NOT** for structured text
type	*OntologyClass*	1..1	term denoting the phenotypic feature. REQUIRED.
excluded	boolean	0..1	defaults to *false*
severity	*OntologyClass*	0..1	description of the severity of the feature described in *type*. For instance terms from HP:0012824
modifiers	list of *OntologyClass*	0..*	For instance one or more terms from HP:0012823
onset	*TimeElement*	0..1	Age or time at which the feature was first observed.
resolution	*TimeElement*	0..1	Age or time at which the feature resolved or abated.
evidence	*Evidence*	0..*	the evidence for an assertion of the observation of a *type*. RECOMMENDED.

4.23.2 Example

The following example specifies recurrent Infantile spasms, which had onset at age 6 months and resolved at age 4 years and 2 months.

```
phenotypicFeature:
    type:
        id: "HP:0012469"
        label: "Infantile spasms"
    modifiers:
        - id: "HP:0031796"
        label: "Recurrent"
    onset:
        age:
            iso8601duration: "P6M"
    resolution:
        age:
            iso8601duration: "P4Y2M"
```

4.23.3 Explanations

description

This element represents a free-text description of the phenotype. It should not be used as the primary means of describing the phenotype, but can be used to supplement the record if ontology terms are not sufficiently able to capture all the nuances. In general, the type and onset etc… fields should be used for this purpose, and this field is a last resort.

type

The element represents the primary *ontology class* which describes the phenotype. For example Craniosynostosis (HP:0001363).

excluded

This element is a flag to indicate whether the phenotype was observed or not. The default is 'false', in other words the phenotype was observed. Therefore it is only required in cases to indicate that the phenotype was looked for, but found to be absent.

severity

This element is an *ontology class* that describes the severity of the condition e.g. subclasses of Severity (HP:0012824) or SNOMED:272141005-Severities

modifiers

This element is a list of *ontology class* elements that can be empty or contain one or more ontology terms that are intended to provide more expressive or precise descriptions of a phenotypic feature, including attributes such as positionality and external factors that tend to trigger or ameliorate the feature. Terms can be taken from the hierarchy of Clinical modifier in the HPO (noting that severity should be coded in the severity element).

onset

This element can be used to describe the age at which a phenotypic feature was first noticed or diagnosed. For many medical use cases, either the Age sub-element or an *ontology class* (e.g., from the HPO Onset (HP:0003674) terms) will be used.

resolution

This element can be used to describe the age or time when a phenotypic feature resolved (disappeared, got better). In the example shown above, infantile spasms no longer occured after the age of 4 years and 2 months.

evidence

This element is recommended and contain one or more *Evidence* elements that specify how the phenotype was determined.

4.24 Procedure

The Procedure element represents a clinical procedure performed on a subject. For example a surgical or diagnostic procedure such as a biopsy.

If the Procedure element is used, it must contain a `code` element, but only need contain the body_site element if needed.

4.24.1 Data model

Field	Type	Multiplicity	Description
code	*OntologyClass*	1..1	clinical procedure performed. REQUIRED.
body_site	*OntologyClass*	0..*	specific body site where the procedure was performed
performed	*TimeElement*	0..*	age/time when the procedure was performed

4.24.2 Example

In this example, a skin biopsy from the skin of the forearm is perfomed on an individual who is 25 years old.

```
procedure:
    code:
        id: "NCIT:C28743"
        label: "Punch Biopsy"
    bodySite:
        id: "UBERON:0003403"
        label: "skin of forearm"
    performed:
        age:
            iso8601duration: "P25Y"
```

4.24.3 Explanations

code

This element is an *OntologyClass* that represents clinical procedure performed on a subject. For instance, Biopsy of Soft Palate would be represented as follows.

```
{
    "code": {
        "id": "NCIT:C51585",
        "label": "Biopsy of Soft Palate"
    }
}
```

body site

In cases where it is not possible to represent the procedure adequately with a single *OntologyClass*, the body site should be indicated using a separate ontology class. For instance, the following indicates a punch biopsy on the skin of the forearm.

```
{
    "code" {
        "id": "NCIT:C28743",
        "label": "Punch Biopsy"
    },
    "bodySite" {
        "id": "UBERON:0003403",
        "label": "skin of forearm"
    }
}
```

4.25 Quantity

This element is meant to denote quantities of items such as medications. The unit of a dose can be expressed with NCIT terms such as Milligram, Microgram, or Unit. The value should be expressed as a number.

4.25.1 Data model

Table 13: Definition of the `Quantity` element

Field	Type	Multiplicity	Description
unit	*OntologyClass*	1..1	The kind of unit. REQUIRED.
value	double	1..1	the value of the quantity in the units e.g. 2.0 mg. REQUIRED.
reference_range	*ReferenceRange*	0..1	the normal range for the *value*

4.25.2 Examples

The following message could be used to represent the quantity corresponding to a 15 mg tablet of Meloxicam.

```
unit:
  id: "NCIT:C28253"
  label: "Milligram"
value: 15.0
```

The following message could be used to represent the quantity corresponding to a bolus of 5000 units of Heparin.

```
unit:
  id: "NCIT:C44278"
  label: "Unit"
value: 5000
```

The following example shows a quantity for a platelet count per microliter, with a reference range.

```
unit:
  id: "UO:0000316"
  label: "cells per microliter"
value: 300000.0
referenceRange:
  unit:
    id: "UO:0000316"
    label: "cells per microliter"
  low: 150000.0
  high: 450000.0
```

4.25.3 Explanations

unit

Ontology terms for units can be taken from the National Cancer Institute Thesaurus (NCIT), from the subhierarchy for Unit of Measure (Code C25709).

value

The corresponding value of the quantity in the indicated units.

4.26 RadiationTherapy

Radiation therapy (or radiotherapy) uses ionizing radiation, generally as part of cancer treatment to control or kill malignant cells.

4.26.1 Data model

Table 14: Definition of the `Radiotherapy` element

Field	Type	Multiplicity	Description
modality	*OntologyClass*	1..1	The modality of radiation therapy (e.g., electron, photon,...). REQUIRED.
body_site	*OntologyClass*	1..1	The anatomical site where radiation therapy was administered. REQUIRED.
dosage	int32	1..1	the total dose given in units of Gray (Gy). REQUIRED.
fractions	int32	1..1	The total number of fractions delivered as part of treatment. REQUIRED.

4.26.2 Explanations

modality

Terms from the NCIT can be used to represent the modality of radiation therapy. Four examples are shown here:

- Electron Beam (NCIT:C28039)
- Proton Radiation (NCIT:C40431).
- Photon Beam Radiation Therapy (NCIT:C104914)
- High-LET Heavy Ion Therapy (NCIT:C15458).

The NCIT terms themselves specify whether the radiation therapy was administered externally or internally. For instance, NCIT terms from the Internal Radiation Therapy (NCIT:C15195) subhierarchy would be used to indicate a type of internal radiation therapy.

body_site

The anatomical site that was irradiated. An uberon term can be used.

dosage

The total dosage administered, indicated in units of Gray.

fractions

The number of fractions into which the radiation dosage was divided.

4.27 ReferenceRange

This elements is provided to support the *Measurement* element, which can be used to report a numerical value such as LOINC:26515-7, Platelets [#/volume] in Blood. The normal range for circulating platelets is 150,000 to 450,000 platelets per microliter.

4.27.1 Data model

Field	Type	Multiplicity	Description
unit	*OntologyClass*	required	Ontology term describing the unit.
low	double	1..1	lower range of normal. REQUIRED.
high	double	1..1	upper range of normal. REQUIRED.

4.27.2 Example

There are several ontologies that provide terms for units of measurement, including the Units of measurement ontology and the National Cancer Institute Thesaurus (NCIT), from the subhierarchy for Unit of Measure (Code C25709).

The following example shows the reference range for platelet count per microliter.

```
referenceRange:
  unit:
      id: "UO:0000316"
      label: "cells per microliter"
  low: 150000.0
  high: 450000.0
```

4.28 Resource

The `Resource` element is a description of an external resource used for referencing an object. For example the resource may be an ontology such as the HPO or SNOMED or another data resource such as the HGNC or ClinVar.

A `Resource` is used to contain data used to expand *CURIE* identifiers when used in an `id` field. This is known as *Identifier resolution*.

The *MetaData* element uses one resource element to describe each resource that is referenced in the Phenopacket.

4.28.1 Data model

Field	Type	Multiplicity	Description
id	string	1..1	Resource identifier. e.g. *hp*. REQUIRED.
name	string	1..1	Formal name of the resource or ontology e.g. human phenotype ontology. REQUIRED.
namespace_prefix	1..1	required	Namespace prefix of the resource e.g *HP*. REQUIRED.
url	string	1..1	Uniform Resource Locator of the resource. REQUIRED.
version	string	1..1	The version string for the ontology or resource 2018-03-08. REQUIRED.
iri_prefix	string	1..1	Internationalized Resource Identifier such as `http://purl.obolibrary.org/obo/HP_`. REQUIRED.

4.28.2 Example

For an ontology, the url SHALL point to the obo or owl file, e.g. This information can also be found at the EBI Ontology Lookup Service

```
resource:
  id: "hp"
  name: "Human Phenotype Ontology"
  url: "http://www.human-phenotype-ontology.org"
  version: "2018-03-08"
  namespacePrefix: "HP"
  iriPrefix: "http://purl.obolibrary.org/obo/HP_"
```

Non-ontology resources which use CURIEs as their native identifiers should be treated in a similarly resolvable manner.

```
resource:
  id: "hgnc"
  name: "HUGO Gene Nomenclature Committee"
  url: "https://www.genenames.org"
  version: "2019-08-08"
  namespacePrefix: "HGNC"
  iriPrefix: "https://www.genenames.org/data/gene-symbol-report/#!/hgnc_id/"
```

Using this *Resource* definition it is possible for software to resolve the identifier *HGNC:12805* to https://www.genenames.org/data/gene-symbol-report/#!/hgnc_id/12805

Non-ontology resources which do *not* use CURIEs as their native identifiers MUST use the namespace from identifiers.org, when present. For example the UniProt Knowledgebase (https://registry.identifiers.org/registry/uniprot)

```
resource:
  id: "uniprot"
  name: "UniProt Knowledgebase"
  url: "https://www.uniprot.org"
  version: "2019_07"
  namespacePrefix: "uniprot"
  iriPrefix: "https://purl.uniprot.org/uniprot/"
```

Using this *Resource* definition it is possible for software to resolve the identifier *uniprot:Q8H0D3* to https://purl.uniprot.org/uniprot/Q8H0D3

4.28.3 Explanations

id

For OBO ontologies, the value of this string MUST always be the official OBO ID, which is always equivalent to the ID prefix in lower case. Examples: hp, go, mp, mondo Consult http://obofoundry.org for a complete list.

For other resources which do not use native CURIE identifiers (e.g. SNOMED, UniProt, ClinVar), use the prefix in identifiers.org.

name

The name of the ontology referred to by the id element, for example, *The Human Phenotype Ontology*. For OBO Ontologies, the value of this string SHOULD be the same as the title field on http://obofoundry.org

Other resources should use the official title for that resource. Note that this field is purely for information purposes and software should not encode any assumptions.

url

For OBO ontologies, this MUST be the PURL, e.g. http://purl.obolibrary.org/obo/hp.owl or http://purl.obolibrary.org/obo/hp.obo

Other resources should link to the official or top-level url e.g. https://www.uniprot.org or https://www.genenames.org

version

The version of the resource or ontology used to make the annotation. For OBO ontologies, this SHALL be the versionIRI. For other resources this should be the native version of the resource, e.g UniProt - "2019_08", DbSNP - "153" for resources without release versions, this field should be left blank.

namespace_prefix

The prefix used in the CURIE of an OntologyClass e.g. HP, MP, ECO for example an HPO term will have a CURIE like this - HP:0012828 which should be used in combination with the iri_prefix to form a fully-resolvable IRI.

iri_prefix

The full IRI prefix which can be used with the namespace_prefix and the OntologyClass::id to resolve to an IRI for a term. Tools such as the curie-util (https://github.com/prefixcommons/curie-util) can utilise this to produce fully-resolvable IRIs for an OntologyClass.

CURIE

The CURIE is defined by the W3C as a means of encoding a "Compact URI". It is a simple string taking the form of colon (`:`) separated *prefix* and *reference* elements - *prefix:reference* e.g. HP:0012828 or HGNC:12805.

It is RECOMMENDED to use CURIE identifiers where possible.

Not all resources use CURIEs as identifiers (e.g. SNOMED, UniProt, ClinVar, PubMed). In these cases it is often possible to create a CURIE form of an identifier by using the prefix for that resource from identifiers.org.

Where no CURIE prefix is present in identifiers.org it is possible for a Resource to define a locally-scoped namespace, although if a Phenopacket is being shared publicly this is NOT recommended if the resource is not publicly resolvable.

When using a CURIE identifier a corresponding *Resource* SHALL also be included in the *MetaData* section.

Identifier resolution

A CURIE identifier can be resolved to an external resource using the *Resource* element by looking-up the CURIE *prefix* against the Resource::namespacePrefix and then appending the CURIE *reference* to the Resource::iriPrefix.

