# phenopackets-schema Documentation

***Release 10.0.0***

**Julius OB Jacobsen, Peter N Robinson, Christopher J Mungall**

**Nov 27, 2019**

# Contents

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.

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 phenoptypes 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.

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.

### 1.2.1 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 practive validation software does not need to check for null pointers.

### 1.2.2 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 woirds `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.

### 1.2.3 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 `background` field of the `variant` message is only used for Phenopackets that describe animal models of disease.

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 PubMed.

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.

The phenopacket schema requires the use of a common ontology, a logically defined hierarchy of terms, that allows sophisticated algorithmic analysis over medically relevant abnormalities. 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 Human Phenotype Ontology (HPO) was built

for genomic diagnostics, translational research, genomic matchmaking, and

systems biology applications in the field of rare disease and other fields of 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.

## 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 phenopackets-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 phenopackets-schema, but is intended to build intuition for how protobuf works. We first need to install protobuf (Note that these intructions 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 sucessful

```
$ 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() {
  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>
```

(continues on next page)

(continued from previous page)

```
  <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 simplcity, the following code snippets could be written in the main method

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

(continues on next page)

(continued from previous page)

```
    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

Phenopackets on FHIR is a FHIR implementation guide based on the Phenopackets standard. It is meant to be conceptually equivalent but not directly interoperable. The guide is hosted at:

https://genomics.ontoserver.csiro.au/phenopackets/

Briefly, the phenopacket equivalents to various FHIR resources are as follows:

*Age* is mapped to the FHIR using Unified Code for Units of Measure (UCUM). See also Condition onset.

*Biosample* maps to Specimen.

*Evidence* maps to Condition.evidence.

*ExternalReference* maps to Reference.

*Individual* maps to Patient.

*OntologyClass* maps to CodeableConcept. See also Coding.

*PhenotypicFeature* maps to Condition or Observation elements. The FHIR mapping of the type element of PhenotypicFeature is *Condition.identifier*, the mapping of the severity element is *Condition.severity*, the mapping of onset is *Condition.onset*.

*Procedure* maps to Procedure.

*Sex* maps to AdministrativeGender.

*Resource* maps to CodeSystem.

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 comittee and the HL7 Fast Healthcare Interoperability Resources Specification (aka FHIR®).

## 2.1 Overview

The diagram below shows an overview of the schema elements.

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

# 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 | Status | Definition |
|---|---|---|---|
| id | string | required | arbitrary identifier |
| subject | *Individual* | recommended | The proband |
| phenotypic_features | List of *PhenotypicFeature* | recommended | Phenotypic features observed in the proband |
| biosamples | *Biosample* | optional | samples (e.g., biopsies), if any |
| genes | *Gene* | optional | Gene deemed to be relevant to the case (application specific) |
| variants | List of *Variant* | optional | Variants identified in the proband |
| diseases | List of *Disease* | optional | Disease(s) diagnosed in the proband |
| hts_files | List of *HtsFile* | optional | VCF or other high-throughput sequencing files |
| meta_data | *MetaData* | required | Information about ontologies and references used in the phenopacket |

### 3.1.1 id

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

### 3.1.2 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.

### 3.1.3 phenotypic_features

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

### 3.1.4 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.

### 3.1.5 genes

This is a field for gene identifiers and can be used for listing either candidate genes or causative genes. The resources using these fields should define what this represents in their context. This could be used in order to obfuscate the specific causative/candidate variant to maintain patient privacy. See *Gene* for further information.

### 3.1.6 variants

This is a field for genetic variants and can be used for listing either candidate variants or diagnosed causative variants. The resources using these fields should define what this represents in their context. See *Variant* for further information.

### 3.1.7 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.

### 3.1.8 hts_files

This element contains a list of pointers to the relevant HTS file(s) for the patient. Each element describes what type of file is meant (e.g., BAM file), which genome assembly was used for mapping, as well as a map of samples and individuals represented in that file. It also contains a URI element which refers to a file on a given file system or a resource on the web.

See *HtsFile* for further information.

### 3.1.9 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.

**Data model**

Table 2: Definition of the `Family` element

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | application-specific identifier |
| proband | *Phenopacket* | required | The proband (index patient) in the family |
| relatives | *Phenopacket* (list) | optional | list of Phenopackets for family members other than the proband |
| pedigree | *Pedigree* | required | representation of the pedigree |
| hts_files | *HtsFile* (list) | optional | list of high-throughput sequencing files |
| meta_data | *MetaData* | required | Metadata about ontologies and references used in this message |

### 3.2.1 id

An identifier specific for this family.

### 3.2.2 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.

### 3.2.3 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.

### 3.2.4 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.

### 3.2.5 hts_files

This element contains a list of pointers to the relevant HTS file(s) for the family as a whole. For a `Family` these files MUST be merged/multi-sample files with appropriate genotype information. For a multi-sample file, the sample identifiers 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 *HtsFile* for further information.

### 3.2.6 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.

**Data model**

Table 3: Definition of the `Cohort` element

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | arbitrary identifier |
| description | string | optional | arbitrary text |
| members | *Phenopacket* | required | Phenopackets that represent members of the cohort |
| hts_files | *HtsFile* | optional | High-thoughput sequencing files obtained from members of the cohort |
| meta_data | *MetaData* | required | Metadata related to the ontologies and references used in this message |

### 3.3.1 id

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

### 3.3.2 description

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

### 3.3.3 members

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

### 3.3.4 hts_files

This element contains a list of pointers to the relevant HTS file(s) for the cohort. The HTS file MUST be a multi-sample file referring to the entire cohort, if appropriate. Individual HTS files MUST otherwise 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 genomic data derived from sequencing that biosample. Each element describes what type of file is meant (e.g., BAM file), which genome assembly was used for mapping, as well as a map of samples and individuals represented in that file. It also contains a URI element which refers to a file on a given file system or a resource on the web.

