BioCompute Objects

Specification Document

BioCompute Object (BCO) specification document

Standard Trial Use (STU) Release 1.0

BioCompute Object Consortium members (BCOC):

FDA: Vahan Simonyan, Mark Walderhaug, Ruth Bandler, Eric Donaldson, Elaine Thompson, Alin Voskanian, Anton Golikov, Konstantinos Karagiannis, Elaine Johanson, Adrian Myers, Errol Strain, Khaled Bouri, Tong Weida, Wenming Xiao, Md Shamsuzzaman
GW: Raja Mazumder, Charles Hadley S. King IV, Amanda Bell, Jeet Vora, Krista M. Smith, Robel Kahsay
Community: Gil Alterovitz (Boston Children's Hospital/Harvard Medical School, SMART/FHIR/HL7, GA4GH), Michael R. Crusoe (CWL), Marco Schito (C-Path), Konstantinos Krampis (CUNY), Alexander (Sasha) Wait Zaranek (Curoverse), John Quackenbush (DFCI/Harvard), Geet Duggal (DNAnexus), Singer Ma (DNAnexus), Yuching Lai (DDL), Warren Kibbe (Duke), Tony Burdett (EBI), Helen Parkinson (EBI), Stuart Young (Engility Corp), Anupama Joshi (Epinomics), Vineeta Agarwala (Flatiron Health), James Hirmas (GenomeNext), David Steinberg (UCSC), Veronica Miller (HIV Forum), Dan Taylor (Internet 2), Paul Duncan (Merck), Jianchao Yao (Merck & Co., Inc., Boston, MA USA), Marilyn Matz (Paradigm4), Ben Busby (NCBI), Eugene Yaschenko (NCBI), Zhining Wang (NCI), Hsinyi (Steve) Tsang (NCI), Durga Addepalli (NCI/Attain), Heidi Sofia (NIH), Scott Jackson (NIST), Paul Walsh (NSilico Life Science), Toby Bloom (NYGC), Jeremy Goecks (Oregon Health and Science University), Srikanth Gottipati (Otsuka-US), Alex Poliakov (Paradigm4), Keith Nangle (Pistoia Alliance), Jonas S Almeida (Stony Brook Univ, SUNY), Dennis A. Dean, II (Seven Bridges Genomics), Dustin Holloway (Seven Bridges Genomics), Nisha Agarwal (Solvuu), Stian Soiland-Reyes (UNIMAN), Carole Goble (UNIMAN), Melanie Price (UNIMAN), Susanna-Assunta Sansone (University of Oxford), Philippe Rocca-Serra (University of Oxford), Phil Bourne (Univ. of Virginia)

High-throughput Sequencing Computational Standards for Regulatory Sciences (HTS-CSRS) Project

Contact: Raja Mazumder (mazumder@gwu.edu) and Vahan Simonyan (vahan.simonyan@fda.hhs.gov)

Table of Contents

1 Introduction

BioCompute is a paradigm and a BioCompute Object (BCO) is an instance of that paradigm. High-throughput sequencing (HTS), also referred to as next-generation sequencing (NGS) or massively parallel sequencing (MPS), has increased the pace at which we generate, compute and share genomic data in biomedical sciences. As a result, scientists, clinicians and regulators are now faced with a new data paradigm that is less portable, more complex and most of all poorly standardized. BCOs use a simple JSON format to encode important information on the execution of computational pipelines. The goal of using a BCO is to streamline communication of these details between stakeholders in academia, industry and regulatory agencies.

The US Food and Drug Administration (FDA) and George Washington University (GW) have partnered to establish a framework for community-based standards development and harmonization of HTS computations and data formats. Standardized HTS data processing and data formats will promote interoperability and simplify the verification of bioinformatics protocols. To do this, a schema has been developed to represent instances of computational analysis as a BCO. A BCO includes:

- Information about parameters and versions of the executable programs in a pipeline
- Reference to input and output test data for verification of the pipeline
- A usability domain
- Keywords
- A list of authors along with other important metadata

In addition to all the information captured in the BCO, the BCO itself must be independent of the execution environment, whether it is a local high-performance or a cloud-based infrastructure.

Additional, non-normative, information on BCOs:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5510742/

1.1 Mission of the BioCompute project

- BioCompute Objects will facilitate communicating HTS computational analysis details with the FDA.
- Develop a community of stakeholders to create a versatile data harmonization framework that allows the standardized definition of platform-independent bioinformatics pipelines for execution, and is easily read by humans AND machines.
- Facilitate the development of tools and facilities implementing data typing, instantiation, deposition, storage, and distribution of validated BioCompute Objects through a BioCompute database, in order to enable reproducible scientific research and regulatory submissions of data and computations.
- Facilitate portability of pipelines for execution on Public Cloud infrastructure.

1.2 Motivation

The unpredictability of tangible physical, chemical, and biological experiments due to the multitude of environmental and procedural factors is well documented. What is often systematically overlooked is that computational biology algorithms are affected by a multiplicity of parameters and are no less volatile. The complexities of computation protocols and interpretation of outcomes is only part of the challenge; there are also virtually no standardized and industry-accepted metadata schemas for reporting the computational pipelines and parameters together with their results. Thus, it is often impossible to reproduce the results of a previously performed computation due to missing information on parameters, versions, arguments, conditions, and procedures of application launch. The BCO concept has been developed specifically to satisfy regulatory research needs for evaluation, validation, and verification of bioinformatics pipelines; however, there is potential utility of BCO within the larger scientific community. This utility can be increased through the creation of a BCO database comprising records relevant to the U.S. Food and Drug Administration.
A BioCompute Object database record will be similar to a GenBank record in form; however, instead of describing a sequence, the BioCompute record will include information related to parameters, dependencies, usage, and other information related to the specific computational instance. This mechanism will extend similar efforts and also serve as a collaborative ground to ensure interoperability between different platforms, industries, scientists, regulators, and other stakeholders interested in biocomputing.

For more information, see the project description on the **FDA Extramural Research** page.