For example, software can use the HPO Resource shown above to resolve the following HPO term encoding the concept of `Severe`:

```
ontologyClass:
  id: "HP:0012828"
  label: "Severe"
```

The id HP:0012828 can be split into the *prefix* - 'HP' and *reference* - '0012828'. The 'HP' prefix matches the Resource::namespacePrefix so we can append the reference '0012828' to the Resource::iriPrefix: which produces the URI

> http://purl.obolibrary.org/obo/HP_0012828

the term can be resolved to http://purl.obolibrary.org/obo/HP_0012828

4.29 Sex

An enumeration used to represent the sex of an individual. This element does not represent gender identity or *KaryotypicSex*, but instead represents typical "phenotypic sex", as would be determined by a midwife or physician at birth.

4.29.1 Data model

Implementation note - this is an enumerated type, therefore the values represented below are the only legal values. The value of this type SHALL NOT be null, instead it SHALL use the 0 (zero) ordinal element as the default value, should none be specified.

Name	Ordinal	Description
UNKNOWN_SEX	0	Not assessed or not available. Maps to NCIT:C17998
FEMALE	1	female sex. Maps to NCIT:C46113
MALE	2	male sex. Maps to NCIT:C46112
OTHER_SEX	3	It is not possible to accurately assess the applicability of MALE/FEMALE. Maps to NCIT:C45908

4.29.2 Example

```
{
    "sex": "UNKNOWN_SEX"
}
```

4.30 TherapeuticRegimen

This element represents a therapeutic regimen which will involve a specified set of treatments for a particular condition. It can be thought of as a shorthand for a more specific set of treatments. This element is supposed to reference a more detailed regimen specification

4.30.1 Data model

TherapeuticRegimen

Table 15: Definition of the `TherapeuticRegimen` element

Field	Type	Multiplicity	Description
identifier	{*OntologyClass* | *ExternalReference*}	1..1	An identifier for the regimen. REQUIRED.
start_time	*TimeElement*	0..1	When the regimen was started. RECOMMENDED.
end_time	*TimeElement*	0..1	When the regimen ended. An empty *end_time* with a populated *start_time* would indicate the regimen was ongoing. RECOMMENDED.
status	*RegimenStatus*	1..1	Current status of the regimen for the enclosing *MedicalAction* on the *Individual*. REQUIRED.

RegimenStatus

Table 16: Definition of the `RegimenStatus` enumeration

Name	Ordinal	Description
UNKNOWN_STATUS	0	The status of the regimen is unknown
STARTED	1	The regimen was started
COMPLETED	2	The regimen was completed
DISCONTINUED	3	The regimen was discontinued but not completed

4.30.2 Example

The following example describes twice daily dosing of 30 mg of losartan given orally.

```
therapeuticRegimen:
  externalReference:
    id: "NCT04576091"
    reference: "https://clinicaltrials.gov/ct2/show/NCT04576091"
    description: "Testing the Addition of an Anti-cancer Drug, BAY1895344, With␣
↪Radiation\
      \ Therapy to the Usual Pembrolizumab Treatment for Recurrent Head and Neck␣
↪Cancer"
  startTime:
    timestamp: "2020-03-15T13:00:00Z"
  regimenStatus: "STARTED"
```

4.30.3 Explanations

identifier

An *OntologyClass* or *ExternalReference* representing the therapeutic regimen which the *subject* (*Individual*) has followed.

start_time

When the regimen was started, as represented by a *TimeElement*.

end_time

When the regimen ended, as represented by a *TimeElement*.

regimen_status

The status of the regimen - whether it has started, completed or was discontinued. Regimens which were discontinued are RECOMMENDED to record any adverse events (*MedicalAction*.adverse_events) and the reason for termination (*MedicalAction*.treatment_termination_reason) in the enclosing *MedicalAction* message.

4.31 TimeElement

This element intends to bundle all of the various ways of denoting time or age in phenopackets schema. Starting with version 2, other elements will be required to use a TimeElement rather than any of the more specific elements. For instance, the version 1.0 of *PhenotypicFeature* uses an *OntologyClass* for the age of onset of the phenotypic feature. Version 2 will replace this with a *TimeElement*. This will mean that all references to time and age throughout the phenopacket standard are uniform. That this change was needed became obvious when trying to model an acute phenotypic abnormality such as an episode of fever occurring one day before admission to the hospital.

4.31.1 Data model

Table 17: Definition of the `TimeElement` element

Field	Type	Multiplicity	Description
gestational_age	*GestationalAge*	(one of the options)	measure of the age of a pregnancy
age	*Age*	(one of the options)	represents age as a ISO8601 duration (e.g., P40Y10M05D).
age_range	*AgeRange*	(one of the options)	indicates that the individual's age lies within a given range
ontology_class	*OntologyClass*	(one of the options)	indicates the age of the individual as an ontology class
timestamp	*Timestamp*	(one of the options)	indicates a specific time
interval	*TimeInterval*	(one of the options)	indicates an interval of time

4.31.2 Example

The following shows a TimeElement with the *Age* option.

```
timeElement:
    age:
        iso8601duration: "P25Y"
```

4.31.3 Explanations

gestational_age

A measure of the age of a pregnancy. Gestation, defined as the time between conception and birth, is measured in weeks and days from the first day of the last menstrual period. See *GestationalAge*.

age

This element can be used to represent age as a ISO8601 duration (e.g., P40Y10M05D). See *Age*.

age_range

This element can be used indicates that the individual's age lies within a given range, which may be desirable to help preserve privacy. See *AgeRange*

ontology_class

If an *OntologyClass* is used to represent the age of onset of a phenotypic feature, then terms for age of onset can be chosen from the Onset subhierarchy of the HPO. See *OntologyClass*.

timestamp

A *Timestamp* can be used to represent a specific time. Note that all timestamps in a phenopacket can be shifted by the same amount to help preserve privacy if desired.

interval

This element can be used to represent a specific interval of time. See *TimeInterval*.

4.32 TimeInterval

An time interval is meant to denote an interval of time whose begin and end is defined by *Timestamp*.

4.32.1 Data model

Table 18: Definition of the `TimeInterval` element

Field	Type	Multiplicity	Description
start	*Timestamp*	1..1	begin of interval. REQUIRED.
end	*Timestamp*	1..1	end of interval. REQUIRED.

4.32.2 Example

The following message could be used to represent the interval from March 15, 2020, 1PM to March 25, 2020, 9PM.

```
timeInterval:
  start: "2020-03-15T13:00:00Z"
  end: "2020-03-25T09:00:00Z"
```

4.32.3 Explanations

start

The date and time of the start of the interval.

end

The date and time of the end of the interval.

4.32.4 Privacy concerns

In some cases it may be desirable to shift all specific dates in a phenopacket by the same random amount. For instance, we might shift all dates by 2 years. In this case the above interval element would be represented as follows

```
timeInterval:
  start: "2018-03-15T13:00:00Z"
  end: "2018-03-25T09:00:00Z"
```

4.33 Timestamp

In phenopackets we define the *Timestamp* as an ISO-8601 date time string.

The following documentation paraphrases the description of how this is represented in protobuf as JSON Timestamp

The format for this is "{year}-{month}-{day}T{hour}:{min}:{sec}[.{frac_sec}]Z" where {year} is always expressed using four digits while {month}, {day}, {hour}, {min}, and {sec} are zero-padded to two digits each. The fractional seconds, which can go up to 9 digits (i.e. up to 1 nanosecond resolution), are optional. The "Z" suffix indicates the timezone ("UTC"); the timezone is required.

For example, "2021-06-02T16:52:15.01Z" encodes 15.01 seconds past 16:52 UTC on June 2, 2021.

In JavaScript, one can convert a Date object to this format using the standard toISOString() method.

In Python, a standard *datetime.datetime* object can be converted to this format using strftime with the time format spec '%Y-%m-%dT%H:%M:%S.%fZ'.

Likewise, in Java, one can use the DateTimeFormatter.ISO_DATE_TIME to obtain a formatter capable of generating timestamps in this format.

4.34 Treatment

This represents treatment with an agent such as a drug (pharmaceutical agent), broadly defined as prescription and over-the-counter medicines, vaccines, and large-molecule biologic therapies.

4.34.1 Data model

Table 19: Definition of the `Treatment` element

Field	Type	Multiplicity	Description
agent	*OntologyClass*	1..1	The drug or therapeutic agent. REQUIRED.
route_of_administration	*OntologyClass*	0..1	How was the drug administered. RECOMMENDED.
dose_intervals	*DoseInterval* (List)	0..*	dosages. RECOMMENDED.
drug_type	*DrugType*	0..1	Context of the drug administration
cumulative_dose	*Quantity*	0..1	The cumulative dose administered during the period of the treatment.

4.34.2 Example

The following example describes twice daily dosing of 30 mg of losartan given orally.

```
treatment:
  agent:
    id: "DrugCentral:1610"
    label: "losartan"
  routeOfAdministration:
    id: "NCIT:C38288"
    label: "Oral Route of Administration"
  doseIntervals:
  - quantity:
      unit:
        id: "UO:0000022"
        label: "milligram"
      value: 30.0
    scheduleFrequency:
      id: "NCIT:C64496"
      label: "Twice Daily"
    interval:
      start: "2020-03-15T13:00:00Z"
      end: "2020-03-25T09:00:00Z"
  drugType: "PRESCRIPTION"
```

The following example specifies that aclarubicin (a type of anthracycline) was given intravenously every three weeks in the time period from 2020-07-10 to 2020-08-10, as part of a cancer chemotherapy treatment for a cumulative dose of 200 mg/kg.

```
treatment:
    treatment:
        agent:
            id: "DrugCentral:80"
            label: "aclarubicin"
        routeOfAdministration:
            id: "NCIT:C38276"
            label: "Intravenous Route of Administration"
        doseIntervals:
            - quantity:
                unit:
                    id: "NCIT:C124458"
                    label: "Milligram per Kilogram per Dose"
                value: 100.0
            scheduleFrequency:
                id: "NCIT:C64535"
                label: "Every Three Weeks"
            interval:
                start: "2020-07-10T00:00:00Z"
                end: "2020-08-10T00:00:00Z"
        drugType: "EHR_MEDICATION_LIST"
    cumulativeDose:
        unit:
            id: "EFO:0002902"
            label: "milligram per kilogram"
        value: 200.0
```

This example represents treatment with tamoxifen, 20 mg a day by mouth, administered over a time period of 5 years from 2015 to 2020 with a total cumulative dose of 36500 mg.

```
treatment:
    agent:
        id: "DrugCentral:2561"
        label: "tamoxifen"
    routeOfAdministration:
        id: "NCIT:C38288"
        label: "Oral Route of Administration"
    doseIntervals:
        - quantity:
            unit:
                id: "NCIT:C28253"
                label: "Milligram"
            value: 20.0
    scheduleFrequency:
        id: "NCIT:C125004"
        label: "Once Daily"
    interval:
        start: "2020-03-15T13:00:00Z"
        end: "2020-03-25T09:00:00Z"
    drugType: "PRESCRIPTION"
cumulativeDose:
    unit:
        id: "NCIT:C28253"
        label: "Milligram"
    value: 36500.0
```

4.34.3 Explanations

agent

An ontology term representing the therapeutic agent. This can be a term from DrugCentral, RxNorm, Drugbank, ChEBI, or other ontologies.

route_of_administration

How the drug is administered, e.g., by mouth or intravenously. This can be specified by ontology terms from the NCIT subhierarchy for Route of Administration.

dose_intervals

block of time in which the dosage of a medication was constant, e.g., 30 mg/day for an interval of 10 days. See *DoseInterval*.

drug_type

The context in which a drug was administered. See *DrugType*.

cumulative_dose

The cumulative dose, defined as the total dose from repeated exposures to chemotherapy, monitoring of which is an important part of treatment with chemotherapy. For instance, cardiac side effect risk increases with greater cumulative doses of antrhacycline.

4.35 Update

A class for holding data about an update event to a record. This is used to hold brief details about when it occurred and optionally who or what made the update and any pertinent information regarding the content and/or reason for the update. The class is used in the *MetaData* element.

4.35.1 Data model

Table 20: Definition of the `Update` element

Field	Type	Multiplicity	Description
timestamp	ISO8601 UTC timestamp	1..1	ISO8601 UTC timestamp at which this record was updated. REQUIRED.
updated_by	string	0..1	Information about the person/organisation/network that has updated the phenopacket.
comment	string	0..1	Textual comment about the changes made to the content and/or reason for the update.

4.35.2 Example

```
update:
  timestamp: "2018-06-10T10:59:06Z"
```

Optionally, with extra information:

```
update:
  timestamp: "2018-06-10T10:59:06Z"
  updatedBy: "Julius J."
  comment: "added phenotypic features to individual patient:1"
```

4.35.3 Explanations

timestamp

An ISO8601 UTC timestamp for when this phenopacket was updated.

updated_by

Information about the person/organisation/network that has updated the phenopacket.

comment

Textual comment about the changes made to the content and/or reason for the update. This should be a brief description only, it is not the actual update.

4.36 Value

The value element is meant to be used as part of the *Measurement* element, and it represents the outcome of a measurment.

4.36.1 Data model

Table 21: Definition of the `Value` element

Field	Type	Multiplicity	Description
value	{*Quantity* | *OntologyClass*}	1..1	the outcome (value) of a measurement. REQUIRED.