See *HtsFile* for further information.

### 3.3.5 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.

## 3.4 Interpretation

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

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | Arbitrary identifier |
| resolution_status | *Resolution_status* | required | The current status of work on the case |
| phenopacket_or_family | *Phenopacket* or *Family* element | required | The subject of this interpretation |
| diagnosis | *Diagnosis* | repeated | One or more diagnoses, if made |
| meta_data | See *MetaData* | required | Metadata about this interpretation |

**Example**

```
{
"id": "SOLVERD:0000123456",
"resolutionStatus": "SOLVED",
"phenopacket": {
  "id": "SOLVERD:0000234567",
  "subject": {
    "id": "SOLVERD:0000345678",
    "dateOfBirth": "1998-01-01T00:00:00Z",
    "sex": "MALE"
  },
```

(continues on next page)

(continued from previous page)

```
  "phenotypicFeatures": [{
    "type": {
      "id": "HP:0001159",
      "label": "Syndactyly"
    },
    "classOfOnset": {
      "id": "HP:0003577",
      "label": "Congenital onset"
    }
  }, {
    "type": {
      "id": "HP:0002090",
      "label": "Pneumonia"
    },
    "classOfOnset": {
      "id": "HP:0011463",
      "label": "Childhood onset"
    }
  }, {
    "type": {
      "id": "HP:0000028",
      "label": "Cryptorchidism"
    },
    "classOfOnset": {
      "id": "HP:0003577",
      "label": "Congenital onset"
    }
  }, {
    "type": {
      "id": "HP:0011109",
      "label": "Chronic sinusitis"
    },
    "severity": {
      "id": "HP:0012828",
      "label": "Severe"
    },
    "classOfOnset": {
      "id": "HP:0003581",
      "label": "Adult onset"
    }
  }],
  "variants": [{
    "hgvsAllele": {
      "hgvs": "NM_001361.4:c.403C\u003eT"
    },
    "zygosity": {
      "id": "GENO:0000135",
      "label": "heterozygous"
    }
  }, {
    "hgvsAllele": {
      "hgvs": "NM_001361.4:c.454G\u003eA"
    },
    "zygosity": {
      "id": "GENO:0000135",
      "label": "heterozygous"
    }
```

(continues on next page)

(continued from previous page)

```
    }, {
      "hgvsAllele": {
        "hgvs": "NM_001369.2:c.12599dupA"
      },
      "zygosity": {
        "id": "GENO:0000136",
        "label": "homozygous"
      }
    }]
  },
  "diagnosis": [{
    "disease": {
      "term": {
        "id": "OMIM:263750",
        "label": "Miller syndrome"
      }
    },
    "genomicInterpretations": [{
      "status": "CAUSATIVE",
      "gene": {
        "id": "HGNC:2867",
        "symbol": "DHODH"
      }
    }]
  }]
}
```

### 3.4.1 id

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

### 3.4.2 Resolution_status

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

**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 | 0 | No information is available about the diagnosis |
| SOLVED | 1 | The interpretation is considered to be a definitive diagnosis |
| UNSOLVED | 2 | No definitive diagnosis was found |
| IN_PROGRESS | 3 | No diagnosis has been found to date but additional differential diagnostic work is in progress. |

**Example**

```
{
  "resolutionStatus": "SOLVED"
}
```

### 3.4.3 phenopacket_or_family

This element refers to the *Phenopacket* or *Family* element for whom the interpretation is being made.

### 3.4.4 diagnosis

This refers to the diagnosis (or if applicable to the diagnoses) made. See *Diagnosis*, below.

### 3.4.5 meta_data

This element contains structured definitions of the resources and ontologies used within the phenopacket. See *Meta-Data* for further information.

### 3.4.6 Diagnosis

The diagnosis element is meant to refer to the disease that is infered 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 | Status | Description |
|---|---|---|---|
| disease | *Disease* | required | The diagnosed condition |
| genomic_interpretations | *GenomicInterpretation* | repeated | The genomic elements assessed as being responsible for the disease or empty |

**Example**

```
{
  "disease": {
    "term": {
      "id": "OMIM:263750",
      "label": "Miller syndrome"
    }
  },
  "genomicInterpretations": [{
    "status": "CAUSATIVE",
    "gene": {
      "id": "HGNC:2867",
      "symbol": "DHODH"
    }
  }]
}
```

The *genomic_interpretations* should be used if the genetic findings were used to help make the diagnosis, but can be omitted if genetic/genomic analysis was not contributory or were not performed.

### 3.4.7 GenomicInterpretation

A statement about the contribution of a genomic element towards the observed phenotype. Note that this does not intend to encode any knowledge or results of specific computations.

**Data model**

| Field | Type | Status | Example |
|---|---|---|---|
| status | *GenomicInterpretation Status* | required | How the *call* of this *GenomicInterpretation* was interpreted |
| call | *Gene* or *Variant* | required | The gene or variant contributing to the diagnosis |

**Example**

```
{
  "status": "CAUSATIVE",
  "gene": {
    "id": "HGNC:2867",
    "symbol": "DHODH"
  }
}
```

A gene can be listed as **CAUSATIVE**. Alternatively, or additionally, a variant may be listed as **CAUSATIVE**. Note that the intended semantics is different from the ACMG interpretation of sequence variants, which classifies variants with respect to their pathogenicity. The Interpretation element classifies variants as being responsible or not for the phenotypic and disease observations in the proband. A variant can be pathogenic according to the ACMG guidelines but not be causative for the particular disease being investigated (for instance, a heterozygous variant associated with an autosomal recessive disease may be found in a proband with causative variants in another gene).