1.2.1 Limitations of the initial effort

- At the initial stages of BioCompute development, we address the challenges of HTS (NGS) bioinformatics.
- BCOs could very easily be extended to other types of computational analysis, but at this stage, we are limiting our focus to HTS/NGS only.

1.3 Audience for this document

- Users performing HTS analysis with a regulatory science perspective
- HTS Platform Developers
- HTS related standard developers

1.4 Potential Stakeholders for the entire BioCompute project

- US Food and Drug Administration, as well as other Regulatory Agencies
- Medical product manufacturers and their suppliers
- Laboratories developing clinical testing protocols

- Bioinformatics tool and platform developers who wish to operate in a regulatory environment, including cloud service (PaaS, IaaS, SaaS, FaaS) providers
- Journals / Scientific Publishing / peer reviewing process
- US National Institutes of Health (NIH) (particularly initiatives such as NCI/ITCR)
- Public cloud companies operating in the Life Sciences sector including electronic health record (EHR) systems

1.5 BCO User stories

- **Reproducibility and Interpretation use case:**
  A pharmaceutical company is submitting NGS data and the FDA conducts a reanalysis of the data. The reanalysis does not concur with the original results. It can be very lengthy and costly to figure out the location of the discrepancies. Attaching a BioCompute Object with the initial submission would prevent most of the ambiguity surrounding the discrepancies.

- **Reusability use case:**
  A regulatory decision has been made where a computational analysis has been used as evidence. New data emerges after the product has been on the market over a year and the regulators cannot reproduce the original environment with the configuration of tools and parameters of pipelines to reanalyze the initial submission data or replicate the initial conclusion.

- **Collaboration use case:**
  Authors and pharmaceutical scientists are unaware of how the regulatory industry is using workflows to analyze data. Openness and transparency are hindered by the lack of ability to communicate, not a lack of willingness. Scientific merit is compromised as a result of not having a common "language" for communicating computations.

- **Accountability use case:**
  A bioinformatics platform provider can use BCO as part of its verification and validation process. A customer submits NGS data provided by a third party sequencing provider. The sequencing data is poor quality. Reproducible pipelines, validated and verified as a “BCO”, were used to demonstrate the fault lies in the sequencing step and not the bioinformatics pipeline.

- **Versioning use case:**
  One potential use case related to this is one of 'differential impact' of how different choices in the workflow affect the outcome of the computational analysis/experiment (e.g. changing expression estimation procedure).

- **Provenance use case:**
  BCOs can serve as a history of what was computed. An example pertaining to provenance, from experience: data are generated and QC'ed as far as possible, and then passed on for analysis. The analysis diagnoses a problem with one or more samples (e.g., cryptic relatedness), which are then locally excluded from the analysis. But that exclusion is not reflected back to the original data, and the same bad samples are included in the next analysis. In this way, a record exists of which samples can be excluded in future analysis.

1.6 BCO community

The BioCompute Object working group facilitates a means for different stakeholders in the HTS communities to provide input on current practices on the BCO. This working group was formed during preparation for the 2017 HTS Computational Standards for Regulatory Sciences Workshop, and was initially made up of the workshop participants, both speakers and panelists. There has been a continual growth of the BCO working group as a direct result of the interaction between a variety of stakeholders from all interested communities in standardization of computational HTS data processing. The Public-Private partnerships formed between universities, private genomic data companies, software platforms, government and regulatory institutions has been an easy point of entry for new individuals or institutions into the BCO project to participate in the discussion of best practices for the objects.

2 Data type for BCOs

The fundamentals of data typing (type primitives, class inheritance, etc.) that are used to define BioCompute Objects are described in detail in section Appendix VI. Developers of BCO enabled platforms should reference this section for details on how to support the creation of BCO programmatically or manually. BCOs are represented in JSON (JavaScript Object Notation) formatted text. The JSON format was chosen because it is both human and machine readable/writable. For a detailed description of JSON see www.json.org.

BioCompute data types are defined as aggregates of the critical fields organized into a few domains: the descriptive domain, the identification and provenance domain, the input and output domains, the parametric domain, the environmental domain, the execution domain, the prerequisite domain, the usability domain, and the error domain. At the time of submission to the BioCompute Object database an instance of BCO type is created, populated with actual values compliant with the data type definitions and assigned a unique identifier. The object could then be assigned a unique digital signature and a unique digital object identifier. (See security section, Appendix V.)

Three of the domains in a BioCompute Object become immutable upon assignment of the digital signature: 1) the Parametric Domain, 2) the Execution Domain and 3) the I/O Domain. Changing anything within these domains invalidates the verification and will break the digital signature.

Optional fields are indicated by the `"vital": "True"` flag, which is shown in the data typing section below (Appendix VI).

2.1 Identification and Provenance fields

2.1.1 ID "id"

A namespaced (source/db/id) unique identifier of this BCO instance assigned by a BCO database engine. IDs should never be deprecated or reused.

```
"id": "https://hive.biochemistry.gwu.edu/bco_db/obj.1270"
```

2.1.2 Name "name"

Name for the BCO. This public field should take free text value using common biological research terminology along with external reference linkage using identifiers whenever possible.

```
"name": "HCV1a [taxonomy:31646] ledipasvir [pubchem.compound:67505836]
resistance SNP [so:0000694] detection"
```

2.1.3 Structured name "structured_name"

Structured name is an optional templated computable text field designed to represent a BCO instance name in visible interfaces. This field can refer to other fields within the same or other objects. For example, a string like "HCV1a [taxonomy:$taxonomy] mutation detection" will be visualized as "HCV1a [taxonomy:31646] mutation detection" assuming the BCO has a field called taxonomy and value 31646.

```
HCV1a [taxonomy:$taxonomy] mutation detection = HCV1a [taxonomy:31646] mutation
detection
```

2.1.4 Version "version"

Records the versioning of this BCO instance object. In BCO versioning, a change in the BCO affecting the outcome of the computation should be deposited as a new BCO, not as a new version. If a parameter in a tool is changed within a BCO, which in turn changes the outcome of the pipeline and the original BCO, a new BCO, not a new version, will be created.
In such cases the connection between the new object and the older one may or may not be (on author's discretion) retained in the form of references. Changes that cannot affect the results of the computation can be incorporated into a new version of the existing BCO. Such changes might include name and title, comments, authors, validity dates, etc.

```
"version": "1.1"
```