4.36.2 Examples

The following example shows a Value used for a quantitative measurement.

```
value:
    quantity:
        unit:
            id: "UO:0000316"
            label: "cells per microliter"
        value: 24000.0
        referenceRange:
            unit:
                id: "UO:0000316"
                label: "cells per microliter"
            low: 150000.0
            high: 450000.0
```

The following example shows a Value used for an ordinal measurement.

```
value:
    ontologyClass:
        id: "NCIT:C25626"
        label: "Present"
```

4.36.3 Explanations

See *Quantity* for explanations of how to contruct that element. For ordinal measurements, the following terms may be useful.

Term	ID
Present	NCIT:C25626
Absent	NCIT:C48190
Abnormal	NCIT:C25401
Elevated	NCIT:C25493
Reduced	NCIT:C25640

4.37 VariationDescriptor

Variation Descriptors are part of the VRSATILE framework, a set of conventions extending the GA4GH Variation Representation Specification (VRS). Descriptors allow for the complemetary use of human-readable labels, descriptions, alternate contexts, and identifier cross-references alongside the precise computable representation of variation concepts provided by VRS.

Consequently, many forms and formats of variation can be used in variation descriptors, including HGVS descriptions, VCF Records, and SPDI alleles. We recommend the use of VRS Variation objects for representing variants when possible.

The Variation Descriptor should be used to describe candidate variants or diagnosed causative variants. The `VariationDescriptor` element itself is an element of a *VariantInterpretation*. If it is present, the Phenopacket standard has the following requirements.

A variation can refer to an external source, for example the ClinGen allele registry, ClinVar, dbSNP, dbVAR etc. using the `id` field. It is RECOMMENDED to use a *CURIE* identifier and corresponding *Resource*. When indicating multiple alternate ids for a variation, use the `alternate_ids` field.

Multiple alleles *in-cis* can be modeled as a VRS Haplotype.

The zygosity of the variant as determined in all of the samples represented in this Phenopacket is represented using a list of terms taken from the Genotype Ontology (GENO). For instance, if a variant affects one of two alleles at a certain locus, we could record the zygosity using the term heterozygous (GENO:0000135). This value is stored in the Variation Descriptor `alleleic_state` field.

4.37.1 Data model

Field	Type	Multiplicity	Description
id	string	1..1	Descriptor ID; MUST be unique within document. REQUIRED.
variation	*Variation*	0..1	The VRS `Variation` object
label	string	0..1	A primary label for the variation
description	string	0..1	A free-text description of the variation
gene_context	*GeneDescriptor*	0..1	A specific gene context that applies to this variant
expressions	*Expression*	0..*	HGVS, SPDI, and gnomAD-style strings should be represented as Expressions
vcf_record	*VcfRecord*	0..1	A VCF Record of the variant. This SHOULD be a single allele, the VCF genotype (GT) field should be represented in the allelic_state
xrefs	string	0..*	List of CURIEs representing associated concepts. Allele registry, ClinVar, or other related IDs should be included as xrefs
alternate_labels	string	0..*	Common aliases for a variant, e.g. EGFR vIII, are alternate labels
extensions	*Extension*	0..*	List of resource-specific Extensions needed to describe the variation
molecule_context	*MoleculeContext*	1..1	The molecular context of the vrs variation.
structural_type	*OntologyClass*	0..1	The structural variant type associated with this variant, such as a substitution, deletion, or fusion. We RECOMMEND using a descendent term of SO:0001537.
vrs_ref_allele_seq	string	0..1	A Sequence corresponding to a "ref allele", describing the sequence expected at a SequenceLocation reference.
allelic_state	*OntologyClass*	0..1	See *allelic_state* below. RECOMMENDED.

Variation

VRS is a GA4GH standard which provides a computable representation of variation, be it a genomic, transcript or protein variation. VRS also provides mechanisms for representing haplotypes and systemic variation such as Copy Number Variants (CNVs).

VcfRecord

This element is used to describe variants using the Variant Call Format, which is in near universal use for exome, genome, and other Next-Generation-Sequencing-based variant calling. It is an appropriate option to use for variants reported according to their chromosomal location as derived from a VCF file.

In the Phenopacket format, it is expected that one `VcfRecord` message described a single allele (in contrast to the actual VCF format that allows multiple alleles at the same position to be reported on the same line; to report these in Phenopacket format, two `VariantDescriptor` messages would be required). In general the `VcfRecord` should be used only for the purposes of reporting variants of specific interest, such as in the *VariantInterpretation*, for cases requiring larger numbers of variants in VCF format, the *File* should be used.

For structural variation the INFO field should contain the relevant information . In general, the `info` field should only be used to report structural variants and it is not expected that the Phenopacket will report the contents of the info field for single nucleotide and other small variants.

Field	Type	Multiplicity	Description
genome_assembly	string	1..1	Identifier for the genome assembly used to call the allele. REQUIRED.
chrom	string	1..1	Chromosome or contig identifier. REQUIRED.
pos	int	1..1	The reference position, with the 1st base having position 1. REQUIRED.
id	string	0..1	Identifier: Semicolon-separated list of unique identifiers where available. If this is a dbSNP variant thers number(s) should be used.
ref	string	1..1	Reference base. REQUIRED.
alt	string	1..1	Alternate base. REQUIRED.
qual	string	0..1	Quality: Phred-scaled quality score for the assertion made in ALT.
filter	string	0..1	Filter status: PASS if this position has passed all filters.
info	string	0..1	Additional information: Semicolon-separated series of additional information fields

Extension

The Extension class provides a means to extend descriptions with other attributes unique to a content provider. These extensions are not expected to be natively understood by all users, but may be used for pre-negotiated exchange of message attributes when needed.

Field	Type	Multiplicity	Description
name	string	1..1	A name for the Extension. REQUIRED.
value	google.protobuf.Any	1..*	Any primitive or structured object. REQUIRED.

Expression

The Expression class is designed to enable descriptions based on a specified nomenclature or syntax for representing an object. Common examples of expressions for the description of molecular variation include the HGVS and ISCN nomenclatures.

We RECOMMEND the use one of the following values in the `syntax` field: `hgvs`, `iscn`, `spdi`

Field	Type	Multiplicity	Description
syntax	string	1..1	A name for the expression syntax. REQUIRED.
value	string	1..1	The concept expression as a string. REQUIRED.
version	string	0..1	An optional version of the expression syntax.

MoleculeContext

The molecular context of the variant. Default is `unspecified_molecule_context`.

4.37.2 Examples

In these examples we will show how the ClinVar allele 13294 can be represented using a `VariationDescriptor`. While it is possible to combine all these in a single message, we have separated them for clarity.

VRS

Here we're representing the genomic variation using VRS, however VRS is capable of representing the variation in genomic, transcript or protein coordinates.

Example

```
variationDescriptor:
  id: "clinvar:13294"
  variation:
    allele:
      sequenceLocation:
        sequenceId: "NC_000010.11"
        sequenceInterval:
          startNumber:
            value: "121496700"
          endNumber:
            value: "121496701"
      literalSequenceExpression:
        sequence: "G"
  moleculeContext: "genomic"
  vrsRefAlleleSeq: "T"
  allelicState:
    id: "GENO:0000135"
    label: "heterozygous"
```

HGVS

Variants can be represented using the HGVS nomenclature as follows.

For example, the Human Genome Variation Society (HGVS) expression `NM_000226.3:c.470T>G` indicates that a T at position 470 of the sequence represented by version 3 of NM_000226 (which is the mRNA of the human keratin 9 gene KRT9).

We recommend using a tool such as VariantValidator or Mutalyzer to validate the HGVS string. See the HGVS recommendations for details about the HGVS nomenclature.

Example

```
variationDescriptor:
  id: "clinvar:13294"
  expressions:
  - syntax: "hgvs"
    value: "NM_000226.3:c.470T>G"
  allelicState:
    id: "GENO:0000135"
    label: "heterozygous"
```

VCF

Example

```
variationDescriptor:
    id: "clinvar:13294"
    vcfRecord:
        genomeAssembly: "GRCh38"
        chrom: "10"
        pos: 121496701
        id: "rs121918506"
        ref: "T"
        alt: "G"
        qual: "."
        filter: "."
        info: "."
    zygosity:
        id: "GENO:0000135"
        label: "heterozygous"
```

SPDI

The Sequence Position Deletion Insertion (SPDI) notation is a relatively new notation which uses the same normalisation protocol as VRS. We recommend that users familiarize themselves with this relatively new notation, which differs in important ways from other standards such as VCF and HGVS.

Tools for interconversion between SPDI, HGVS and VCF exist at the NCBI.

SPDI stands for

1. S = SequenceId
2. P = Position , a 0-based coordinate for where the Deleted Sequence starts
3. D = DeletedSequence , sequence for the deletion, can be empty
4. I = InsertedSequence , sequence for the insertion, can be empty

For instance, `Seq1:4:A:G` refers to a single nucleotide variant at the fifth nucleotide (nucleotide 4 according to zero-based numbering) from an `A` to a `G`. See the SPDI webpage for more examples.

The SPDI notation represents variation as deletion of a sequence (D) at a given position (P) in reference sequence (S) followed by insertion of a replacement sequence (I) at that same position. Position 0 indicates a deletion that starts immediately before the first nucleotide, and position 1 represents a deletion interval that starts between the first and second residues, and so on. Either the deleted or the inserted interval can be empty, resulting in a pure insertion or deletion.

Note that the deleted and inserted sequences in SPDI are all written on the positive strand for two-stranded molecules.

Example

```
variationDescriptor:
  id: "clinvar:13294"
  expressions:
  - syntax: "spdi"
    value: "NC_000010.11:121496700:T:G"
  allelicState:
    id: "GENO:0000135"
    label: "heterozygous"
```

ISCN

The International System for Human Cytogenetic Nomenclature (ISCN), an international standard for human chromosome nomenclature, which includes band names, symbols and abbreviated terms used in the description of human chromosome and chromosome abnormalities.

For example del(6)(q23q24) describes a deletion from band q23 to q24 on chromosome 6.

Example

```
variationDescriptor:
  id: "id:A"
  expressions:
  - syntax: "iscn"
    value: "t(8;9;11)(q12;p24;p12)"
```

4.37.3 allelic_state

The zygosity of the variant as determined in all of the samples represented in this Phenopacket is represented using a list of terms taken from the Genotype Ontology (GENO). For instance, if a variant affects one of two alleles at a certain locus, we could record the zygosity using the term heterozygous (GENO:0000135).

4.38 VariantInterpretation

This element represents the interpretation of a variant according to the American College of Medical Genetics (ACMG) variant interpretation guidelines.

4.38.1 Data model

VariantInterpretation

Table 22: Definition of the `VariantInterpretation` element

Field	Type	Multiplicity	Description
acmg_pathogenicity_classification	*AcmgPathogenicityClassification*	1..1	one of the five ACMG pathogenicity categories, or NOT_PROVIDED. The default is NOT_PROVIDED
therapeutic_actionability	*TherapeuticActionability*	1..1	The therapeutic actionability of the variant, default is UNKNOWN_ACTIONABILITY
variant	*VariationDescriptor*	1..1	a genetic/genomic variant

AcmgPathogenicityClassification

Table 23: Definition of the `AcmgPathogenicityClassification` enumeration

Name	Ordinal	Description	
NOT_PROVIDED	0	The variant has not been subject to classification	
BENIGN	1	This variant does not cause disease	
LIKELY_BENIGN	2	This variant is not expected to have a major effect on disease. However	the scientific evidence is currently insufficient to prove this conclusively
UNCERTAIN_SIGNIFICANCE	3	There is not enough information at this time to support a more definitive classification of this variant	
LIKELY_PATHOGENIC	4	There is a high likelihood (greater than 90% certainty) that this variant is disease-causing	
PATHOGENIC	5	This variant directly contributes to the development of disease	

TherapeuticActionability

Table 24: Definition of the `TherapeuticActionability` enumeration

Name	Ordinal	Description
UNKNOWN_ACTIONABILITY	0	There is not enough information at this time to support any therapeutic actionability for this variant
NOT_ACTIONABLE	1	This variant has no therapeutic actionability.
ACTIONABLE	2	This variant is known to be therapeutically actionable.

4.38.2 Example

The following element shows how to denote an interpretation of a variant as pathogenic.

```
variantInterpretation:
  acmgPathogenicityClassification: "PATHOGENIC"
  variationDescriptor:
    expressions:
    - syntax: "hgvs"
      value: "NM_001848.2:c.877G>A"
    allelicState:
      id: "GENO:0000135"
      label: "heterozygous"
```

4.38.3 Explanations

acmg_pathogenicity_classification

The ACMG has recommended a five-tier classification system (Richards et al., 2015).

- Benign (BENIGN): This variant does not cause disease.
- Likely benign (LIKELY_BENIGN): This variant is not expected to have a major effect on disease; however, the scientific evidence is currently insufficient to prove this conclusively.
- Uncertain significance (UNCERTAIN_SIGNIFICANCE): There is not enough information at this time to support a more definitive classification of this variant.
- Likely pathogenic (LIKELY_PATHOGENIC): There is a high likelihood (greater than 90% certainty) that this variant is disease-causing.
- Pathogenic (PATHOGENIC): This variant directly contributes to the development of disease.