### 3.4.8 GenomicInterpretation Status

An enumeration describing the status of a *GenomicInterpretation*

**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 | 0 | It is not known how this genomic element contributes to the diagnosis |
| REJECTED | 1 | The genomic element has been investigated and ruled-out |
| CANDIDATE | 2 | The genomic element is under investigation |
| CAUSATIVE | 3 | The genomic element has been judged to be contributing to the diagnosis |

**Example**

```
{
  "status": "CAUSATIVE"
}
```

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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| age | string | required | An ISO8601 string represent age |

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

**Example**

```
{
    "age": "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.

### 4.1.1 age

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

## 4.2 AgeRange

The AgeRange element is inteded 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| start | *Age* | required | An Age message |
| end | *Age* | required | An Age message |

**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.

```
{
  "start": {
      "age": "P45Y"
  },
  "end": {
      "age": "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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | arbitrary identifier |
| individual_id | string | recommended | arbitrary identifier |
| description | string | optional | arbitrary text |
| sampled_tissue | *OntologyClass* | required | Tissue from which the sample was taken |
| phenotypic_features | *PhenotypicFeature* | recommended | List of phenotypic abnormalities of the sample |
| taxonomy | *OntologyClass* | optional | Species of the sampled individual |
| individual_age_at_collection | *Age* OR *AgeRange* | recommended | Age of the proband at the time the sample was taken |
| histological_diagnosis | *OntologyClass* | recommended | Disease diagnosis that was inferred from the histological examination |
| tumor_progression | *OntologyClass* | recommended | Indicates primary, metastatic, recurrent |
| tumor_grade | *OntologyClass* | recommended | List of terms representing the tumor grade |
| diagnostic_markers | *OntologyClass* | recommended | Clinically relevant biomarkers |
| procedure | *Procedure* | required | The procedure used to extract the biosample |
| hts_files | *HtsFile* | optional | list of high-throughput sequencing files derived from the biosample |
| variants | *Variant* | optional | List of variants determined to be present in the biosample |
| is_control_sample | boolean | optional (default: false) | whether the sample is being used as a normal control |

**Example**

```
{
  "id": "sample1",
  "individualId": "patient1",
  "description": "",
  "sampledTissue": {
    "id": "UBERON_0001256",
    "label": "wall of urinary bladder"
  },
  "ageOfIndividualAtCollection": {
    "age": "P52Y2M"
  },
  "histologicalDiagnosis": {
    "id": "NCIT:C39853",
    "label": "Infiltrating Urothelial Carcinoma"
  },
  "tumorProgression": {
    "id": "NCIT:C84509",
    "label": "Primary Malignant Neoplasm"
  },
  "procedure": {
    "code": {
      "id": "NCIT:C5189",
```

(continues on next page)

(continued from previous page)

```
      "label": "Radical Cystoprostatectomy"
    }
  },
  "htsFiles": [{
    "uri": "file://data/genomes/urothelial_ca_wgs.vcf.gz",
    "description": "Urothelial carcinoma sample"
    "htsFormat": "VCF",
    "genomeAssembly": "GRCh38",
    "individualToSampleIdentifiers": {
      "patient1": "NA12345"
    }
  }],
  "variants": [],
  "isControlSample": false
}
```

### 4.3.1 id

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

### 4.3.2 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*.

### 4.3.3 description

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

### 4.3.4 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`.

### 4.3.5 phenotypic_features

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

### 4.3.6 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.

### 4.3.7 individual_age_at_collection

An *Age* or *AgeRange* describing the age or age range of the individual this biosample was derived from at the time of collection. See *Age* for further information.

### 4.3.8 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.

### 4.3.9 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

### 4.3.10 tumor_grade

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

### 4.3.11 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).

### 4.3.12 procedure

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

### 4.3.13 hts_files

This element contains a list of pointers to the relevant HTS file(s) for the biosample. Each element describes what type of file is meant (e.g., BAM file), which genome assembly was used for mapping, as well as a map of samples and individuals represented in that file. It also contains a URI element which refers to a file on a given file system or a resource on the web.

See *HtsFile* for further information.

### 4.3.14 variants

This is a field for genetic variants and can be used for listing either candidate variants or diagnosed causative variants. If this biosample represents a cancer specimen, the variants might refer to somatic variants identified in the biosample. The resources using these fields should define what this represents in their context. See *Variant* for further information.

### 4.3.15 is_control_sample

A boolean (true/false) value. If true, this sample is being use as a normal control, often in combination with another sample that is thought to contain a pathological finding the default value is false.

## 4.4 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., and 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| term | *OntologyClass* | required | An ontology class that represents the disease |
| onset | *Age* or *AgeRange* or *OntologyClass* | optional | an element representing the age of onset of the disease |
| disease_stage | *OntologyClass* | optional | List of terms representing the disease stage e.g. AJCC stage group. |
| tnm_finding | *OntologyClass* | optional | List of terms representing the tumor TNM score |

**Example**

```
{
"term": {
  "id": "OMIM:164400",
  "label": "Spinocerebellar ataxia 1 "
  },
"ageOfOnset": {
  "age": "P38Y7M"
  }
}
```

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

### 4.4.1 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).

### 4.4.2 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.

### 4.4.3 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.

### 4.4.4 age_of_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.

## 4.5 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)

**Data model**

Table 1: Definition the `Evidence` element

| Field | Type | Status | Description |
|---|---|---|---|
| evidence_code | *OntologyClass* representing ECO:0006017 | required | An ontology class that represents the evidence type |
| reference | *ExternalReference* | optional | Representation of the source of the evidence |

**Example**

```
{
  "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"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

### 4.5.1 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*).

### 4.5.2 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.

## 4.6 ExternalReference

This element encodes information about an external reference.

**Data model**

Table 2: Definition of the `ExternalReference` element

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | An application specific identifier |
| description | string | optional | An application specific description |

**Example**

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

### 4.6.1 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 and corresponding *Resource*.

### 4.6.2 description

An optional free text description of the evidence.

## 4.7 Gene

This element represents an identifier for a gene. It 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.

**Data model**

Table 3: Definition of the `Gene` element

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | Official identifier of the gene |
| alternate_ids | repeated string | optional | Alternative identifier(s) of the gene |
| symbol | string | required | Official gene symbol |

**Example**