2.1.5 Digital signature "digital_signature"

A string, read-only value generated by a BCO database, protecting the object from internal or external alterations without proper validation. This value should not be submitted during deposition but can be read during downloading or transferring validated BCOs. The BCO server can provide an API validating the signature versus BCO content, allowing users to validate the signature "offline" on their own. The server will also be able to provide a reference to the signature creation algorithm, allowing for greater interoperability.

```
"digital_signature": "905d7fce3f3ac64c8ea86f058ca71658"
```

2.1.6 Verification status "verification_status"

Describes the status of an object in the verification process. The 'unreviewed' flag indicates that the object has been submitted, but no further evaluation or verification has occurred. The 'in_progress' flag indicates that verification is underway. The 'reviewed' flag indicates that the BCO has been verified and reviewed. The 'manual' flag indicates that the object has been manually verified. The 'suspended' flag indicates an object that was once valid is no longer considered valid. The 'error' flag indicates that an error was detected in the BCO.

```
"verification_status": "in_progress" OR "unreviewed" OR "reviewed" OR
"published" OR "rejected"
```

2.1.7 Publication status "publication_status"

This is a choice field with three options. The 'draft' status indicates that an object is in draft form and is still being edited. The 'open_access' status indicates that an object has been published and is freely available to anyone. The objects with the 'private' status have restrictions on who can view and access them. This is a way for researchers using restricted data or metadata to ensure the confidentiality is maintained. The permissions to the object will be defined by the database access control rules.

```
"publication_status": "draft" OR "in_progress" OR "private" OR "open_access"
```

2.1.8 Authors "authors"

This is a list to hold author identifiers. We encourage the use of ORCIDs to record author information, as they allow for the author to curate their information after submission. If an author does not have an ORCID then they can provide a name using free unicode text.

```
"authors":["Charles Darwin", "http://orcid.org/0000-0000-0000-0000"]
```

2.2 Usability domain "usability_domain"

This field provides a space for the author to define the usability domain of the BCO. It is an array of free text values. This field is to aid in search-ability and provide a specific description of the object. The usability domain along with keywords can help determine when and how the BCO can be used. Novel use of the BCO could result in the creation of a new entry with a new usability domain.

```
"Identify baseline single nucleotide polymorphisms SNPs [so:0000694],
insertions [so:0000667], and deletions [so:0000045] that correlate with reduced
ledipasvir [pubchem.compound:67505836] antiviral drug efficacy in Hepatitis C
virus subtype 1 [taxonomy:31646]"
"GitHub CWL example: https://github.com/mr-c/hive-cwl-
examples/blob/master/workflow/hive-viral-mutation-detection.cwl#L20"
```

2.3 Description Domain "description_domain"

Structured field for description of external references, the pipeline steps, and the relationship of I/O objects. Information in this domain is not used for computation. This domain is meant to capture information that is currently being provided in FDA submission in journal format. It is possible that in the future this field can be semi-automatically generated from the `execution_domain` information.

2.3.1 Keywords "keywords"

This is an array field to hold a list of keywords to aid in search-ability and description of the object.

```
"keywords": ["antiviral resistance", "SNP", "Ledipasvir", "HCV1a", "amino acid
substitution"]
```

2.3.2 External References "xref"

This field contains a list of the databases and/or ontology IDs that are cross-referenced in the BCO. The external references are used to provide more specificity in the information related to BCO entries. Cross-referenced resources need to be available in the public domain. The external references are stored in the form of prefixed identifiers (CURIEs). These CURIEs map directly to the URIs maintained by identifiers.org. See Appendix-II for a list of the CURIEs used in this example.

```
"xref": ["so:0000694", "so:0000667", "so:0000045", "pubchem.compound:67505836",
"so:0000048", "taxonomy:31646", "pubmed:25123381", "pubmed:26508693"]
```

2.3.2.1 Extension to External References: SMART on FHIR Genomics

The external references include an optional extension to FHIR resource where specific data elements can be extracted from EHR systems or other secure FHIR endpoints via technologies such as SMART on FHIR Genomics (https://www.ncbi.nlm.nih.gov/pubmed/26198304) without

compromising patient and providers' information. This is because the portions being transferred contain no identifiable information about the patient. Instead there is a reference to the actual resource instance (via FHIR URL) through which all data is accessed.

The FHIR Endpoint URL coupled with the specific resource type and a unique FHIR identifier leads to a resource that can contain everything from the date and time of the procedure, specimen details, sequence information, linked sequence repositories, associated pedigrees, or even a set of observations linked from diagnostic reports. The link to FHIR can also be added to the usability domain. More on FHIR Genomics in release 3 of FHIR can be found here: https://www.hl7.org/fhir/genomics.html

SMART on FHIR Genomics provides a framework for EHR-based apps built on FHIR that integrate clinical and genomic information. For more information on how to use the SMART on FHIR Genomics apps, please visit http://projects.iq.harvard.edu/smartgenomics/.


```
{
	"FHIR_extension": [
		{
			"FHIRendpoint_Resource": "sequence",
			"FHIRendpoint_URL": "http://fhirtest.uhn.ca/baseDstu3",
			"FHIRendpoint_Ids": ["21376"]
		},
		{
			"FHIRendpoint_Resource": "diagnostics-genetics",
			"FHIRendpoint_URL": "http://fhirtest.uhn.ca/baseDstu3",
			"FHIRendpoint_Ids": ["43135"]
		},
		{
			"FHIRendpoint_Resource": "procedurerequest-genetics",
			"FHIRendpoint_URL": "http://fhirtest.uhn.ca/baseDstu3",
			"FHIRendpoint_Ids": ["25544"]
		},
		{
			"FHIRendpoint_Resource": "observation-genetics",
			"FHIRendpoint_URL": "http://fhirtest.uhn.ca/baseDstu3",
			"FHIRendpoint_Ids": ["92440"]
		},
		{
			"FHIRendpoint_ResourceType": "familymemberhistory-genetics",
			"FHIRendpoint_URL": "http://fhirtest.uhn.ca/baseDstu3",
			"FHIRendpoint_Ids": ["39025"]
		}
	]
}
```