In the case that the variant has not been subject to classification, the value 'NOT_PROVIDED' MUST be used.

therapeutic_actionability

An enumeration flagging the variant as being a candidate for treatment/ clinical intervention of the disorder caused by this variant, which could improve the clinical outcome.

variation_descriptor

The subject of the variant interpretation. See *VariationDescriptor* for more information.

4.39 VitalStatus

This element can be used to report whether the individual is living or dead at the timepoint when the phenopacket was created (or if the status is unknown).

Data model

Table 25: Definition of the `VitalStatus` element

Field	Type	Multiplicity	Description
status	*Status*	1..1	one of UNKNOWN_STATUS, ALIVE, DECEASED. REQUIRED.
time_of_death	*TimeElement*	0..1	Should be left blank if patient not known to be deceased
cause_of_death	*OntologyClass*	0..1	Should be left blank if patient not known to be deceased
survival_time_in_days	integer	0..1	Number of days the patient was alive after their primary diagnosis

The vital status is commonly collected in cohort studies on cancer.

The following shows how the element can be used to report the time and cause of death.

4.39.1 Status

Name	Ordinal	Description
UNKNOWN_STATUS	0	Not assessed or not available.
ALIVE	1	Alive. Maps to NCIT:C37987
DECEASED	2	Dead. Maps to NCIT:C28554

```
vitalStatus:
    status: "DECEASED"
    timeOfDeath:
        timestamp: "2015-10-01T10:54:20.021Z"
    causeOfDeath:
        id: "NCIT:C36263"
        label: "Metastatic Malignant Neoplasm"
```

The following element should be used to report the individual is alive.

```
vitalStatus:
    status: "ALIVE"
```

In practice, this element is useful in cohort studies in which the association of some treatment or genetic variation is compared with mortality. For many other applications, it may not be necessary to use a VitalStatus element.

CHAPTER 5

Recommended Ontologies

The phenopacket schema can be used with any ontologies. The phenopacket can be compared to a hierarchical structure with "slots" for ontology terms and other data. Different use cases may require different ontology terms to cover the subject matter or to fulfil requirements of a specific research project. The spectrum of requirements is so broad that we do not think it is appropriate to require a specific set of ontologies for use with phenopackets. Nonetheless, the value of phenopacket-encoded data will be greatly increased if the community of users converges towards a common set of ontologies (to the extent possible). Here, we provide general recommendations for ontologies that we have found to be useful. This list is incomplete and we would welcome feedback from the community about ontologies that should be added to this page.

We do anticipate that individual research consortia or other groups should agree on a set of allowed ontologies for specific projects.

5.1 Diseases

Mondo Disease Ontology provides a comprehensive logically structured ontology of diseases that integrates multiple other disease ontologies.

Table 1: Example terms from Mondo

Label	Id
incontinentia pigmenti	MONDO:0010631
dilated cardiomyopathy 3B	MONDO:0010542

Other disease ontologies of note include The National Cancer Institute's thesaurus (NCIT), Orphanet Rare Disease Ontology (ORDO), Disease Ontology (DO), and the Online Mendelian Inheritance in Man (OMIM).

5.2 Phenotypic features

The Human Phenotype Ontology (HPO) provides a comprehensive logical standard to describe and computationally analyze phenotypic abnormalities found in human disease.

Table 2: Example terms from HPO

Label	Id
Arachnodactyly	HP:0001166
Patent ductus arteriosus	HP:0001643

5.3 Anatomy

UBERON is an integrated cross-species ontology with classes representing a variety of anatomical entities.

Table 3: Example terms from UBERON

Label	Id
heart	UBERON:0000948
brain	UBERON:0000955

5.4 Genes

The HUGO Gene Nomenclature Committee (HGNC) provides standard names, symbols, and IDs for human genes.

Table 4: Example terms from HGNC

Label	Id
FBN1	HGNC:3603
NF1	HGNC:7765

Other sources of gene IDs include NCBI Gene and Ensembl.

5.5 Units of Measurement

The Units of measurement ontology (denoted UO) provides terms for units commonly encountered in medical data. The following table shows some typical examples.

Table 5: Example terms from Units of measurement ontology

Label	Id
millimolar	UO:0000063
milligram	UO:0000022
millimetres of mercury	UO:0000272

5.6 Genotypes

GENO is anontology of genotypes their more fundamental sequence components, and links to related biological and experimental entities. We use GENO terms to denote genotypes.

Table 6: Example terms from GENO

Label	Id
heteroyzgous	GENO:0000135
homozygous	GENO:0000136

5.7 Assays

Logical Observation Identifiers Names and Codes (LOINC) is a database and universal standard for identifying medical laboratory observations. It can be used to denote clinical assays in the *Measurement* element.

Table 7: Example terms from LOINC

Label	Id
Platelets [#/volume] in Blood	LOINC:26515-7
Calcium [Mass/volume] in Serum or Plasma	LOINC:17861-6 <https://www

5.8 Medications

DrugCentral integrates a broad spectrum of drug resources related to chemical structures, biological activities, regulatory data, pharmacology and drug formulations

Table 8: Example terms from DrugCentral

Label	Id
losartan	DrugCentral:1610
selumetinib	DrugCentral:5388

Other ontologies with coverage of drugs include ChEBI, RxNorm, and DrugBank.

5.9 The National Cancer Institute's Thesaurus

The National Cancer Institute's thesaurus (NCIT) provides a wide range of terms that can be useful for phenopackets. In addition to providing an ontology of cancers, NCIT provides terms for procedures, findings, units or measurement, scheduling, etc. The following table shows an an example pf the subhierarchy for Unit of Measure (NCIT:C25709). and for Schedule Frequency (NCIT:C64493).

Table 9: Example terms from NCIT Unit of Measure and Schedule Frequency subhierarchies

Label	Id
Milligram per Kilogram per Dose	NCIT:C124458
Twice Daily	NCIT:C64496
Cells per Milliliter	NCIT:C74919

5.10 Experimental Factor Ontology

Experimental factor ontology (EFO) is an ontology of experimental variables particularly those used in molecular biology. EFO imports terms from many source ontologies and provides additional terms needed to provide a systematic description of many experimental variables available in EBI databases.

Table 10: Example terms from EFO

Label	Id
abnormal sample	EFO:0009655
genomic DNA	EFO:0008479
milligram per kilogram	EFO:0002902

CHAPTER 6

Working with Phenopackets

The phenopacket schema has been introduced in *Phenopacket Schema* and can be considered the source of truth for the specification. While it is possible to inter-operate with other services using JSON produced from hand-crafted/alternative implementations, we **strongly** suggest using the schema to compile any required language implementations.

6.1 Example code

We provide several examples that demonstrate how to work with Phenopackets in Java and C++. There are also Python examples in the source code test directory. All three langauge implementations are automatically produced as part of the build (*Java Build*).

6.1.1 Working with Phenopackets in Java

Here we provide some guidance on how to work with Phenopackets in Java. The sections on *Java Build*, *Exporting and Importing Phenopackets* provide general guidance about using Java to work with phenopackets. The following sections provide a few examples of how to build various elements of Phenopackets. Finally, we present the full Java code used to build each of the examples for *rare disease* and *cancer*.

Java Build

Most users of phenopacket-schema in Java should use maven central to include the phenopacket-schema package.

Setting up the Java build

To include the phenopacket-schema package from maven central, add the following to the pom file

Define the phenopackets.version in the properties section of the pom.xml file.

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  ...
  <phenopackets.version>1.0.0</phenopackets.version>
</properties>
```

Then put the following stanza into the `dependencies` section of the maven pom.xml file.

```
<dependency>
  <groupId>org.phenopackets</groupId>
  <artifactId>phenopacket-schema</artifactId>
  <version>${phenopackets.version}</version>
</dependency>
```

Building phenopacket-schema locally

Users can also download phenopacket-schema from its GitHub repository and install it locally.

```
$ git clone https://github.com/phenopackets/phenopacket-schema
$ cd phenopacket-schema
$ mvn compile
$ mvn install
```

Exporting and Importing Phenopackets

It is easy to export Phenopackets in JSON, YAML, or protobuf format. Bear in mind that protobuf was designed as a wire-format allowing for 'schema evolution' so this is safest to use in this environment. It would be advisable to store your data in a datastore with a schema relevant to your requirements and be able to map that to the relevant Phenopacket message types for exchange with your users/partners. If you don't it is possible that breaking changes to the schema will mean you cannot exchange data with parties using a later version of the schema or if you update the schema your tools are using they will no longer be able to read your data written using the previous version. While protobuf allows for 'schema evolution' by design which will limit the impact of changes to the schema precipitating this scenario, it is nonetheless a possibility which the paranoid might wish to entertain.

JSON export

In many situations it may be desirable to export the Phenopacket as JSON. This is easy with the following commands (we show how to create a Phenopacket in Java elsewhere).

```
import org.phenopackets.schema.v1.Phenopacket;
import com.google.protobuf.util.JsonFormat;
import java.io.IOException;

Phenopacket phenoPacket = // create a Phenopacket
try {
    String jsonString = JsonFormat.printer().includingDefaultValueFields().print(pp);
    System.out.println(jsonString);
 } catch (IOException e) {
   e.printStackTrace();
 }
```

YAML export

YAML (YAML Ain't Markup Language) is a human friendly data serialization standard for all programming languages.

```
import org.phenopackets.schema.v1.Phenopacket;
import com.google.protobuf.util.JsonFormat;
import java.io.IOException;
import com.fasterxml.jackson.dataformat.yaml.YAMLMapper;

Phenopacket phenoPacket = // create a Phenopacket
try {
    String jsonString = JsonFormat.printer().includingDefaultValueFields().
↪print(phenoPacket);
    JsonNode jsonNodeTree = new ObjectMapper().readTree(jsonString);
    String yamlString = new YAMLMapper().writeValueAsString(jsonNodeTree);
    System.out.println(yamlString);
} catch (IOException e) {
    e.printStackTrace(); // or handle the Exception as appropriate
}
```

Protobuf export

For most use case, we recommend using JSON as the serialization format for Phenopackets. Protobuf is more space efficient than JSON but it is a binary format that is not human readable.

```
import org.phenopackets.schema.v1.Phenopacket;
import java.io.IOException;

Phenopacket phenoPacket = // create a Phenopacket
try {
    phenoPacket.writeTo(System.out);
 } catch (IOException e) {
    e.printStackTrace(); // or handle the Exception as appropriate
}
```

We can write to any OutputStream (replace System.out in the above code), e.g. a file or network.

Importing Phenopackets (JSON format)

There are multiple ways of doing this with different JSON libraries e.g. Jackson, Gson, JSON.simple.... The following code explains how to convert the JSON String object into a protobuf class. This isn't limited to a Phenopacket message, so long as you know the type of message contained in the json, you can merge it into the correct Java representation.

```
String phenopacketJsonString = // Phenopacket in JSON as a String;
try {
    Phenopacket.Builder phenoPacketBuilder = Phenopacket.newBuilder();
    JsonFormat.parser().merge(jsonString, phenoPacketBuilder);
    Phenopacket phenopacket = phenoPacketBuilder.build();
    // do something with phenopacket ...
} catch (IOException e1) {
    e1.printStackTrace(); // or handle the Exception as appropriate
}
```

Evidence (Java)

The evidence code is used to document the support for an assertion. Here, we will show an example for the assertion that flexion contractures are found in stiff skin syndrome.

```
import org.phenopackets.schema.v1.core.Evidence;
import org.phenopackets.schema.v1.core.ExternalReference;
import org.phenopackets.schema.v1.core.OntologyClass;

OntologyClass publishedClinicalStudy = OntologyClass.
            newBuilder().
            setId("ECO:0006017").
            setLabel("author statement from published clinical study used in manual␣
↪assertion").
            build();
    ExternalReference reference = ExternalReference.newBuilder().
            setId("PMID:20375004").
            setDescription("Mutations in fibrillin-1 cause congenital scleroderma:␣
↪stiff skin syndrome").
            build();
    Evidence evidence = Evidence.newBuilder().
            setEvidenceCode(publishedClinicalStudy).
            setReference(reference).
            build();
```

This code produces the following Evidence element.

```
{
    evidence_code {
        id: "ECO:0006017"
        label: "author statement from published clinical study used in manual␣
↪assertion"
    }
    reference {
        id: "PMID:20375004"
        description: "Mutations in fibrillin-1 cause congenital scleroderma: stiff␣
↪skin syndrome"
    }
}
```

Timestamp (Java)

A Timestamp represents a point in time independent of any time zone or local calendar, encoded as a count of seconds and fractions of seconds at nanosecond resolution. The count is relative to an epoch at UTC midnight on January 1, 1970, in the proleptic Gregorian calendar which extends the Gregorian calendar backwards to year one (see Unix time).

A timestamp is required for several elements of the Phenopacket including the *MetaData*. Usually, code will create a timestamp to represent the current time (the time at which the Phenopacket is being created).

```
import com.google.protobuf.Timestamp;

long millis  = System.currentTimeMillis();
Timestamp timestamp = Timestamp.newBuilder().setSeconds(millis / 1000)
            .setNanos((int) ((millis % 1000) * 1000000)).build();
```

It is also possible to create a timestamp for an arbitrary date. For instance, the following code creates a timepoint for an important date in English history.