```
{
  "id": "HGNC:347"
  "symbol": "ETF1"
}
```

Optionally, with alternative identifiers:

```
{
  "id": "HGNC:347",
  "alternate_ids": ["ensembl:ENSRNOG00000019450", "ncbigene:307503"],
  "symbol": "ETF1"
}
```

### 4.7.1 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

### 4.7.2 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*.

### 4.7.3 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.

### 4.7.4 Model Organisms

Model organisms represented by the Alliance of Genome Resources should use the primary identifier and symbol provided. e.g. for Mus musculus gene eukaryotic translation termination factor 1

```
{
  "id": "MGI:2385071",
  "alternate_ids": ["ensembl:ENSMUSG00000024360", "ncbigene:225363"],
  "symbol": "Etf1"
}
```

## 4.8 HtsFile

Phenopackets can be used to hold phenotypic information that can inform the analysis of sequencing data in VCF format as well as other high-throughput sequencing (HTS) or other data types. The HtsFile message allows a Phenopacket to link HTS files with data.

Given that HtsFile elements are listed in various locations such as the `Phenopacket`, `Biosample`, `Family` etc. which can in turn be nested, individual HTS 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 that biosample. Aggregate data types such as `Cohort` and `Family` MUST contain aggregate HTS 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 HTS files pertaining to that individual/biosample(s).

### 4.8.1 HtsFile

| Field | Type | Status | Description |
|---|---|---|---|
| uri | string | required | A valid URI e.g. file://data/file1.vcf.gz or https://opensnp.org/data/60.23andme-exome-vcf.231?1341012444 |
| description | string | optional | arbitrary description of the file |
| hts_format | *HtsFormat* | required | VCF |
| genome_assembly | string | required | e.g. GRCh38 |
| individual_to_sample_identifiers | map: string key: value | recommended | The mapping between the Individual.id or Biosample.id to the sample identifier in the HTS file |

### 4.8.2 HtsFormat

This message is used for a file in one of the HTS formats.

| Field | Description |
|---|---|
| UNKNOWN | An HTS file of unknown type. |
| SAM | A SAM format file |
| BAM | A BAM format file |
| CRAM | A CRAM format file |
| VCF | A VCF format file |
| BCF | A BCF format file |
| GVCF | A GVCF format file |
| FASTQ | A FASTQ format file |

**Example**

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

### 4.8.3 uri

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

### 4.8.4 description

An arbitrary description of the file contents.

The *File* message MUST have at least one of *path* and *uri* and usually should just have one of the two (in exceptional cases the same file might be referenced on a local file system and on the network).

### 4.8.5 hts_format

This indicates which format the file has.

### 4.8.6 genome_assembly

The genome assembly the contents of this file was called against. We recommend using the Genome Reference Consortium nomenclature e.g. GRCh37, GRCh38.

### 4.8.7 individual_to_sample_identifiers

A map of identifiers mapping an individual refered to in the Phenopacket to a sample 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 the samples in the HTS file.

## 4.9 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | An arbitrary identifier |
| alternate_ids | a list of *CURIE* | optional | A list of alternative identifiers for the individual |
| date_of_birth | timestamp | optional | A timestamp either exact or imprecise |
| age | *Age* or *AgeRange* | recommended | The age or age range of the individual |
| sex | *Sex* | recommended | Observed apparent sex of the individual |
| karyotypic_sex | *KaryotypicSex* | optional | The karyotypic sex of the individual |
| taxonomy | *OntologyClass* | optional | an *OntologyClass* representing the species (e.g., NCBITaxon:9615) |

**Example**

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

```
{
    "id": "patient:0",
    "dateOfBirth": "1998-01-01T00:00:00Z",
    "sex": "MALE"
}
```

### 4.9.1 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 rsthtsfile.

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 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 rsthtsfile 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.

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

"htsFile": {
    "uri": "file://data/genomes/germline_wgs.vcf.gz",
    "description": "Germline sample",
    "htsFormat": "VCF",
    "genomeAssembly": "GRCh38",
    "individualToSampleIdentifiers": {
      "patient23456": "NA12345"
    }
}

"pedigree": {
    "persons": [
        {
            "familyId": "family 1",
            "individualId": "patient23456",
            "sex": "MALE",
            "affectedStatus": "AFFECTED"
        }
    ]
}
```

### 4.9.2 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.

### 4.9.3 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 `Age` element should be preferred.

### 4.9.4 age

An age object describing 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 *Age element*, which can represent an Age in several different ways, 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).

### 4.9.5 sex

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

### 4.9.6 karyotypic_sex

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

### 4.9.7 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.

## 4.10 KaryotypicSex

This enumeration represents the chromosomal sex of an individual.

**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 |

## 4.11 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.

**Data model**

Table 4: Definition of the `MetaData` element

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

**Example**

```
{
  "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": "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/"
```

(continues on next page)

(continued from previous page)

```
  }],
  "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."
  }]
}
```

### 4.11.1 created

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

### 4.11.2 created_by

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

### 4.11.3 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).

### 4.11.4 resources

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

### 4.11.5 updates

This element contains a list of rstupdate 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.

### 4.11.6 phenopacket_schema_version

A string representing the version of the phenopacket-schema according to which a phenopacket was made.

### 4.11.7 external_references

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

## 4.12 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | a CURIE-style identifier e.g. HP:0001875 |
| label | string | required | human-readable class name e.g. Neutropenia |

**Example**

```
{
  "id": "HP:0001875",
  "label": "Neutropenia"
}
```

### 4.12.1 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*.

### 4.12.2 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*.

## 4.13 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). PED files are typically used by software for genetic linkage analysis. The phenopacket schema uses conventions similar to those of PED files 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.