2.3.2.2 Extension to External References: GitHub

The external references also include an extension to GitHub repositories where HTS computational analysis pipelines, workflows, protocols, and tool or software source code can be stored/deposited/downloaded. The BCO would contain link to the GitHub repository where the information is stored and easily retrieved. The links to GitHub can be added to the usability domain.

```
{
    "github_extension": {
        "github_repository": "hive-cwl-examples",
        "github_url": "https://github.com/common-workflow-language/hive-cwl-
examples",
        "github_uri": ["workflow/hive-viral-mutation-detection.cwl"]
    }
}
```

2.3.3 Pipeline tools "pipeline_steps"

This is an optional structured domain for recording the specifics of a pipeline. Each individual tool (or a well defined and reusable script) is represented as step, at the discretion of the author. Parallel processes are given the same step number. URIs are listed for the inputs and outputs for each tool.

2.3.3.1 Tool Name "tool_name"

Name for the specific tool. This field is a string (A-z, 0-1) and should be a single uniquely identifying word for the tool.

```
"tool_name": "HIVE-hexagon"
```

2.3.3.2 Tool Description "tool_desc"

A free text field for describing the specific use/purpose of the tool.

```
"tool_desc": "Alignment of reads to a set of references",
```

2.3.3.3 Tool Version "tool_version"

The version assigned to the instance of the tool used.

```
"tool_version": "1.3",
```

2.3.3.4 Tool Requirements or Packages "tool_package"

A list of text values to indicate any packages or prerequisites for running the tool used.

```
"tool_package": ["Python version 2.6", "megablast version 1.2"]
```

2.3.3.5 Step Number "step_number"

This is an integer value representing the position of the tool in the one-dimensional representation of the pipeline. Parallel computations are assigned the same number.

```
"step_number": "1"
```

2.3.3.6 Input URI List "input_uri_list"

Each tool lists the URIs of the input files. This is divided into two optionally segregated subdomains. The “input_files“ subdomain is a catchall for read files, or any other type of input file. The optional “reference_files” subdomain is specifically for recording which instance of a reference is used in the computation, and so has a field for the “name” and for the “URI”.

```
{
        "input_uri": {
            "reference_files": [
                {"name": "Hepatitis C virus genotype 1",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/22129792"},
                {"name": "Hepatitis C virus type 1b complete genome",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/5420376"},
                {"name": "Hepatitis C virus (isolate JFH-1) genomic RNA",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/13122261"},
                {"name": "Hepatitis C virus clone J8CF, complete genome",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/386646758"},
                {"name": "Hepatitis C virus S52 polyprotein gene",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/295311559"}
            ],
            "input_file_list": [
                "hive://nuc-read/514683",
                "hive://nuc-read/514682",
                "hive://data/514769/dnaAccessionBased.csv"
            ]
 },
```

2.3.3.7 Output URI List "output_uri_list"

Each tool lists the URI of the output files for that tool.

```
"output_uri_list": [
    "hive://data/514769/allCount-aligned.csv"
    "hive://data/514801/SNPProfile.csv",
    "hive://data/14769/allCount-aligned.csv"
]
```

2.4 Execution domain "execution_domain"

The fields required for execution of the BCO have been encapsulated together in order to clearly separate information needed for deployment, software configuration and running applications in a dependent environment. One byproduct of an accurate BCO definition is facilitation of reproducibility as defined by the *Oxford English Dictionary* as "the extent to which consistent results are obtained when produced repeatedly."

2.4.1 Script Type "script_type"

This is a choice field to indicate whether the "script" to execute the BioCompute Object is a reference to an external file (URI) or text in the "script" field.

```
"script_type": "URI" OR "text"
```

2.4.2 Script "script"

The Script field points to an internal or external reference to a script object that was used to perform computations for this BCO instance. This may be a reference to Galaxy Project or Seven Bridges Genomics pipeline, a Common Workflow Language (CWL) object in GitHub, a High-performance Integrated Virtual Environment (HIVE) computational service or any other type of script.

```
"script": "https://hive/workflows/antiviral_resistance_detection_hive.py"
```

2.4.3 Pipeline Version "pipeline_version"

This field records the version of the pipeline implementation.

```
"pipeline_version": "2.0"
```

2.4.4 Platform/Environment "platform"

The multi-value reference to a particular deployment of an existing platform where this BCO can be reproduced. A platform can be a bioinformatic platform such as Galaxy or HIVE or it can be a software package such as CASAVA or apps that includes multiple algorithms and software.

```
"platform": "HIVE"
```

2.4.5 Script driver "script_driver"

The reference to an executable that can be launched in order to perform a sequence of commands described in the script (see above) in order to run the pipeline. For example, if the pipeline is driven by a HIVE script, the script driver is the "hive" execution engine. For CWL based scripts, the cwl-execution tool, such as cwltool or other implementations like rabix(rabix.io), is the driver which is capable of interpreting the commands in the workflow. Another very general script driver commonly

used in Linux based operating systems is "shell" and the type of scripts it can run are operating system shell scripts. The combination of script driver and script is a capability to run a particular sequence of computational steps in order to produce BCO outputs given the inputs and parameters.