```
import com.google.protobuf.Timestamp;
import java.time.Instant;
import java.time.format.DateTimeFormatter;
import java.time.temporal.TemporalAccessor;

String hastings = "1066-10-14T00:00:00.001Z";
TemporalAccessor date = DateTimeFormatter.ISO_DATE_TIME.parse(hastings);
System.out.println(DateTimeFormatter.ISO_DATE_TIME.format(date));
Instant instant = Instant.from(date);
Timestamp timestamp = Timestamp.newBuilder()
        .setSeconds(instant.getEpochSecond())
        .setNanos(instant.getNano())
        .build();
System.out.println(JsonFormat.printer().print(timestamp));
```

Duration (Java)

The *Age* messages use ISO8601 duration strings. These can be easily converted to Java types using the Period class.

```
import java.time.Period;

Subject subject = phenopacket.getSubject();
if (subject.hasAgeAtCollection()) {
    // Phenopacket Age
    Age ageAtCollection = subject.getAgeAtCollection();
    // Java Period
    Period agePeriod = Period.parse(ageAtCollection.getAge());
}
```

The Java code

We show some Java code that demonstrates the basic methodology for building a Phenopacket. We have put the entire code into one function for didactic purposes, but real-life code might be more structured. We do define one auxialliary function

```
/** convenience function to help creating OntologyClass objects. */
 public static OntologyClass ontologyClass(String id, String label) {
     return OntologyClass.newBuilder()
             .setId(id)
             .setLabel(label)
             .build();
 }
```

With this, we present a function that creates a Phenopacket that represents the case report described above

```
public Phenopacket spherocytosisExample() {
      final String PROBAND_ID = "PROBAND#1";
      PhenotypicFeature spherocytosis = PhenotypicFeature.newBuilder()
              .setType(ontologyClass("HP:0004444", "Spherocytosis"))
              .setClassOfOnset(ontologyClass("HP:0011463", "Childhood onset"))
              .build();
      PhenotypicFeature jaundice = PhenotypicFeature.newBuilder()
```

(continues on next page)

(continued from previous page)

```
            .setType(ontologyClass("HP:0000952", "Jaundice"))
            .setClassOfOnset(ontologyClass("HP:0011463", "Childhood onset"))
            .build();
    PhenotypicFeature splenomegaly = PhenotypicFeature.newBuilder()
            .setType(ontologyClass("HP:0001744", "Splenomegaly"))
            .setClassOfOnset(ontologyClass("HP:0011463", "Childhood onset"))
            .build();
    PhenotypicFeature notHepatomegaly = PhenotypicFeature.newBuilder()
            .setType(ontologyClass("HP:0002240", "Hepatomegaly"))
            .setExcluded(true)
            .build();
    PhenotypicFeature reticulocytosis = PhenotypicFeature.newBuilder()
            .setType(ontologyClass("HP:0001923", "Reticulocytosis"))
            .build();

    VcfAllele vcfAllele = VcfAllele.newBuilder()
            .setGenomeAssembly("GRCh37")
            .setChr("8")
            .setPos(41519441)
            .setRef("G")
            .setAlt("A")
            .build();

    Variant ANK1_variant = Variant.newBuilder()
            .setVcfAllele(vcfAllele)
            .setZygosity(ontologyClass("GENO:0000135", "heterozygous"))
            .build();

    Individual proband = Individual.newBuilder()
            .setSex(Sex.FEMALE)
            .setId(PROBAND_ID)
            .setAgeAtCollection(Age.newBuilder().setAge("P27Y3M").build())
            .build();

    MetaData metaData = MetaData.newBuilder()
            .addResources(Resource.newBuilder()
                    .setId("hp")
                    .setName("human phenotype ontology")
                    .setNamespacePrefix("HP")
                    .setIriPrefix("http://purl.obolibrary.org/obo/HP_")
                    .setUrl("http://purl.obolibrary.org/obo/hp.owl")
                    .setVersion("2018-03-08")
                    .build())
            .addResources(Resource.newBuilder()
                    .setId("geno")
                    .setName("Genotype Ontology")
                    .setNamespacePrefix("GENO")
                    .setIriPrefix("http://purl.obolibrary.org/obo/GENO_")
                    .setUrl("http://purl.obolibrary.org/obo/geno.owl")
                    .setVersion("19-03-2018")
                    .build())
            .setCreatedBy("Example clinician")
            .build();

    return Phenopacket.newBuilder()
            .setSubject(proband)
            .addPhenotypicFeatures(spherocytosis)
```

(continues on next page)

(continued from previous page)

```
                .addPhenotypicFeatures(jaundice)
                .addPhenotypicFeatures(splenomegaly)
                .addPhenotypicFeatures(notHepatomegaly)
                .addPhenotypicFeatures(reticulocytosis)
                .addAllVariants(ImmutableList.of(ANK1_variant))
                .setMetaData(metaData)
                .build();
  }
```

Writing a Phenopacket in protobuf format

Messages can be written in binary format using the native protobuf encoding. While this is useful for machine-to-machine communication due to low latency and overhead of serialisation it is not human-readable. We refer the reader to the official documentation on this topic, but will briefly give an example of writing to an `OutputStream` here.

```
Path path = Paths.get("/path/to/file");
try (OutputStream outputStream = Files.newOutputStream(path)) {
    Phenopacket phenoPacket = new PhenoPacketExample().spherocytosisExample();
    phenoPacket.writeTo(outputStream);
} catch (IOException e) {
    e.printStackTrace();
}

// read it back again
try (InputStream inputStream = Files.newInputStream(path)) {
    Phenopacket deserialised = Phenopacket.parseFrom(inputStream);
} catch (IOException e) {
    e.printStackTrace();
}
```

JSON export

In many situations it may be desirable to export the Phenopacket as JSON. This is easy with the following commands:

First add the protobuf-java-util dependency to your Maven POM.xml

```
<dependency>
    <groupId>com.google.protobuf</groupId>
    <artifactId>protobuf-java-util</artifactId>
    <version>${protobuf.version}</version>
    <scope>test</scope>
</dependency>
```

Then you can use it to print out JSON using the *JsonFormat* class.

```
Phenopacket p = spherocytosisExample();
try {
    String jsonString = JsonFormat.printer().includingDefaultValueFields().print(p);
    System.out.println(jsonString);
} catch (Exception e) {
    e.printStackTrace();
}
```

6.1.2 Working with Phenopackets in C++

Here we provide some guidance on how to work with Phenopackets in C++.

Generating the C++ files

The maven build generates Java, C++, and Python code that can be directly used in other projects. Therefore, if you have maven set up on your machine, the easiest way to generate the C++ files is

```
$ mvn compile
$ mvn package
```

This will generate four files in the following location.

```
$ ls target/generated-sources/protobuf/cpp/
    base.pb.cc          phenopackets.pb.cc
    base.pb.h           phenopackets.pb.h
```

The other option is to use Google's `protoc` tool to generate the C++ files (The tool can be obtained from the Protobuf website Install the tool using commands appropriate to your system). The following commands will generate identical files in a new directory called `gen`.

```
$ mkdir gen
$ protoc \
    --proto_path=src/main/proto/ \
    --cpp_out=gen/ \
    src/main/proto/phenopackets.proto src/main/proto/base.proto
```

The `protoc` command specifies the directory where the protobuf files are located (*–proto_path*), the location of the directory to which the corresponding C++ files are to be written, and then passes the two protobuf files.

Compiling and building Phenopackets

The phenopacket code can be compiled and built using standard tools. Here we present a small example of a C++ program that reads in a phenopacket JSON file from the command line and prints our some of the information contained in it to the shell. The classes defined by the phenopacket are located within namespace declarations that mirror the Java package names, and thus are extremly unlikely to collide with other C++ identifiers.

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

#include <google/protobuf/message.h>
#include <google/protobuf/util/json_util.h>

#include "phenopackets.pb.h"


using namespace std;

int main(int argc, char ** argv) {
    // check that user has passed a file.
    if (argc!=2) {
        cerr << "usage: ./phenopacket_demo phenopacket-file.json\n";
```

(continues on next page)

(continued from previous page)

```
        exit(EXIT_FAILURE);
    }
    string fileName=argv[1];

    GOOGLE_PROTOBUF_VERIFY_VERSION;

    stringstream sstr;
    ifstream inFile;
    inFile.open(fileName);
    if (! inFile.good()) {
        cerr << "Could not open Phenopacket file at " << fileName <<"\n";
        return EXIT_FAILURE;
    }
    sstr << inFile.rdbuf();
    string JSONstring = sstr.str();

    ::google::protobuf::util::JsonParseOptions options;
    ::org::phenopackets::schema::v1::Phenopacket phenopacket;
    ::google::protobuf::util::JsonStringToMessage(JSONstring,&phenopacket,options);
    cout << "\n::: Reading Phenopacket at: " << fileName << " ::::\n\n";
    cout << "\tsubject.id: "<<phenopacket.subject().id() << "\n";
    // print age if available
    if (phenopacket.subject().has_age_at_collection()) {
        ::org::phenopackets::schema::v1::core::Age age = phenopacket.subject().age_at_
↪collection();
        if (! age.age().empty()) {
            cout <<"\tsubject.age: " << age.age() << "\n";
        }
    cout <<"\tsubject.sex: " ;
    org::phenopackets::schema::v1::core::Sex sex = phenopacket.subject().sex();
    switch (sex) {
        case ::org::phenopackets::schema::v1::core::UNKNOWN_SEX : cout << " unknown";␣
↪break;
        case ::org::phenopackets::schema::v1::core::FEMALE : cout <<"female"; break;
        case ::org::phenopackets::schema::v1::core::MALE: cout <<"male"; break;
        case ::org::phenopackets::schema::v1::core::OTHER_SEX:
        default:
                cout <<"other"; break;
    }
    cout << "\n";
}
cout <<"\n\tPhenotypes:\n";
for (auto i = 0; i < phenopacket.phenotypes_size(); i++) {
  const ::org::phenopackets::schema::v1::core::PhenotypicFeature& phenotype =␣
↪phenopacket.phenotypes(i);
  const ::org::phenopackets::schema::v1::core::OntologyClass type = phenotype.type();
  cout << "\tid: " << type.id() << ": " << type.label() << "\n";
}
cout <<"\n";
}
```

The Makefile for this little program is as follows.

```
CXX=g++
CXXFLAGS=-Wall -g -O0 --std=c++17 -pthread
LIBS=-lprotobuf

```

(continues on next page)

(continued from previous page)

```
TARGET=phenopacket_demo
all:$(TARGET)

OBJS=phenopackets.pb.o base.pb.o

$(TARGET):main.cpp $(OBJS)
        $(CXX) $< $(OBJS) $(CXXFLAGS) ${LIBS} -o $@

%.o: %.cpp
        $(CXX) $(CXXFLAGS) -o $@ -c $<

.PHONY: clean
clean:
        rm -f $(OBJS) $(TARGET)
```

The executable can be generated by calling `make`. Running it on a simple phenopacket would lead to the following output.

```
$ ./phenopacket_demo Gebbia-1997-ZIC3.json

::: Reading Phenopacket at: Gebbia-1997-ZIC3.json ::::

        subject.id: III-1
        subject.age: 7W
        subject.sex: male
Phenotypes:
        id: HP:0002139: Arrhinencephaly
        id: HP:0001750: Single ventricle
        id: HP:0001643: Patent ductus arteriosus
        id: HP:0001746: Asplenia
        id: HP:0004971: Pulmonary artery hypoplasia
        id: HP:0001674: Complete atrioventricular canal defect
        id: HP:0001669: Transposition of the great arteries
        id: HP:0012890: Posteriorly placed anus
        id: HP:0001629: Ventricular septal defect
        id: HP:0012262: Abnormal ciliary motility
        id: HP:0004935: Pulmonary artery atresia
        id: HP:0003363: Abdominal situs inversus
```

More information about using C++ with Protobuf is available at the Protobuf website.

phenotools

A more complete C++ implementation that performs Q/C is being developed as phenotools.

6.2 Security disclaimer

A stand-alone security review has been performed on the specification itself, however these example implementations are offered as-is, and without any security guarantees. They will need an independent security review before they can be considered ready for use in security-critical applications. If you integrate this code into your application it is AT YOUR OWN RISK AND RESPONSIBILITY to arrange for an audit.

CHAPTER 7

Examples

We provide three in-depth examples of Phenopackets. Each example was generated by Java code that is available in the `src/test/org/phenopackets/schema/v2/examples`.

7.1 A complete example: Rare Disease

We will now present a phenopacket that represents a family with one individual affected by Bethlem myopathy. We present each of the sections of the Phenopacket in separate subsections for legibility. Recall that JSON data is represented as as name/value pairs that are separated by commas (we show the trailing comma on all but the last name/value pair of the Phenopacket).

We show how to create this phenopacket in Java *here*.

7.1.1 COL6A1 mutation leading to Bethlem myopathy with recurrent hematuria: a case report

We present an example that summarizes the data presented in a case report about a boy with Bethlem myopathy. For simplicity's sake, we have omitted some of the details, but it should be obvious how one would construct the full Phenopacket.