The information in this element can be used 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| persons | list of *Person* | required | list of family members in this pedigree |

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

### 4.13.1 Person

Table 5: Definition of the `Person` element

| Field | Type | Status | Description |
|---|---|---|---|
| family_id | string | required | application specific identifier |
| individual_id | string | required | application specific identifier |
| paternal_id | string | required | application specific identifier |
| maternal_id | string | required | application specific identifier |
| sex | *Sex* | required | see text |
| affected_status | *AffectedStatus* | required | see text |

**Example**

```
{
   "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": "FEMALE",
           "affectedStatus": "AFFECTED"
       },
       {
           "familyId": "family 1",
           "individual_id": "MOTHER",
           "paternalId": "0",
           "maternalId": "0",
           "sex": "FEMALE",
           "affectedStatus": "UNAFFECTED"
       },
       {
           "familyId": "family 1",
           "individualId": "FATHER",
           "sex": "MALE",
           "paternalId": "0",
           "maternalId": "0",
           "affectedStatus": "UNAFFECTED"
       }
   ]
}
```

### 4.13.2 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.

The Pedigree object does not support reporting multiple phenotypes in one individual. The phenotype represented by the affectation status is whether the disease is present or not. If this is desired, then one would have to create full phenopackets for each individual in a family.

## 4.14 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 (negated), with possible onset, modifiers and frequency.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| description | string | optional | human-readable verbiage **NOT** for structured text |
| type | *OntologyClass* | required | |
| negated | boolean | optional | defaults to *false* |
| severity | *OntologyClass* | optional | description of the severity of the feature described in *type* representing HP:0012824 |
| modifier | list of *OntologyClass* | optional | representing one or more terms from HP:0012823 |
| onset | *OntologyClass* | optional | HP:0003674 HP:0011462 |
| evidence | *Evidence* | recommended | the evidence for an assertion of the observation of a *type* |

**Example**

```
{
    "type": {
      "id": "HP:0000520",
      "label": "Proptosis"
    },
    "severity": {
        id": "HP:0012825",
        "label": "Mild"
    }
    "classOfOnset": {
      "id": "HP:0003577",
      "label": "Congenital onset"
    }
}
```

## 4.14.1 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.

## 4.14.2 type

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

## 4.14.3 negated

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.

### 4.14.4 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

### 4.14.5 modifier

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).

### 4.14.6 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.

### 4.14.7 evidence

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

## 4.15 Procedure

The Procedure element represents a clinical procedure performed on a subject in order to extract a biosample.

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

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| code | *OntologyClass* | required | clinical procedure performed on a subject |
| body_site | *OntologyClass* | optional | specific body site if unable to represent this is the *code* |

**Example**

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

### 4.15.1 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"
    }
}
```

### 4.15.2 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"
    }
}
```

## 4.16 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | required | hp |
| name | string | required | human phenotype ontology |
| namespace_prefix | string | required | HP |
| url | string | required | http://purl.obolibrary.org/obo/hp.owl |
| version | string | required | 2018-03-08 |
| iri_prefix | string | required | http://purl.obolibrary.org/obo/**HP_** |

**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

```
{
    "id": "so",
    "name": "Sequence types and features",
    "url": "http://purl.obolibrary.org/obo/so.owl",
    "version": "2015-11-24",
    "namespacePrefix": "SO",
    "iriPrefix": "http://purl.obolibrary.org/obo/SO_"
}
```

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

```
{
  "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)

```
{
  "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

### 4.16.1 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.

### 4.16.2 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.

### 4.16.3 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

### 4.16.4 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.

### 4.16.5 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.

### 4.16.6 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.

### 4.16.7 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.

### 4.16.8 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, using the HPO term encoding the concept of 'Severe', using this instance of an OntologyClass:

```
{
  "id": "HP:0012828",
  "label": "Severe",
}
```

and this instance of a Resource:

```
{
    "id": "hp",
    "name": "Human Phenotype Ontology",
    "url": "http://purl.obolibrary.org/obo/hp.owl",
    "version": "17-06-2019",
    "namespacePrefix": "HP",
    "iriPrefix": "http://purl.obolibrary.org/obo/HP_"
}
```

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

## 4.17 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.

**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 |

**Example**

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

## 4.18 Variant

This element should be used to describe candidate variants or diagnosed causative variants. There is currently no standard variant nomenclature that can represent all kinds of genetic variation that is relevant to human medicine, science, and model organisms. Therefore, we represent variants using the keyword `oneof`, which is used in protobuf for an item with many optional fields where at most one field will be set at the same time. Variant messages contain an allele and the zygosity of the allele.

Alleles can be listed using HGVS, VCF, SPDI or ISCN notation. The phenopacket schema will implement the GA4GH Variation Representation Specification once that is mature. The VR-Spec will be the recommended option in some settings.

- See: https://vr-spec.readthedocs.io/en/1.0rc/
- See: https://github.com/ga4gh-beacon/specification/blob/master/beacon.yaml

The `Variant` element itself is an optional element of a `Phenopacket` or `Biosample`. If it is present, the Phenopacket standard has the following requirements.

Alleles can refer to external sources, 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*.

*n.b.* phase information for alleles are not represented in this model.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| allele | *allele* | required | one of the Allele types described below |
| zygosity | *OntologyClass* | recommended | See *zygosity* below |

**Example**

```
{
    "spdiAllele": {
      "id": "clinvar:13294"
      "seqId": "NC_000010.10",
      "position": 123256214,
      "deletedSequence": "T",
      "insertedSequence": "G"
    },
    "zygosity": {
      "id": "GENO:0000135",
      "label": "heterozygous"
    }
}
```

### 4.18.1 zygosity

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).

### 4.18.2 allele

The allele element is required and can be one and only one of `HgvsAllele`, `VcfAlelle`, `SpdiAllele` or `IcsnAllele`.