It is noteworthy to mention that scripts and script drivers by themselves can be objects. These objects can exist in internal (BCO) or external databases and be publically or privately accessible.

```
"driver": "shell"
```

2.4.6 Algorithmic tools and Software Prerequisites "software_prerequisites"

An optional multi-value field listing the minimal necessary prerequisites, library, tool versions needed to successfully run the script to produce BCO.

```
 "software_prerequisites": [
    {"name": "HIVE_hexagon", "version": "1.3"},
    {"name":"HIVE_heptagon","version":"1.3"}
]
```

2.4.7 Domain Prerequisites "domain_prerequisites"

An optional multi-value field listing the minimal necessary domain specific external data source access in order to successfully run the script to produce BCO.

```
"domain_prerequisites": [
    {"url": "protocol://domain:port/application/path","Name": "generic name"},
    {"url": "ftp://:22/",
    "Name": "access to ftp"},
    {"url": "http://eutils.ncbi.nlm.nih.gov/entrez/eutils",
    "Name": "access to e-utils"}
]
```

2.4.8 Environmental parameters "env_parameters"

Multi-value additional key value pairs useful to configure the execution environment on the target platform. For example, one might specify the number of compute cores, or available memory use of the script.

```
"env_parameters": [{"OSTYPE":"linux"}, {"QPRIDE_BIN":"~qpride/bin"}]
```

2.5 Parametric domain "parametric_domain"

This represents the list of parameters customizing the computational flow which can affect the output of the calculations. These fields are custom to each type of analysis and are tied to a particular pipeline implementation. All BCOs should inherit from the fundamental BioCompute data

type and as such inherit all of the core fields described in document. Specific BioCompute types introduce specific fields designed to customize the use of pipelines for a particular use pattern. Please refer to documentation of individual scripts and specific BCO descriptions for details.

```
"parametric_domain": {
        "heptagon_divergence_threshold_percent": "30",
        "hexagon_minimum_coverage": "0.15",
        "hexagon_seed": "14",
        "heptagon_freq_cutoff": "0.10",
        "hexagon_minimum_match_len": "66"
    },
```

2.6 Input and output domains "io_domain"

This represents the list of global input and output files created by the computational workflow, excluding the intermediate files. These fields are pointers to objects that can reside in the system performing the computation or any other accessible system. Just like the fields of parametric domain, these fields are custom to every specific BCO implementation and can refer to named input output arguments of underlying pipelines. Please refer to documentation of individual scripts and specific BCO descriptions for further details.

2.6.1 Input Subdomain "input_subdomain"

This field records the references and input files for the entire pipeline. Each type of input file is listed under a key for that type. The file types are specified when the BCO type is created. This allows the author to be very specific about a particular type of input file, if they so choose. For example: reference files have common names, and adding the common name here, in addition to the uri would make this more readable and understandable (eg, "HCV reference version..." or "human reference GRCH38")

```
"input_subdomain": {
    "HCV_Complete_Genome_type1": [
        "http://www.ncbi.nlm.nih.gov/nuccore/22129792",
        "http://www.ncbi.nlm.nih.gov/nuccore/5420376"
        ],
    "read_files": [
        "hive://nuc-read/514683",
        "hive://nuc-read/514682"
    ]
}
```

2.6.2 Output Subdomain "output_uri_list"

This field records the outputs for the entire pipeline. Each file should be an object with a key, and a title, URI, and mime-type (https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types) value.

```
"output_subdomain": {
    "hit_list ": {
        "title": "hit list",
        "uri": "hive://data/514769/dnaAccessionBased.csv",
        "mime-type" : "text/csv"
    },
    "mutation_profile": {
        "title": "mutation profile",
        "uri": "hive://data/514801/SNPProfile*.csv",
        "mime-type": "text/csv"
    }
}
```

2.7 Error domain, acceptable range of variability "error_domain"

The error domain consists of two subdomains: empirical and algorithmic.

The empirical error subdomain contains the limits of detectability, false positives, false negatives, statistical confidence of outcomes, etc. This can be measured by running the algorithm on multiple data samples of the usability domain or in carefully designed in-silico spiked data. For example a set of spiked, well-characterized samples can be run through the algorithm to determine the false positives, negatives and limits of detection.

The algorithmic subdomain is descriptive of errors that originated by fuzziness of the algorithms, driven by stochastic processes, in dynamically parallelized multi-threaded executions, or in machine learning methodologies where the state of the machine can affect the outcome. This can be measured in repeatability experiments of multiple runs or using some rigorous mathematical modeling of the accumulated errors. For example: bootstrapping is frequently used with stochastic simulation based algorithms to accumulate sets of outcomes and estimate statistically significant variability for the results.

```
"error_domain": {
      "empirical_error": {
            "false_negative_alignment_hits": "<0.0010"
      },
      "algorithmic_error": {
            "false_positive_mutation_calls_discovery": "<0.0005"
      }
}
```