7.1.2 id

The *id* field is an arbitrary identifier of the family. In this publication, the family is not refered to by any special name because only one family is reported, but often one sees identifiers such as "family A", etc.

```
id: "family"
```

7.1.3 proband

This is a *Phenopacket* element that describes the proband in this case, a 14-year old boy.

```
proband:
  id: "14 year-old boy"
  subject:
    id: "14 year-old boy"
    timeAtLastEncounter:
      age:
        iso8601duration: "P14Y"
    sex: "MALE"
```

At this point in the *Phenopacket* element, there follows a list of phenotypic observations,

```
phenotypicFeatures: [ ... , ... , ... , ... ]
```

We present each phenotype separately in the following.

The following block describes Decreased fetal movement.

```
- type:
    id: "HP:0001558"
    label: "Decreased fetal movement"
  onset:
    ontologyClass:
      id: "HP:0011461"
      label: "Fetal onset"
  evidence:
  - evidenceCode:
      id: "ECO:0000033"
      label: "author statement supported by traceable reference"
    reference:
      id: "PMID:30808312"
      description: "COL6A1 mutation leading to Bethlem myopathy with recurrent␣
↪hematuria:\
        \ a case report."
```

This block refers to the fact that the authors reported that "Tests of . . . cranial nerves function were normal".

```
- type:
    id: "HP:0031910"
    label: "Abnormal cranial nerve physiology"
  excluded: true
  evidence:
  - evidenceCode:
      id: "ECO:0000033"
      label: "author statement supported by traceable reference"
    reference:
      id: "PMID:30808312"
      description: "COL6A1 mutation leading to Bethlem myopathy with recurrent␣
↪hematuria:\
        \ a case report."
```

This block refers to recurrent gross hematuria which had occured beginning six months before admission at age 14 years (We record the age as 14 years because more precise data is not presented).

```
- type:
    id: "HP:0011463"
    label: "Macroscopic hematuria"
  modifiers:
  - id: "HP:0031796"
    label: "Recurrent"
  onset:
    age:
      iso8601duration: "P14Y"
  evidence:
  - evidenceCode:
      id: "ECO:0000033"
      label: "author statement supported by traceable reference"
    reference:
      id: "PMID:30808312"
      description: "COL6A1 mutation leading to Bethlem myopathy with recurrent␣
↪hematuria:\
       \ a case report."
```

Finally, this block describe mild motor delay in childhood.

```
- type:
    id: "HP:0001270"
    label: "Motor delay"
  severity:
    id: "HP:0012825"
    label: "Mild"
  onset:
    ontologyClass:
      id: "HP:0011463"
      label: "Childhood onset"
```

7.1.4 relatives

Each of the relatives can be added as a *Phenopacket*. In this case, we add Phenopackets for the mother and father, both of whom are health. Therefore, the corresponding phenopackets only have fields for `id` and `sex`.

```
relatives:
- subject:
    id: "MOTHER"
    sex: "FEMALE"
- subject:
    id: "FATHER"
    sex: "MALE"
```

7.1.5 pedigree

The *Pedigree* object represents the information that is typically included in a PED file. It is important that the identifiers are the same as those used for the Phenopackets.

```
pedigree:
  persons:
  - individualId: "14 year-old boy"
    paternalId: "FATHER"
```

(continues on next page)

(continued from previous page)

```
    maternalId: "MOTHER"
    sex: "MALE"
    affectedStatus: "AFFECTED"
  - individualId: "MOTHER"
    sex: "FEMALE"
    affectedStatus: "UNAFFECTED"
  - individualId: "FATHER"
    sex: "MALE"
    affectedStatus: "UNAFFECTED"
```

7.1.6 metaData

The *MetaData* is required to provide details about all of the ontologies and external references used in the Phenopacket.

```
metaData:
  created: "2021-05-11T14:37:28.328Z"
  createdBy: "Peter R."
  resources:
  - id: "hp"
    name: "human phenotype ontology"
    url: "http://purl.obolibrary.org/obo/hp.owl"
    version: "2018-03-08"
    namespacePrefix: "HP"
    iriPrefix: "http://purl.obolibrary.org/obo/HP_"
  - id: "geno"
    name: "Genotype Ontology"
    url: "http://purl.obolibrary.org/obo/geno.owl"
    version: "19-03-2018"
    namespacePrefix: "GENO"
    iriPrefix: "http://purl.obolibrary.org/obo/GENO_"
  - id: "pubmed"
    name: "PubMed"
    namespacePrefix: "PMID"
    iriPrefix: "https://www.ncbi.nlm.nih.gov/pubmed/"
  phenopacketSchemaVersion: "2.0"
  externalReferences:
  - id: "PMID:30808312"
    description: "Bao M, et al. COL6A1 mutation leading to Bethlem myopathy with␣
↪recurrent\
      \ hematuria: a case report. BMC Neurol. 2019;19(1):32."
```

7.2 A complete example: Oncology

We will now present a phenopacket that represents a case of an individual with bladder cancer. We present each of the sections of the Phenopacket in separate subsections for legibility. Recall that JSON data is represented as as name/value pairs that are separated by commas (we show the trailing comma on all but the last name/value pair of the Phenopacket).

7.2.1 Infiltrating urothelial carcinoma: A case report

We here present a case report about an individual with infiltrating urothelial carcinoma (NCIT:C39853). Three somatic variants were identified in this sample which are judged to be related to tumorigenesis. Additionally, a secondary

finding of prostate carcinoma is made. The two distal ureter segments are found to be cancer-free. A pelvic lymph node is found to have a metastasis, but no molecular investigation was made. The patient was found to have two clinical abnormalities, hematuria (blood in the urine) and dysuria (painful urination).

In this example, we imagine that whole genome sequencing was performed on the infiltrating urothelial carcinoma as well as on the metastasis in the pelvic lymph node. A paired normal germline sample was also sequenced. All three files are located on some local file system, and this Phenopacket is used to organize the information about the diagnosis and phenotypes of the cancer in a way that would be able to support analysis of the WGS data. In a larger study, one such Phenopacket could organize the information for each of thousands of patients.

7.2.2 id

The *id* field is an arbitrary but required value.

```
id: "example case"
```

7.2.3 subject

The subject block is an *Individual* element. The *id* can be arbitrary identifiers (`id` is required).

```
subject:
  id: "patient1"
  dateOfBirth: "1964-03-15T00:00:00Z"
  sex: "MALE"
```

7.2.4 phenotypicFeatures

This field can be used to represent clinical manifestations using the phenotype element. Phenotypes directly related to the biopsied or extirpated tumor specimens should be reported in the *Biosample* element (see below). In this example, the patient is found to have Hematuria and severe Dysuria.

```
phenotypicFeatures:
   - type:
       id: "HP:0000790"
       label: "Hematuria"
   - type:
       id: "HP:0100518"
       label: "Dysuria"
     severity:
       id: "HP:0012828"
       label: "Severe"
```

7.2.5 biosamples

This is a list of *Biosample* elements that describe the evaluation of pathological examination of tumor specimens. We will present each *Biosample* in turn. The entire collection of biosamples is represented as follows.

```
biosamples: [  ... ,  ... , ....],
```

7.2.6 biosample 1: Infiltrating Urothelial Carcinoma

The first biosample is a biopsy taken from the wall of the urinary bladder. The histologuical diagnosis is represented by a National Cancer Institute's Thesaurus code. This is a primary malignant neoplasm with stage T2bN2. A VCF file representing a paired normal germline sample is located at `/data/genomes/urothelial_ca_wgs.vcf.gz` on the file system. In order to specify the procedure used to remove the bladder and prostate gland, we use the NCIT term for Radical Cystoprostatectomy (defined as the simultaneous surgical resection of the urinary bladder and prostate gland with pelvic lymphadenectomy).

```
- id: "sample1"
  individualId: "patient1"
  sampledTissue:
    id: "UBERON_0001256"
    label: "wall of urinary bladder"
  timeOfCollection:
    age:
      iso8601duration: "P52Y2M"
  histologicalDiagnosis:
    id: "NCIT:C39853"
    label: "Infiltrating Urothelial Carcinoma"
  tumorProgression:
    id: "NCIT:C84509"
    label: "Primary Malignant Neoplasm"
  procedure:
    code:
      id: "NCIT:C5189"
      label: "Radical Cystoprostatectomy"
  files:
  - uri: "file:///data/genomes/urothelial_ca_wgs.vcf.gz"
    individualToFileIdentifiers:
      patient1: "NA12345"
    fileAttributes:
      genomeAssembly: "GRCh38"
      fileFormat: "vcf"
      description: "Urothelial carcinoma sample"
  materialSample:
    id: "EFO:0009655"
    label: "abnormal sample"
```

7.2.7 Biosample 2: Prostate Acinar Adenocarcinoma

Prostate adenocarcinoma was discovered as an incidental finding. The tumor was found to have a Gleason score of 7.

```
- id: "sample2"
  individualId: "patient1"
  sampledTissue:
    id: "UBERON:0002367"
    label: "prostate gland"
  timeOfCollection:
    age:
      iso8601duration: "P52Y2M"
  histologicalDiagnosis:
    id: "NCIT:C5596"
    label: "Prostate Acinar Adenocarcinoma"
  tumorProgression:
    id: "NCIT:C95606"
```

(continues on next page)

(continued from previous page)

```
    label: "Second Primary Malignant Neoplasm"
  tumorGrade:
    id: "NCIT:C28091"
    label: "Gleason Score 7"
  procedure:
    code:
      id: "NCIT:C15189"
      label: "Biopsy"
```

7.2.8 Biosample 3: Left ureter

A biopsy of the left ureter reveal normal findings.

```
- id: "sample3"
  individualId: "patient1"
  sampledTissue:
    id: "UBERON:0001223"
    label: "left ureter"
  timeOfCollection:
    age:
      iso8601duration: "P52Y2M"
  histologicalDiagnosis:
    id: "NCIT:C38757"
    label: "Negative Finding"
  procedure:
    code:
      id: "NCIT:C15189"
      label: "Biopsy"
```

7.2.9 Biosample 4: Right ureter

A biopsy of the right ureter reveal normal findings.

```
- id: "sample4"
  individualId: "patient1"
  sampledTissue:
    id: "UBERON:0001222"
    label: "right ureter"
  timeOfCollection:
    age:
      iso8601duration: "P52Y2M"
  histologicalDiagnosis:
    id: "NCIT:C38757"
    label: "Negative Finding"
```

7.2.10 Biosample 4: Pelvic lymph node

A biopsy of a pelvic lymph node revealed a metastasis. A reference to a somatic genome sequence file is provided.

```
- id: "sample5"
 individualId: "patient1"
```

(continues on next page)

(continued from previous page)

```
  sampledTissue:
    id: "UBERON:0015876"
    label: "pelvic lymph node"
  timeOfCollection:
    age:
      iso8601duration: "P52Y2M"
  tumorProgression:
    id: "NCIT:C3261"
    label: "Metastatic Neoplasm"
  procedure:
    code:
      id: "NCIT:C15189"
      label: "Biopsy"
  files:
  - uri: "file://data/genomes/metastasis_wgs.vcf.gz"
    individualToFileIdentifiers:
      sample5: "BS730275"
    fileAttributes:
      genomeAssembly: "GRCh38"
      fileFormat: "vcf"
      description: "lymph node metastasis sample"
```

7.2.11 genes and variants

These elements of the Phenopacket are empty. One could have used them to specify that a certain gene or variant was identified that was inferred to be related to the tumor specimen (for instance, a germline mutation in a cancer susceptibility gene).

7.2.12 diseases

We recommend using the National Cancer Institute's Thesaurus codes to represent cancer diagnoses, but any relevant ontology term can be used. Information about tumor staging should be added here. See *Disease* for details.

```
diseases:
- term:
    id: "NCIT:C39853"
    label: "Infiltrating Urothelial Carcinoma"
  diseaseStage:
  - id: "NCIT:C27971"
    label: "Stage IV"
  clinicalTnmFinding:
  - id: "NCIT:C48766"
    label: "pT2b Stage Finding"
  - id: "NCIT:C48750"
    label: "pN2 Stage Finding"
  - id: "NCIT:C48700"
    label: "M1 Stage Finding"
```

7.2.13 htsFiles

This is a reference to the paired normal germline sample.

```
htsFiles:
- uri: "file://data/genomes/germline_wgs.vcf.gz"
  description: "Matched normal germline sample"
  htsFormat: "VCF"
  genomeAssembly: "GRCh38"
  individualToSampleIdentifiers:
    example case: "NA12345"
```

7.2.14 metaData

The *MetaData* is required to provide details about all of the ontologies and external references used in the Phenopacket.

```
metaData:
   created: "2021-05-11T15:07:16.662Z"
   createdBy: "Peter R"
   submittedBy: "Peter R"
   resources:
   - id: "hp"
     name: "human phenotype ontology"
     url: "http://purl.obolibrary.org/obo/hp.owl"
     version: "2019-04-08"
     namespacePrefix: "HP"
     iriPrefix: "http://purl.obolibrary.org/obo/HP_"
   - id: "uberon"
     name: "uber anatomy ontology"
     url: "http://purl.obolibrary.org/obo/uberon.owl"
     version: "2019-03-08"
     namespacePrefix: "UBERON"
     iriPrefix: "http://purl.obolibrary.org/obo/UBERON_"
   - id: "ncit"
     name: "NCI Thesaurus OBO Edition"
     url: "http://purl.obolibrary.org/obo/ncit.owl"
     version: "18.05d"
     namespacePrefix: "NCIT"
   phenopacketSchemaVersion: "2.0"
   externalReferences:
   - id: "PMID:29221636"
     description: "Urothelial neoplasms in pediatric and young adult patients: A␣
↪large\
       \ single-center series"
```

The Java code that was used to create this example is explained *here*.