### 4.18.3 HgvsAllele

This element is used to describe an allele according to the nomenclature of the Human Geneome Variation Society (HGVS). For instance, `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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | recommended | An arbitrary identifier |
| hgvs | string | required | NM_000226.3:c.470T>G |

**Example**

```
{
    "id": "",
    "hgvs": "NM_000226.3:c.470T>G"
}
```

## 4.18.4 VcfAllele

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 `VcfAllele` 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 `variant` messages would be required).

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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| genome_assembly | string | required | GRCh38 |
| id | string | recommended | An arbitrary identifier |
| chr | string | required | chr2 |
| pos | int32 | required | 134327882 |
| re | string | required | A |
| alt | string | required | C |
| info | string | optional | END=43500;SVTYPE=DUP;CHR2=1;SVLEN=29000; |

**Example**

```
{
    "genome_assembly": "GRCh38",
    "chr": "2",
    "id": ".",
    "pos": 134327882,
    "ref": "A",
    "alt": "<DUP>",
    "info": "END=43500;SVTYPE=DUP;CHR2=1;SVLEN=29000;",
}
```

### 4.18.5 SpdiAllele

This option can be used as an alternative to the VcfAllele, and describes variants using the Sequence Position Deletion Insertion (SPDI) notation. 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 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | recommended | An arbitrary identifier |
| seq_id | string | required | Seq1 |
| position | int32 | required | 4 |
| deleted_sequence | string | required | A |
| inserted_sequence | string | required | G |

**Example**

```
{
    "id": 1,
    "seqId": "NC_000001.10",
    "position": 12346,
    "deletedSequence": "",
    "insertedSequence": "T"
}
```

### 4.18.6 IscnAllele

This element can be used to describe cytogenetic anomalies according to 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.

**Data model**

| Field | Type | Status | Description |
|---|---|---|---|
| id | string | recommended | An arbitrary identifier |
| iscn | string | required | t(8;9;11)(q12;p24;p12) |

**Example**

```
{
  "id": "ISCN:12345",
  "iscn": "t(8;9;11)(q12;p24;p12)"
}
```

CHAPTER 5

# 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.

## 5.1 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*).

### 5.1.1 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 three examples for *Rare Disease*, cancer, and model organism phenotypes. TODO.

#### Java Build

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

#### Setting up the Java build

To include the phenopackets-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 phenopackets-schema locally

Users can also download phenopackets-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.

```java
import com.google.protobuf.Timestamp;
import java.text.SimpleDateFormat;
import java.util.Date;

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
String hastings = "1066-10-14";
Date date = formatter.parse(hastings);
long millis = date.getTime();
Timestamp timestamp = Timestamp.newBuilder().setSeconds(millis / 1000)
          .setNanos((int) ((millis % 1000) * 1000000)).build();
```

If more precision is desired, use the following format

```java
SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
```

### 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 auxilliary function

```java
/** 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

```java
public Phenopacket spherocytosisExample() {
      final String PROBAND_ID = "PROBAND#1";
      final OntologyClass FEMALE = ontologyClass("PATO:0000383", "female");
      PhenotypicFeature spherocytosis = PhenotypicFeature.newBuilder()
              .setType(ontologyClass("HP:0004444", "Spherocytosis"))
              .setClassOfOnset(ontologyClass("HP:0011463", "Childhood onset"))
              .build();
      PhenotypicFeature jaundice = PhenotypicFeature.newBuilder()
              .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"))
              .setNegated(true)
              .build();
      PhenotypicFeature reticulocytosis = PhenotypicFeature.newBuilder()
              .setType(ontologyClass("HP:0001923", "Reticulocytosis"))
              .build();
```

(continues on next page)

(continued from previous page)

```
    Variant ANK1_variant = Variant.newBuilder()
            .setSequence("NM_001142446.1")
            .setPosition(5620)
            .setDeletion("C")
            .setInsertion("T")
            .setHgvs("NM_001142446.1:c.5620C>T ")
            .putSampleGenotypes(PROBAND_ID, ontologyClass("GENO:0000135",
↪"heterozygous"))
            .build();

    Individual proband = Individual.newBuilder()
            .setSex(FEMALE)
            .setId(PROBAND_ID)
            .setAgeAtCollection(Age.newBuilder().setAge("P27Y3M").build())
            .addPhenotypicFeatures(spherocytosis)
            .addPhenotypicFeatures(jaundice)
            .addPhenotypicFeatures(splenomegaly)
            .addPhenotypicFeatures(notHepatomegaly)
            .addPhenotypicFeatures(reticulocytosis)
            .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("pato")
                    .setName("Phenotype And Trait Ontology")
                    .setNamespacePrefix("PATO")
                    .setIriPrefix("http://purl.obolibrary.org/obo/PATO_")
                    .setUrl("http://purl.obolibrary.org/obo/pato.owl")
                    .setVersion("2018-03-28")
                    .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)
            .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();
}
```

## 5.1.2 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";
        exit(EXIT_FAILURE);
    }
    string fileName=argv[1];

    GOOGLE_PROTOBUF_VERIFY_VERSION;

    stringstream sstr;
    ifstream inFile;
    inFile.open(fileName);
    if (! inFile.good()) {
```

(continues on next page)

(continued from previous page)

```
        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

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 $<
```

(continues on next page)

(continued from previous page)

```
.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.

### 5.1.3 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/v1/examples` directory by a method called `printAsJson`.

#### 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*.

### 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.

#### 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",
```

#### 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",
        "ageAtCollection": {
            "age": "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"
  },
  "classOfOnset": {
    "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."
```

(continues on next page)

(continued from previous page)

```
    }
  }]
}
```

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"
  },
  "negated": 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"
  }],
  "ageOfOnset": {
    "age": "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.