3 Appendices

3.1 Appendix-I: BCO expanded view example


```
{
    "id": "obj.1270",
    "name": "HCV1a [taxonomy:31646] ledipasvir [pubchem.compound:67505836]
resistance SNP [so:0000694] detection",
    "version": "1.1",
    "createdby": "hadley_king@gwmail.gwu.edu",
    "created": "Jan 24, 2017 09:40:17",
    "modified": "Mar 27, 2017 13:27:02",
    "digital_signature": "905d7fce3f3ac64c8ea86f058ca71658",
    "verification_status": "unreviewed",
    "publication_status": "draft",
    "usability_domain": [
        "Identify baseline single nucleotide polymorphisms SNPs [so:0000694],
insertions [so:0000667], and deletions [so:0000045] that correlate with reduced
ledipasvir [pubchem.compound:67505836] antiviral drug efficacy in Hepatitis C
virus subtype 1 [taxonomy:31646]",
"Identify treatment emergent amino acid substitutions [so:0000048] that
correlate with antiviral drug treatment failure",
        "Determine whether the treatment emergent amino acid substitutions
[SO:0000048] identified correlate with treatment failure involving other drugs
against the same virus",
        "GitHub CWL example: https://github.com/mr-c/hive-cwl-
examples/blob/master/workflow/hive-viral-mutation-detection.cwl#L20"
    ],
    "authors": ["Charles Darwin", "http://orcid.org/0000-0000-0000-0000"],
    "description_domain": {
        "keywords": [
            "Antiviral resistance",
            "SNP",
            "Ledipasvir",
            "HCV1a",
            "Amino acid substitution"
        ],
        "xref": [
            "so:0000694",
            "so:0000667",
            "so:0000045",
            "pubchem.compound:67505836",
            "so:0000048",
            "taxonomy:31646",
            "pubmed:25123381",
            "pubmed:26508693"
```


```
        ],
        "github_extension": {
            "github_repository": "hive-cwl-examples",
            "github_url": "https://github.com/common-workflow-language/hive-
cwl-examples",
            "github_uri": ["workflow/hive-viral-mutation-detection.cwl"]
            },
        "pipeline_steps": [
            {
                "tool_name": "HIVE-hexagon",
                "tool_desc": "Alignment of reads to a set of references",
                "tool_version": "1.3",
                "tool_package": "",
                "step_number": "1",
                "reference_files": [
                    {"name": "Hepatitis C virus genotype 1",
                     "URI": "http://www.ncbi.nlm.nih.gov/nuccore/22129792"},
                    {"name": "Hepatitis C virus type 1b complete genome",
                     "URI": "http://www.ncbi.nlm.nih.gov/nuccore/5420376"},
                    {"name": "Hepatitis C virus (isolate JFH-1) genomic RNA",
                     "URI": "http://www.ncbi.nlm.nih.gov/nuccore/13122261"},
                    {"name": "Hepatitis C virus clone J8CF, complete genome",
                     "URI": "http://www.ncbi.nlm.nih.gov/nuccore/386646758"},
                    {"name": "Hepatitis C virus S52 polyprotein gene",
                     "URI": "http://www.ncbi.nlm.nih.gov/nuccore/295311559"}
                ],
                "input_uri_list": [
                    "hive://nuc-read/514683",
                    "hive://nuc-read/514682"
                ],
                "output_uri_list": [
                    "hive://data/514769/allCount-aligned.csv"
                ]
            },
            {
                "tool_name": "HIVE-heptagon",
                "tool_desc": "variant calling",
                "tool_version": "1.3",
                "tool_package": "",
                "step_number": "2",
                "input_uri_list": [
                    "hive://data/514769/dnaAccessionBased.csv"
                ],
                "output_uri_list": [
                    "hive://data/514801/SNPProfile.csv",
                    "hive://data/14769/allCount-aligned.csv"
                ]
            }
        ],
    "execution_domain": {
```

```
        "script_type": "URI",
        "script":
"https://hive.biochemistry.gwu.edu/workflows/antiviral_resistance_detection_hiv
e.sh",
        "pipeline_version": "2.0",
        "platform": "hive",
        "driver": "shell",
        "software_prerequisites": [
            {"name": "HIVE-hexagon","version": "1.3"},
            {"name": "HIVE-heptagon","version": "1.3"}
        ],
        "domain_prerequisites": [
            {
                "url": "protocol://domain:port/application/path",
                "name": "generic name"
            },
            {
                "url": "ftp://:22/",
                "name": "access to ftp"
            },
            {
                "url": "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/",
                "name": "access to e-utils"
            }
        ],
        "env_parameters": [
            {"OSTYPE": "linux"},
            {"QPRIDE_BIN": "~qpride/bin"}
        ]
    },
    "parametric_domain": {
        "heptagon_divergence_threshold_percent": "30",
        "hexagon_minimum_coverage": "0.15",
        "hexagon_seed": "14",
        "heptagon_freq_cutoff": "0.10",
        "hexagon_minimum_match_len": "66"
    },
    "io_domain": {
        "input_uri ": {
            "reference_files": [
                {"name": "Hepatitis C virus genotype 1",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/22129792"},
                {"name": "Hepatitis C virus type 1b complete genome",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/5420376"},
                {"name": "Hepatitis C virus (isolate JFH-1) genomic RNA",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/13122261"},
                {"name": "Hepatitis C virus clone J8CF, complete genome",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/386646758"},
                {"name": "Hepatitis C virus S52 polyprotein gene",
                 "URI": "http://www.ncbi.nlm.nih.gov/nuccore/295311559"}
```

```
            ],
            "input_file_list": [
                "hive://nuc-read/514683",
                "hive://nuc-read/514682",
                "hive://data/514769/dnaAccessionBased.csv"
            ]
        },
        "output_subdomain": {
            "hit_list": {
                "title": "hit list",
                "uri": "hive://data/514769/dnaAccessionBased.csv",
                "mime-type": "text/csv"
            },
            "mutation_profile": {
                "title": "mutation profile",
                "uri": "hive://data/514801/SNPProfile*.csv",
                "mime-type": "text/csv"
            }
        }
    },
    "error_domain": {
        "empirical_error": {
            "false_negative_alignment_hits": "<0.0010"
        },
        "algorithmic_error": {
            "false_positive_mutation_calls_discovery": "<0.0005"
        }
    }
}
```


3.2 Appendix-II: External reference database list

This list contains the databases that are currently being used in our BCOs. We use the CURIEs that map to URIs maintained by *identifiers.org*.
"*Identifiers.org* is an established resolving system the enables the referencing of data for the scientific community, with a current focus on the Life Sciences domain. *Identifiers.org* provides direct access to the identified data using one selected physical location (or resource). Where multiple physical locations are recorded in the registry the most stable one is selected for resolution. This allows the location independent referencing (and resolution if required) of data records."
In the entries below the "namespace" and identifier combine to become the CURIEs.

```
Recommended name: Taxonomy
Namespace: taxonomy
Identifier pattern: ^\d+$
Registry identifier: MIR:00000006
URI: http://identifiers.org/taxonomy/
```

```
Recommended name: Sequence Ontology
Namespace: so
Identifier pattern: ^SO:\d{7}$
Registry identifier: MIR:00000081
URI: http://identifiers.org/so/

Recommended name: PubMed
Namespace: pubmed
Identifier pattern: ^\d+$
Registry identifier: MIR:00000015
URI: http://identifiers.org/pubmed/

Recommended name: PubChem-compound
Namespace: pubchem.compound
Identifier pattern: ^\d+$
Registry identifier: MIR:00000034
URI: http://identifiers.org/pubchem.compound/
```

3.2.2 Title 21 CFR Part 11

Code of Federal Regulations Title 21 Part 11: Electronic Records - Electronic Signatures

BioCompute project is being developed with Title 21 CFR Part 11 compliance in mind. The digital signatures incorporated into the format will provide the basis for provenance of BioCompute Object integrity using NIST proposed encryption algorithms. Execution domain and parametric domain (that have a potential impact on a result of computation) and identity domain will be used to create hash values and digital signature encryption keys which later can be used for computer or human validation of transmitted objects.