7.3 A complete example: COVID-19

Here, we demonstrate how phenopackets are used to represent the time course of a case of severe COVID-19 based on a published case report. We will explain each of the components of the phenopacket in order.

7.3.1 subject

The subject is a 70 year-old male who died owing to complications of COVID-19. The time and cause of death are represented by the *VitalStatus* element. Note that the timestamp is with reference to the Unix time, defined as the

number of seconds that have elapsed since the Unix epoch, minus leap seconds; the Unix epoch is 00:00:00 UTC on 1 January 1970. Thus, 1585353600 corresponds to Saturday March 28, 2020 00:00:00 (AM).

```
subject:
   id: "P123542"
   sex: "MALE"
   timeAtLastEncounter:
     age:
       iso8601duration: "P70Y"
   vitalStatus:
     status: "DECEASED"
     timeOfDeath:
       timestamp: "2020-03-28T00:00:00Z"
     causeOfDeath:
       id: "MONDO:0100096"
       label: "COVID-19"
```

7.3.2 phenotypicFeatures

The phenotypic features here describe the more qualitative phenotypic features the patient exhibited. He is described as having an A+ blood group and suffering from a fever, flank pain, hematuria and later on myalgia, diarrhea and dyspnea which develops into acute respiratory failure three days after the initial onset of symptoms at which point he is admitted into hospital.

```
phenotypicFeatures:
- type:
    id: "NCIT:C76246"
    label: "Blood Group A"
- type:
    id: "NCIT:C76251"
    label: "Rh Positive Blood Group"
- type:
    id: "NCIT:C3038"
    label: "Fever"
  onset:
    timestamp: "2020-03-17T00:00:00Z"
- type:
    id: "NCIT:C34615"
    label: "Flank Pain"
  onset:
    timestamp: "2020-03-17T00:00:00Z"
- type:
    id: "NCIT:C3090"
    label: "Hematuria"
  onset:
    timestamp: "2020-03-17T00:00:00Z"
- type:
    id: "NCIT:C27009"
    label: "Myalgia"
  onset:
    interval:
      start: "2020-03-18T00:00:00Z"
      end: "2020-03-20T00:00:00Z"
- type:
    id: "NCIT:C2987"
    label: "Diarrhea"
```

(continues on next page)

(continued from previous page)

```
  onset:
    interval:
      start: "2020-03-18T00:00:00Z"
      end: "2020-03-20T00:00:00Z"
- type:
    id: "NCIT:C2998"
    label: "Dyspnea"
  onset:
    interval:
      start: "2020-03-18T00:00:00Z"
      end: "2020-03-20T00:00:00Z"
- type:
    id: "NCIT:C27043"
    label: "Acute Respiratory Failure"
  onset:
    timestamp: "2020-03-20T00:00:00Z"
```

7.3.3 measurements

The measurements block allows quantitative recording of measurements as opposed to qualitative. For example 'Fever' (NCIT:C3038) listed in the phenotypicFeatures section could be represented quantitatively as a *Measurement* rather than a *PhenotypicFeature*.

In this section we show two of the measurements for the absolute blood lymphocyte count listed in table 1 of the reference. This shows the historical count from measurements taken within the six months prior to hospital admission, followed by the count taken on admission.

```
measurements:
- assay:
    id: "NCIT:C113237"
    label: "Absolute Blood Lymphocyte Count"
  value:
    quantity:
      unit:
        id: "NCIT:C67245"
        label: "Thousand Cells"
      value: 1.4
      referenceRange:
        unit:
          id: "NCIT:C67245"
          label: "Thousand Cells"
        low: 1.0
        high: 4.5
  timeObserved:
    interval:
      start: "2019-09-01T00:00:00Z"
      end: "2020-03-01T00:00:00Z"
- assay:
    id: "NCIT:C113237"
    label: "Absolute Blood Lymphocyte Count"
  value:
    quantity:
      unit:
        id: "NCIT:C67245"
        label: "Thousand Cells"
```

(continues on next page)

(continued from previous page)

```
      value: 0.7
      referenceRange:
        unit:
          id: "NCIT:C67245"
          label: "Thousand Cells"
        low: 1.0
        high: 4.5
  timeObserved:
    timestamp: "2020-03-20T00:00:00Z"
```

7.3.4 diseases

This section displays a selection of the patient's prior medical history, without any timeframe indicating it included ischemic cardiomyopathy, stage 3 chronic kidney disease and obesity. He did not have diabetes. He had a positive PCR test for COVID-19 three days before admission to hospital.

```
diseases:
- term:
    id: "NCIT:C2985"
    label: "Diabetes Mellitus"
  excluded: true
- term:
    id: "NCIT:C34830"
    label: "Cardiomyopathy"
- term:
    id: "NCIT:C80389"
    label: "Chronic Kidney Disease, Stage 3"
- term:
    id: "NCIT:C3283"
    label: "Obesity"
- term:
    id: "MONDO:0100096"
    label: "COVID-19"
  onset:
    timestamp: "2020-03-17T00:00:00Z"
```

7.3.5 medicalActions

As post of his medical history it was noted he had previously had a left ventricular assist device implanted at some point in 2016. Given this as not precisely specified a timestamp for Jan 1st 2016 is stated. Once admitted to hospital for acute respiratory failure as a result of the COVID-19 infection the patient was administered nasal oxygen at 2L/min for two days before requiring the dosage to be increased to 50L/min on day 2. Days 2 - 8, the patient required intubation and oxygen was delivered via a positive end expiratory pressure valve device. He was treated with hydroxychloroquine for the first 2 days before being administered Tocilizumab.

```
medicalActions:
- procedure:
    code:
      id: "NCIT:C80473"
      label: "Left Ventricular Assist Device"
    performed:
      timestamp: "2016-01-01T00:00:00Z"
- treatment:
```

(continues on next page)

(continued from previous page)

```
    agent:
      id: "NCIT:C722"
      label: "Oxygen"
    routeOfAdministration:
      id: "NCIT:C38284"
      label: "Nasal Route of Administration"
    doseIntervals:
    - quantity:
        unit:
          id: "NCIT:C67388"
          label: "Liter per Minute"
        value: 2.0
      interval:
        start: "2020-03-20T00:00:00Z"
        end: "2020-03-22T00:00:00Z"
    - quantity:
        unit:
          id: "NCIT:C67388"
          label: "Liter per Minute"
        value: 50.0
      interval:
        start: "2020-03-22T00:00:00Z"
        end: "2020-03-23T00:00:00Z"
- treatment:
    agent:
      id: "NCIT:C557"
      label: "Hydroxychloroquine"
    doseIntervals:
    - quantity:
        unit:
          id: "NCIT:C28253"
          label: "mg"
        value: 450.0
      scheduleFrequency:
        id: "NCIT:C64496"
        label: "Twice Daily"
      interval:
        start: "2020-03-20T00:00:00Z"
        end: "2020-03-20T00:00:00Z"
    - quantity:
        unit:
          id: "NCIT:C28253"
          label: "mg"
        value: 450.0
      scheduleFrequency:
        id: "NCIT:C125004"
        label: "Once Daily"
      interval:
        start: "2020-03-21T00:00:00Z"
        end: "2020-03-22T00:00:00Z"
- procedure:
    code:
      id: "NCIT:C116648"
      label: "Tracheal Intubation"
    performed:
      timestamp: "2020-03-22T00:00:00Z"
- treatment:
```

(continues on next page)

(continued from previous page)

```
    agent:
      id: "NCIT:C722"
      label: "Oxygen"
    routeOfAdministration:
      id: "NCIT:C50254"
      label: "Positive end Expiratory Pressure Valve Device"
    doseIntervals:
    - quantity:
        unit:
          id: "NCIT:C91060"
          label: "Centimeters of Water"
        value: 14.0
      interval:
        start: "2020-03-22T00:00:00Z"
        end: "2020-03-28T00:00:00Z"
- treatment:
    agent:
      id: "NCIT:C84217"
      label: "Tocilizumab"
    doseIntervals:
    - interval:
        start: "2020-03-24T00:00:00Z"
        end: "2020-03-28T00:00:00Z"
```

7.3.6 metaData

The *MetaData* is required to provide details about all of the ontologies and external references used in the Phenopacket.

```
metaData:
  resources:
  - id: "ncit"
    name: "NCI Thesaurus OBO Edition"
    url: "http://purl.obolibrary.org/obo/ncit.owl"
    version: "http://purl.obolibrary.org/obo/ncit/releases/2019-11-26/ncit.owl"
    namespacePrefix: "NCIT"
  - id: "mondo"
    name: "Mondo Disease Ontology"
    url: "http://purl.obolibrary.org/obo/mondo.obo"
    namespacePrefix: "MONDO"
  - id: "doi"
    name: "Digital Object Identifier"
    url: "http://dx.doi.org"
    namespacePrefix: "DOI"
  - id: "pubmed"
    name: "PubMed"
    url: "https://pubmed.ncbi.nlm.nih.gov/"
    namespacePrefix: "PUBMED"
  phenopacketSchemaVersion: "2.0"
  externalReferences:
  - id: "DOI:10.1016/j.jaccas.2020.04.001"
    reference: "PMID:32292915"
    description: "The Imperfect Cytokine Storm: Severe COVID-19 With ARDS in a␣
↪Patient\
      \ on Durable LVAD Support"
```

Each example has a corresponding explanation of how to create the Phenopacket using Java.

7.4 Rare Disease

This page explains the Java code that was used to generate *A complete example: Rare Disease*. The complete code is available in the src/test package of this repository in the class `BethlemMyopathyExample`.

7.4.1 Builders and Short cuts

The individual elements of a Phenopacket are constructed with functions provided by the protobuf framework. These functions use the Builder pattern. For instance, to create an OntologyClass object, we use the following code.

```
OntologyClass hematuria = OntologyClass.newBuilder()
            .setId("HP:0000790")
            .setLabel("Hematuria")
            .build();
```

Developers may find it easier to define convenience functions that wrap the builders. For instance, for the OntologyClass example, we might define the following function.

```
public static OntologyClass ontologyClass(String id, String label) {
    return OntologyClass.newBuilder()
            .setId(id)
            .setLabel(label)
            .build();
}
```

We will use the `ontologyClass` function in our examples, but otherwise show all steps for clarity.

7.4.2 Family members and variants

We define the names of the family members.

```
private static final String PROBAND_ID = "14 year-old boy";
private static final String MOTHER_ID = "MOTHER";
private static final String FATHER_ID = "FATHER";
```

7.4.3 Proband

The following function then creates the Proband object. Note how we create OntologyClass objects for onset and severity modifiers, and create an Evidence object that indicates the provenance of the data.

```
static Phenopacket proband() {

       OntologyClass mild = OntologyClass.newBuilder().setId("HP:0012825").setLabel(
↪"Mild").build();

       OntologyClass evidenceCode = OntologyClass.newBuilder().
               setId("ECO:0000033").
               setLabel("author statement supported by traceable reference").
               build();

       Evidence citation = Evidence.newBuilder().
               setReference(ExternalReference.newBuilder().
```

(continues on next page)

(continued from previous page)

```
                        setId("PMID:30808312").
                        setDescription("COL6A1 mutation leading to Bethlem myopathy␣
↪with recurrent hematuria: a case report.").
                        build()).
                setEvidenceCode(evidenceCode)
                .build();

        PhenotypicFeature decreasedFetalMovement = PhenotypicFeature.newBuilder()
                .setType(ontologyClass("HP:0001558", "Decreased fetal movement"))
                .setOnset(TimeElement.newBuilder().setOntologyClass(ontologyClass(
↪"HP:0011461", "Fetal onset")).build())
                .addEvidence(citation)
                .build();

        PhenotypicFeature absentCranialNerveAbnormality = PhenotypicFeature.
↪newBuilder()
                .setType(ontologyClass("HP:0031910", "Abnormal cranial nerve physiology
↪"))
                .setExcluded(true)
                .addEvidence(citation)
                .build();

        PhenotypicFeature motorDelay = PhenotypicFeature.newBuilder()
                .setType(ontologyClass("HP:0001270","Motor delay"))
                .setOnset(TimeElement.newBuilder().setOntologyClass(ontologyClass(
↪"HP:0011463","Childhood onset")))
                .setSeverity(mild)
                .build();

        PhenotypicFeature hematuria = PhenotypicFeature.newBuilder()
                .setType(ontologyClass("HP:0011463", "Macroscopic hematuria"))
                .setOnset(TimeElement.newBuilder().setAge(Age.newBuilder().
↪setIso8601Duration("P14Y")))
                .addModifiers(ontologyClass("HP:0031796","Recurrent"))
                .addEvidence(citation)
                .build();


        Individual proband = Individual.newBuilder()
                .setSex(Sex.MALE)
                .setId(PROBAND_ID)
                .setTimeAtLastEncounter(TimeElement.newBuilder().setAge(Age.
↪newBuilder().setIso8601Duration("P14Y")))
                .build();
        return Phenopacket.newBuilder()
                .setId(PROBAND_ID)
                .setSubject(proband)
                .addPhenotypicFeatures(decreasedFetalMovement)
                .addPhenotypicFeatures(absentCranialNerveAbnormality)
                .addPhenotypicFeatures(hematuria)
                .addPhenotypicFeatures(motorDelay)
                .build();
    }
```

7.4.4 Unaffected parents

The unaffected father is coded as follows:

```
static Phenopacket unaffectedFather() {
     Individual father = Individual.newBuilder()
              .setSex(Sex.MALE)
              .setId(FATHER_ID)
              .build();
     return Phenopacket.newBuilder()
              .setSubject(father)
              .build();
 }
```

The mother is coded analogously. Note that in both cases, on two of the elements of the *Phenopacket* are actually used.