```
  {
```

(continues on next page)

(continued from previous page)

```
    "type": {
      "id": "HP:0001270",
      "label": "Motor delay"
    },
    "severity": {
      "id": "HP:0012825",
      "label": "Mild"
    },
    "classOfOnset": {
      "id": "HP:0011463",
      "label": "Childhood onset"
    }
  }],
  "variants": [{
    "hgvsAllele": {
      "hgvs": "NM_001848.2:c.877G\u003eA"
    },
    "zygosity": {
      "id": "GENO:0000135",
      "label": "heterozygous"
    }
  }]
}
```

### 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"
  }
}],
```

### 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",
    "maternalId": "MOTHER",
    "sex": "MALE",
    "affectedStatus": "AFFECTED"
```

(continues on next page)

(continued from previous page)

```
  }, {
    "individualId": "MOTHER",
    "sex": "FEMALE",
    "affectedStatus": "UNAFFECTED"
  }, {
    "individualId": "FATHER",
    "sex": "MALE",
    "affectedStatus": "UNAFFECTED"
  }]
},
```

### metaData

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

```
"metaData": {
  "created": "2019-04-04T13:49:22.827Z",
  "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/"
  }],
  "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."
  }]
}
```

### 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).

### 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.

#### id

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

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

#### 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",
    "karyotypicSex": "UNKNOWN_KARYOTYPE"
},
```

#### 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"
   },
 }],
```

### 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": [ { ... }, { ... }, {....}],
```

### 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",
    "description": "",
    "sampledTissue": {
        "id": "UBERON_0001256",
        "label": "wall of urinary bladder"
    },
    "ageOfIndividualAtCollection": {
        "age": "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"
        }
    },
    "htsFiles": [{
      "uri": "file://data/genomes/urothelial_ca_wgs.vcf.gz",
      "description": "Urothelial carcinoma sample",
      "htsFormat": "VCF",
      "genomeAssembly": "GRCh38",
      "individualToSampleIdentifiers": {
        "sample1": "BS342730"
      }
    }],
    "isControlSample": false
}
```

### 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"
    },
    "ageOfIndividualAtCollection": {
        "age": "P52Y2M"
    },
    "histologicalDiagnosis": {
        "id": "NCIT:C5596",
        "label": "Prostate Acinar Adenocarcinoma"
    },
    "tumorProgression": {
        "id": "NCIT:C95606",
        "label": "Second Primary Malignant Neoplasm"
    },
    "tumorGrade": {
        "id": "NCIT:C28091",
        "label": "Gleason Score 7"
    },
    "procedure": {
        "code": {
            "id": "NCIT:C15189",
            "label": "Biopsy"
        }
    },
    "isControlSample": false
}
```

### Biosample 3: Left ureter

A biopsy of the left ureter reveal normal findings.

```
{
    "id": "sample3",
    "individualId": "patient1",
    "sampledTissue": {
        "id": "UBERON:0001223",
        "label": "left ureter"
    },
    "ageOfIndividualAtCollection": {
        "age": "P52Y2M"
    },
    "histologicalDiagnosis": {
        "id": "NCIT:C38757",
        "label": "Negative Finding"
    },
    "procedure": {
        "code": {
            "id": "NCIT:C15189",
```

(continues on next page)

(continued from previous page)

```
            "label": "Biopsy"
        }
    },
    "isControlSample": false
}
```

### Biosample 4: Right ureter

A biopsy of the right ureter reveal normal findings.

```
{
    "id": "sample4",
    "individualId": "patient1",
    "sampledTissue": {
        "id": "UBERON:0001222",
        "label": "right ureter"
    },
    "ageOfIndividualAtCollection": {
        "age": "P52Y2M"
    },
    "histologicalDiagnosis": {
        "id": "NCIT:C38757",
        "label": "Negative Finding"
    },
    "procedure": {
        "code": {
            "id": "NCIT:C15189",
            "label": "Biopsy"
        }
    },
    "isControlSample": false
}
```

### 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",
   "sampledTissue": {
       "id": "UBERON:0015876",
       "label": "pelvic lymph node"
   },
   "ageOfIndividualAtCollection": {
       "age": "P52Y2M"
   },
   "tumorProgression": {
       "id": "NCIT:C3261",
       "label": "Metastatic Neoplasm"
   },
   "procedure": {
       "code": {
```

(continues on next page)

(continued from previous page)

```
            "id": "NCIT:C15189",
            "label": "Biopsy"
        }
    },
    "htsFiles": [{
      "uri": "file://data/genomes/metastasis_wgs.vcf.gz",
      "description": "lymph node metastasis sample",
      "htsFormat": "VCF",
      "genomeAssembly": "GRCh38",
      "individualToSampleIdentifiers": {
        "sample5": "BS730275"
      }
    }],
    "isControlSample": false
}
```

### 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).

### 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"
    }],
    "tnmFinding": [{
      "id": "NCIT:C48766",
      "label": "pT2b Stage Finding"
    }, {
      "id": "NCIT:C48750",
      "label": "pN2 Stage Finding"
    }, {
      "id": "NCIT:C48700",
      "label": "M1 Stage Finding"
    }]
  }]
}
```

### 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": {
          "patient1": "NA12345"
        }
    }],
}
```

### metaData

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

```
{
    "metaData": {
         "created": "2019-04-03T15:31:40.765Z",
         "createdBy": "Peter R",
         "submittedBy": "Peter R",
         "resources": [{
             "id": "hp",
             "name": "human phenotype ontology",
             "namespacePrefix": "HP",
             "url": "http://purl.obolibrary.org/obo/hp.owl",
             "version": "2019-04-08",
             "iriPrefix": "http://purl.obolibrary.org/obo/HP_"
             }, {
             "id": "uberon",
             "name": "uber anatomy ontology",
             "namespacePrefix": "UBERON",
             "url": "http://purl.obolibrary.org/obo/uberon.owl",
             "version": "2019-03-08",
             "iriPrefix": "http://purl.obolibrary.org/obo/UBERON_"
             }, {
             "id": "ncit",
             "name": "NCI Thesaurus OBO Edition",
             "namespacePrefix": "NCIT",
             "url": "http://purl.obolibrary.org/obo/ncit.owl",
             "version": "18.05d",
             "iriPrefix": ""
             }],
         "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*.