Discussions are now taking place to consider relevance of BioCompute Objects with relation to Title 21 CFR part 11. We encourage continuous input from BioCompute stakeholders on this subject now and while the concept is becoming more mature and more widely accepted by scientific and regulatory communities.

Relevant document link:
Part 11: Electronic Records

3.3 Appendix IV - Compatibility

3.3.1 ISA for the experimental metadata

ISA is a metadata framework to manage an increasingly diverse set of life science, environmental and biomedical experiments that employ one or a combination of technologies. Built around the **Investigation** (the project context), **Study** (a unit of research) and **Assay** (analytical

measurements) concepts, ISA helps to provide rich descriptions of experimental metadata (i.e. sample characteristics, technology and measurement types, sample-to-data relationships) so that the resulting data and discoveries are reproducible and reusable. The ISA Model and Serialization Specifications define an Abstract Model of the metadata framework that has been implemented in two format specifications, ISA-Tab and ISA-JSON (http://isa-tools.org/format/specification), both of which have supporting tools and services associated with them, including by a programmable Python AP (http://isa-tools.org) and a varied user community and contributors (http://www.isacommons.org). ISA focuses on structuring experimental metadata; raw and derived data files, codes, workflows etc are considered as external file that are referenced. An example, along its complementarity with other models and a computational workflow is illustrated in this paper, which shows how to explicitly declare elements of experimental design, variables, and findings: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0127612

3.4 Appendix V Data typing

The conceptual schema for BCO creation is built on top of two layers: the data definition layer and the BCO layer. The first layer is where all fundamental data types are defined. Complex types are composed of multiple atomic or complex types, like a character string. Using these principles one can construct a datum that has the ability to represent any level of complexity that is needed. A BCO is, essentially, a federation of other objects.

3.4.1 Primitive data types

When defining a field in a data type, one can place any number of constraints on the data and the field will be accepted as valid. So, if a data type field was being constructed for holding DNA sequencing information, one could restrain the type of characters that field would accept. This further refinement would ensure that only the characters used for representing nucleic acids would be accepted as input in this field (e.g. A, T, C, G). A list of the primitive types used in BCO data typing is below.

```
{
    "primitives": {
        "description": "primitives branch",
        "bool": {
            "_type": "core",
            "_limit": {
                    "true" : 1,
                    "false" : 0
                }
            },
        "int": {
            "_type" : "core"
        },
        "uint": {
            "_type": "int",
```

```
    "_limit": {
        "eval-c/c++" : "$val>=0"
    }
},
"percent": {
    "_type": "uint",
    "_limit": {
        "eval-c/c++": "$val<=100 && $val>=0"
    }
},
"string" : {
    "_type": "core"
},
"text" : {
    "_type": "core"
},
"memory" : {
    "_type": "uint"
},
"cmdline": {
    "_type": "text"
},
"url" : {
    "_type": "text"
},
"file" : {
    "_type": "string"
},
"date" : {
    "_type": "core",
    "_default": "$datenow()"
},
"time" : {
    "_type": "core",
    "_default": "$timenow()"
},
"datetime": {
    "_type": "core",

    "date": {
        "_type": "date",
        "_default" : "$datenow()"
    },
    "time": {
        "_type": "time",
        "_default": "$timenow()"
    }
},
"timespan": {
    "_type": "core",
```

```
            "start": {
                "_type": "datetime"
            },
            "end": {
                "_type": "datetime"
            }
        },
        "ref": {
            "_type": "core",

            "source": {
                "_type": "string"
            },
            "db": {
                "_type": "string"
            },
            "id": {
                "_type": "string",
                "_vital": true
            }
        },
        "password": {
            "_type": "string"
        },
        "email": {
            "_type": "string"
        },
        "keyval": {

            "_type": "core",

            "key": {
                "_type": "string"
            },

            "value" : {
                "_type": "string",
                "_plural": true
            }
        }
    }
}
```

3.4.2 Base BioCompute Type

The second layer is constructed with objects from first layer, producing a derived data type called the "base BioCompute type". Extending the same principles that allowed us to construct a string representing a DNA sequence from the primitive character type, one can construct a type definition that is the absolute minimum fields necessary to create a BCO. By taking the primitive BCO type