7.4.5 Pedigree

The following code builds the *Pedigree* object.

```
private static Pedigree pedigree() {
    Pedigree.Person pedProband = Pedigree.Person.newBuilder()
             .setIndividualId(PROBAND_ID)
             .setSex(Sex.MALE)
             .setMaternalId(MOTHER_ID)
             .setPaternalId(FATHER_ID)
             .setAffectedStatus(Pedigree.Person.AffectedStatus.AFFECTED)
             .build();

    Pedigree.Person pedMother = Pedigree.Person.newBuilder()
             .setIndividualId(MOTHER_ID)
             .setSex(Sex.FEMALE)
             .setAffectedStatus(Pedigree.Person.AffectedStatus.UNAFFECTED)
             .build();

    Pedigree.Person pedFather = Pedigree.Person.newBuilder()
             .setIndividualId(FATHER_ID)
             .setSex(Sex.MALE)
             .setAffectedStatus(Pedigree.Person.AffectedStatus.UNAFFECTED)
             .build();

    return Pedigree.newBuilder()
             .addPersons(pedProband)
             .addPersons(pedMother)
             .addPersons(pedFather)
             .build();
}
```

7.4.6 Family

Finally, the following code pulls everything together to build the Family object.

```
/**
 * Example taken from PMID:30808312
```

(continues on next page)

(continued from previous page)

```
 */
static Family bethlemMyopathyFamily() {

    long millis  = System.currentTimeMillis();
    Timestamp timestamp = Timestamp.newBuilder().setSeconds(millis / 1000)
            .setNanos((int) ((millis % 1000) * 1000000)).build();

    MetaData metaData = MetaData.newBuilder()
            .addResources(Resource.newBuilder()
                    .setId("hp")
                    .setName("human phenotype ontology")
                    .setNamespacePrefix("HP")
                    .setIriPrefix("http://purl.obolibrary.org/obo/HP_")
                    .setUrl("http://purl.obolibrary.org/obo/hp.owl")
                    .setVersion("2018-03-08")
                    .build())
            .addResources(Resource.newBuilder()
                    .setId("geno")
                    .setName("Genotype Ontology")
                    .setNamespacePrefix("GENO")
                    .setIriPrefix("http://purl.obolibrary.org/obo/GENO_")
                    .setUrl("http://purl.obolibrary.org/obo/geno.owl")
                    .setVersion("19-03-2018")
                    .build())
            .addResources(Resource.newBuilder()
                    .setId("pubmed")
                    .setName("PubMed")
                    .setNamespacePrefix("PMID")
                    .setIriPrefix("https://www.ncbi.nlm.nih.gov/pubmed/")
                    .build())
            .setCreatedBy("Peter R.")
            .setCreated(timestamp)
            .setPhenopacketSchemaVersion("2.0")
            .addExternalReferences(ExternalReference.newBuilder()
                    .setId("PMID:30808312")
                    .setDescription("Bao M, et al. COL6A1 mutation leading to Bethlem␣
↪myopathy with recurrent hematuria: " +
                            "a case report. BMC Neurol. 2019;19(1):32.")
                    .build())
            .build();

    return Family.newBuilder()
            .setId("family")
            .setProband(proband())
            .addAllRelatives(ImmutableList.of(unaffectedMother(), unaffectedFather()))
            .setPedigree(pedigree())
            .setMetaData(metaData)
            .build();
}
```

Note that we use `System.currentTimeMillis()` to get the current time (when we are creating and submitting this Phenopacket).

7.5 A complete example in Java: Oncology

This example shows how to create the Phenopacket that was explained *here*.

7.5.1 subject

We create an object to represent the proband as an *Individual*.

```
private Individual subject() {
      return Individual.newBuilder()
               .setId(this.patientId)
               .setSex(Sex.MALE)
               .setDateOfBirth(Timestamp.newBuilder()
                       .setSeconds(Instant.parse("1964-03-15T00:00:00Z").
↪getEpochSecond()))
               .build();
  }
```

7.5.2 phenotypicFeatures

There are two categories of phenotypes that can be of interest with cancer data. Firstly, there are constitutional phenotypes such as weight loss that are related to the disease of cancer. Second, the tumor, and is applicable metasases, each have their own phenotypes including histology and grade. The Phenopacket standard represents constitutional Phenotypes using a list of rstphenotype elements, and represents phenotypes of the tumor and metastases in *Biosample* elements. In the present case, the patient was found to have hematuria and severe dysuria, which are coded as follows.

```
PhenotypicFeature hematuria = PhenotypicFeature.newBuilder()
        .setType(ontologyClass("HP:0000790","Hematuria"))
        .build();
PhenotypicFeature dsyuria = PhenotypicFeature.newBuilder()
        .setType(ontologyClass("HP:0100518","Dysuria"))
        .setSeverity(ontologyClass("HP:0012828","Severe"))
        .build();
```

7.5.3 File

We use three *File* objects in this Phenopacket. One represents the pair normal germline whole-genome sequence (WGS) VCF file, one one each represents somatic WGS data from the bladder carcinoma specimen and from the metastasis specimen. All three packets are created analogously. Here is the code for the bladder carcinoma WGS file.

```
public File createSomaticHtsFile() {
    // first create a File
    // We are imagining there is a reference to a VCF file for a normal germline␣
↪genome seqeunce
    String path = "file://data/genomes/urothelial_ca_wgs.vcf.gz";
    String description = "Urothelial carcinoma sample";
    // Now create an HtsFile object
    return HtsFile.newBuilder()
             .setHtsFormat(HtsFile.HtsFormat.VCF)
             .setGenomeAssembly("GRCh38")
             .setUri(path)
             .setDescription(description)
```

(continues on next page)

(continued from previous page)

```
            .putIndividualToSampleIdentifiers("sample1", "BS342730")
            .build();
}
```

7.5.4 biosamples

This example Phenopacket contains five *Biosample* objects, each of which is constructed using a function similar to the following code, which represents the bladder carcinoma specimen.

```
private Biosample bladderBiosample() {
      String sampleId = "sample1";
      // left wall of urinary bladder
      OntologyClass sampleType = ontologyClass("UBERON_0001256", "wall of urinary␣
↪bladder");
      Biosample.Builder biosampleBuilder = biosampleBuilder(patientId, sampleId, this.
↪ageAtBiopsy, sampleType);
      // also want to mention the procedure, Prostatocystectomy (NCIT:C94464)
      //Infiltrating Urothelial Carcinoma (Code C39853)
      biosampleBuilder.setHistologicalDiagnosis(ontologyClass("NCIT:C39853",
↪"Infiltrating Urothelial Carcinoma"));
      // A malignant tumor at the original site of growth
      biosampleBuilder.setTumorProgression(ontologyClass("NCIT:C84509", "Primary␣
↪Malignant Neoplasm"));
      biosampleBuilder.addHtsFiles(createSomaticHtsFile());
      biosampleBuilder.setProcedure(Procedure.newBuilder().setCode(ontologyClass(
↪"NCIT:C5189", "Radical Cystoprostatectomy")).build());
      return biosampleBuilder.build();
  }
```

7.5.5 Normal findings

In the biosamples for the left and right ureter, normal findings were obtains. This is represented by an *OntologyClass* for normal (negative) findings. We recommend using the following term from NCIT.

```
OntologyClass normalFinding = ontologyClass("NCIT:C38757", "Negative Finding");
```

This is used to create a "normal" *Biosample* object as follows.

```
private Biosample leftUreterBiosample() {
     String sampleId = "sample3";
     OntologyClass sampleType = ontologyClass("UBERON:0001223", "left ureter");
     Biosample.Builder biosampleBuilder = biosampleBuilder(patientId, sampleId, this.
↪ageAtBiopsy, sampleType);
     OntologyClass normalFinding = ontologyClass("NCIT:C38757", "Negative Finding");
     biosampleBuilder.setHistologicalDiagnosis(normalFinding);
     biosampleBuilder.setProcedure(Procedure.newBuilder().setCode(ontologyClass(
↪"NCIT:C15189", "Biopsy")).build());
     return biosampleBuilder.build();
 }
```

7.5.6 diseases

We recommend using the National Cancer Institute's Thesaurus codes to represent cancer diagnoses, but any relevant ontology term can be used. The following Java code creates a *Disease* object.

```
private Disease infiltratingUrothelialCarcinoma() {
    return Disease.newBuilder()
            .setTerm(ontologyClass("NCIT:C39853", "Infiltrating Urothelial Carcinoma
↪"))
            // Disease stage here is calculated based on the TMN findings
            .addDiseaseStage(ontologyClass("NCIT:C27971", "Stage IV"))
            // The tumor was staged as pT2b, meaning infiltration into the outer␣
↪muscle layer of the bladder wall
            // pT2b Stage Finding (Code C48766)
            .addClinicalTnmFinding(ontologyClass("NCIT:C48766", "pT2b Stage Finding"))
            //pN2 Stage Finding (Code C48750)
            // cancer has spread to 2 or more lymph nodes in the true pelvis (N2)
            .addClinicalTnmFinding(ontologyClass("NCIT:C48750", "pN2 Stage Finding"))
            // M1 Stage Finding
            // the tumour has spread from the original site (Metastatic Neoplasm in␣
↪lymph node - sample5)
            .addClinicalTnmFinding(ontologyClass("NCIT:C48700", "M1 Stage Finding"))
            .build();
}
```

7.5.7 Metadata

The *MetaData* section MUST indicate all ontologies used in the phenopacket together with their versions. This Phenopacket used HPO, UBERON, and NCIT. We additionally use a *Timestamp (Java)* object to indicate the current time (at which we are creating this Phenopacket).

```
private MetaData buildMetaData() {
    long millis  = System.currentTimeMillis();
    Timestamp timestamp = Timestamp.newBuilder().setSeconds(millis / 1000)
            .setNanos((int) ((millis % 1000) * 1000000)).build();
    return MetaData.newBuilder()
            .addResources(Resource.newBuilder()
                    .setId("hp")
                    .setName("human phenotype ontology")
                    .setNamespacePrefix("HP")
                    .setIriPrefix("http://purl.obolibrary.org/obo/HP_")
                    .setUrl("http://purl.obolibrary.org/obo/hp.owl")
                    .setVersion("2019-04-08")
                    .build())
            .addResources(Resource.newBuilder()
                    .setId("uberon")
                    .setName("uber anatomy ontology")
                    .setNamespacePrefix("UBERON")
                    .setIriPrefix("http://purl.obolibrary.org/obo/UBERON_")
                    .setUrl("http://purl.obolibrary.org/obo/uberon.owl")
                    .setVersion("2019-03-08")
                    .build())
            .addResources(Resource.newBuilder()
                    .setId("ncit")
                    .setName("NCI Thesaurus OBO Edition")
                    .setNamespacePrefix("NCIT")
```

(continues on next page)

(continued from previous page)

```
                    .setUrl("http://purl.obolibrary.org/obo/ncit.owl")
                    .setVersion("18.05d")
                    .build())
            .setCreatedBy("Peter R")
            .setCreated(timestamp)
            .setSubmittedBy("Peter R")
            .setPhenopacketSchemaVersion("2.0")
            .addExternalReferences(ExternalReference.newBuilder()
                    .setId("PMID:29221636")
                    .setDescription("Urothelial neoplasms in pediatric and young␣
↪adult patients: A large single-center series")
                    .build())
            .build();
}
```

7.5.8 Putting it all together

Finally, we utilize a Phenopacket builder to generate the complete Phenopacket object.

```
Phenopacket phenopacket = Phenopacket.newBuilder()
            .setId("example case")
            .setSubject(subject())
            .addPhenotypicFeatures(hematuria)
            .addPhenotypicFeatures(dsyuria)
            .addBiosamples(bladderBiosample())
            .addBiosamples(prostateBiosample())
            .addBiosamples(leftUreterBiosample())
            .addBiosamples(rightUreterBiosample())
            .addBiosamples(pelvicLymphNodeBiosample())
            .addDiseases(infiltratingUrothelialCarcinoma())
            .addHtsFiles(createNormalGermlineHtsFile())
            .setMetaData(metaData)
            .build();
```

7.5.9 Output of data

There are many ways of outputting the Phenopacket in JSON format. See *Exporting and Importing Phenopackets* for details. The following line will output the entire Phenopacket to STDOUT as YAML, using the Jackson library.

```
String json = JsonFormat.printer().print(phenopacket);
JsonNode jsonNodeTree = new ObjectMapper().readTree(json);
String yaml = new YAMLMapper().writeValueAsString(jsonNodeTree);
System.out.println(yaml);
```