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

### 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`.

### 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 Ontology-Class 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.

### Family members and variants

We define the names of the family members and also an object to represent the variant that was found to occur in a de novo fashion in the son.

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

// Allele
private static final HgvsAllele c_877G_to_A = HgvsAllele.
        newBuilder().
        setHgvs("NM_001848.2:c.877G>A").
        build();
// Corresponding variant
private static final Variant heterozygousCOL6A1Variant = Variant.newBuilder()
        .setHgvsAllele(c_877G_to_A)
        .setZygosity(ontologyClass("GENO:0000135", "heterozygous"))
        .build();
```

### 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().
                        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"))
                .setClassOfOnset(ontologyClass("HP:0011461", "Fetal onset"))
                .addEvidence(citation)
                .build();
        PhenotypicFeature absentCranialNerveAbnormality = PhenotypicFeature.
↪newBuilder()
                .setType(ontologyClass("HP:0031910", "Abnormal cranial nerve physiology
↪"))
                .setAbsent(true)
                .addEvidence(citation)
                .build();
        PhenotypicFeature motorDelay = PhenotypicFeature.newBuilder()
                .setType(ontologyClass("HP:0001270","Motor delay"))
                .setClassOfOnset(ontologyClass("HP:0011463","Childhood onset"))
                .setSeverity(mild)
                .build();
        PhenotypicFeature hematuria = PhenotypicFeature.newBuilder()
                .setType(ontologyClass("HP:0011463", "Macroscopic hematuria"))
                .setAgeOfOnset(Age.newBuilder().setAge("P14Y").build())
                .addModifiers(ontologyClass("HP:0031796","Recurrent"))
                .addEvidence(citation)
                .build();

        Individual proband = Individual.newBuilder()
                .setSex(Sex.MALE)
                .setId(PROBAND_ID)
                .setAgeAtCollection(Age.newBuilder().setAge("P14Y").build())
                .build();
        return Phenopacket.newBuilder()
                .setId(PROBAND_ID)
                .setSubject(proband)
                .addPhenotypicFeatures(decreasedFetalMovement)
                .addPhenotypicFeatures(absentCranialNerveAbnormality)
                .addPhenotypicFeatures(hematuria)
                .addPhenotypicFeatures(motorDelay)
                .addVariants(heterozygousCOL6A1Variant)
```

(continues on next page)

(continued from previous page)

```
                .build();
    }
```

## 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)
```

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

## 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();
    }
```

### Family

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

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

### A complete example in Java: Oncology

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

### subject

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

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

### 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();
```

### HtsFile

We use three *HtsFile* 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 HtsFile createSomaticHtsFile() {
    // first create a File
    // We are imagining there is a reference to a VCF file for a normal germline␣
↪genome seqeunce
```

(continues on next page)

(continued from previous page)

```
    String path = "/data/genomes/urothelial_ca_wgs.vcf.gz";
    String description = "Urothelial carcinoma sample";
    File file = File.newBuilder().setPath(path).setDescription(description).build();
    // Now create an HtsFile object
    return HtsFile.newBuilder().
            setHtsFormat(HtsFile.HtsFormat.VCF).
            setGenomeAssembly("GRCh38").
            setFile(file).
            build();
}
```

### 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)
      OntologyClass infiltratingUrothelialCarcinoma = ontologyClass("NCIT:C39853",
↪"Infiltrating Urothelial Carcinoma");
      biosampleBuilder.setHistologicalDiagnosis(infiltratingUrothelialCarcinoma);
      // A malignant tumor at the original site of growth
      OntologyClass primary = ontologyClass("NCIT:C84509", "Primary Malignant Neoplasm
↪");
      biosampleBuilder.setTumorProgression(primary);
      biosampleBuilder.addHtsFiles(HtsFileTest.createSomaticHtsFile());
      biosampleBuilder.setProcedure(Procedure.newBuilder().setCode(ontologyClass(
↪"NCIT:C15189", "Biopsy")).build());
      return biosampleBuilder.build();
  }
```

### 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");
```

(continues on next page)

(continued from previous page)

```
    biosampleBuilder.setHistologicalDiagnosis(normalFinding);
    biosampleBuilder.setProcedure(Procedure.newBuilder().setCode(ontologyClass(
↪"NCIT:C15189", "Biopsy")).build());
    return biosampleBuilder.build();
}
```

### 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)
        .addTnmFinding(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)
        .addTnmFinding(ontologyClass("NCIT:C48750", "pN2 Stage Finding"))
        // M1 Stage Finding
        // the tumour has spread from the original site (Metastatic Neoplasm in␣
↪lymph node - sample5)
        .addTnmFinding(ontologyClass("NCIT:C48700", "M1 Stage Finding"))
        .build();
}
```

### 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")
```

(continues on next page)

(continued from previous page)

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

## 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())
    .addPhenotypes(hematuria)
    .addPhenotypes(dsyuria)
    .addBiosamples(bladderBiosample())
    .addBiosamples(prostateBiosample())
    .addBiosamples(leftUreterBiosample())
    .addBiosamples(rightUreterBiosample())
    .addBiosamples(pelvicLymphNodeBiosample())
    .addDiseases(infiltratingUrothelialCarcinoma())
    .addHtsFiles(createNormalGermlineHtsFile())
    .setMetaData(metaData)
    .build();
```

## 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 including empty fields.

```
System.out.println(JsonFormat.printer().includingDefaultValueFields().
↪print(phenopacket));
```

## 5.2 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.