and adding parametric and metadata fields unique to a particular instance, a BCO can be created. Below it the type definition for “BioCompute_base_type”:

```
{
    "BioCompute_base_type": {
        "_type": "type",
        "_id": "$newid()",
        "_inherit": "base_database_object",
        "name": "BioCompute_base_type",
        "title": "Base type for all BioCompute Objects",
        "descr": "all BioComputes must inherit from this type in order to be
compliant with BioCompute framework",

        "_field": {
            "id": {
                "_type":"string",
                "title": "Identifier of the object",
                "_public": true,
                "_write": false
            },
            "name": {
                "_type":"string",
                "title": "public searchable name for BioCompute Objects",
                "_public": true
            },
            "structured_name": {
                "_type":"string",
                "title": "public searchable name for BioCompute Objects",
                "_public": true,
                "_vital": false
            },
            "version": {
                "_type":"string",
                "title": "version for BioCompute Object",
                "_public": true
            },
            "digital_signature": {
                "_type": "string",
                "title": "digital signature of the BioCompute Object",
                "_write":false,
                "_vital": false
            },
            "verification_status": {
                "_type": "string",
                "_limit": {
                    "choice ": ["in_progress", "unreviewed", "reviewed",
"published", "rejected"]
                },
                "title": "the current verification status of the BioCompute
Object",
                "_default": "in_progress"
```

```
            },
            "publications_status": {
                "_type": "string",
                "_limit": {
                    "choice ": ["draft", "in_progress", "published",
"embargoed"]
                },
                "title": "the current publication status of the BioCompute
Object",
                "_default": "draft"
            },
            "authors": {
                "_type": "core",
                "name": {
                    "_type": "string",
                    "_hidden": "eval-js: orcid===undefined"
                },
                "orcid": {
                    "_type": "string",
                    "_hidden": "eval-js: name===undefined"
                },
                "_plural": true
            },
            "usability_domain": {
                "_type": " string",
                "_plural": true,
                "descr": "text from biospec"
            },
            "description domain": {
                "keywords": {
                    "_type": "keyval",
                    "_plural": true,
                    "vital": false
                },
                "xref": {
                    "_type": "xref",
                    "_plural": true,
                    "vital": false
                },
                "pipeline_steps": {
                    "tool_name": {
                        "_type": "string",
                        "descr": "this is a recognized name of the software
tool"
                    },
                    "tool_desc": {
                       "_type": "text"
                    },
                    "tool_version": {
                       "_type": "string"
```

```
            },
            "tool_package": {
               "_type": "string",
                "_plural": true
            },
            "step_number": {
               "_type": "uint",
                "_limit": {
                    "eval-js": "$step_number >= 1"
                }
            },
            "input_uri_list": {
                "_type": "url",
               " _vital": false
            },
            "output_uri_list": {
                "_type": "url",
               " _vital": false
            },
            "_plural": true
        },
        "_vital": false
    },
    "execution_domain": {
        "script_type": {
           "_type":"string",
            "_limit": {
                "choice": ["uri","text"]
            }
        },
        "script": {
           "_type": "$script_type"
        },
        "script_driver": {
           "_type": "string"
        },
        "pipeline_version": {
            "_type": "string"
        },
        "platform": {
            "_type": "string"
        },
        "software_prerequisites": {
            "name": {
                "_type": "string"
            },
            "version": {
                "_type": "string"
            },
            "url": {
```

```
                        "_type": "url",
                        "_vital": false
                    },
                    "_plural": true

                },
                "access_prerequisites": {
                    "name": {
                        "_type": "string"
                    },
                    "url": {
                        "_type": "url",
                        "_plural": true
                    }
                },
                "environmental_parameters": {
                    "_type": "keyval",
                    "_plural": true
                },
                "_vital": true
            },
            "parametric_domain": {
                "descr": "all fields in this domain should be defined in
inheriting BioCompute subtypes"
            },
            "io_domain": {
                "input_subdomain": {
                    "descr": "all fields in this subdomain are specific
BioCompute specific and should be defined in inheriting BioCompute subtypes"
                },
                "output_subdomain": {
                    "descr": "all fields in this subdomain are specific
BioCompute specific and should be defined in inheriting BioCompute subtypes"
                }
            }
        }
    }
}
```

3.4.3 Meaning associations

All data field values stored in the data format have an associated file name and/or absolute or relative field value location. The association and assignment of meaning to a particular data value can be ontological (derived from a name-value pair), or topological (derived from a location-value pair).

As an example, in a name-value pair, the name of the field is explicitly specified. Examples of name-value pairing are: a JSON object; the mapping from ids to nodes in an XML document; or the association between the column names and columns of a CSV table. In a location-value pair, the

name of the field is not specified but assumed, according to the order and positioning of the value assumes its meaning. An example of location-value pairing is a FASTQ file, where sequence and quality lines do not have explicit names, but their relative locations are used to identify the data line type.

3.4.4 Atomic and complex data

Data values can be atomic, (not decomposable into smaller sets without the loss of meaning) or complex (containing other simpler values or topologies). Data structures of arbitrary complexity can be created using substructures and hybrids of name-value and location-value aggregations. Some fields may be declared parent fields of others and the field name value pairs can be allocated along the hierarchy of field types. It is important to note that atomicity of the field values here does not directly imply independent scientific interpretability of the atomic values: it only implies that such information cannot be fragmented into smaller more fundamental types.

For example, the value for a person's age is an atomic value and cannot be decomposed. However, a person's identity is a complex data which contains first name, last name, age, social security number, passport number and perhaps other simpler values which have a meaning in of their own.

3.4.5 Fundamental and derivable types

Primitive field types, such as numbers, strings, bit-fields, and uncharacterized blobs, are generally the most frequently used; this fact is reflected in computer science where these four types are chosen as fundamental units of data representation. A data type by itself does not carry physical interpretation; this interpretation comes from the value associations mentioned above. Other field types (sequences, alignments, etc.), whether atomic or complex types, are constructed using the fundamental types and adjusting their interpretive significance. Computable data types, such as references and formulas, are also useful for creating operational information infrastructures; such fields do not carry a value themselves but link to another value, in the same or another object, directly or through the usage of a transformation formula.

In the person’s identity example mentioned in the previous section, the atomic subfields first name, last name, age, etc. are primitive data types such as strings and numbers.

Data typing is the process of creating a derived data type as a collection of primitive field types and previously defined complex types. During this process, field attributes such as name, type, constraints, and default value are specified for each of the fields.

To construct the type "Identity" we define a collection of the fundamental fields that comprise this derived type. ***Identity:***

- *First name is a free text string.*
- *Last name is a free text string.*

- *Age is a single whole number.*

Identity: {Charles, Darwin, 208}

Derived/computed fields

Derived/computed data types are produced during the output of the object instance and cannot be entered during instantiation of the object into the database. These fields are just the reproducible outcome of other field values through some kind of derivation mechanisms. For example: a taxonomy identifier is non-redundantly unique enough to recover different names, synonyms and relations for a taxonomic unit. However, a derived field "name" can be a useful informative construct derived and outputted every time where taxonomic identification is necessary.

```
TaxID: 9606
```

Virtual fields

Field values can be explicitly specified in an object instance or computable by other means. For example: the metadata object describing a file can have a field named "file-size" reflective of the actual file size in storage. The value of that field cannot be explicitly specified or computed from other values in the same object. Such virtual fields are descriptive and can be used for validation of the object itself and for definition of further constraints.

Constraints

Field values can have explicit and implicit constraints determining the universe of possible values for that particular field. Implicit constraints may be derived from the actual data types where, for example, numeric fields can have only numeric values and strings can have only characters of a particular alphabet. Explicit constraints are those specifically targeting a particular field in a data format, further limiting the value to a smaller set of possible values. Implicit and explicit constraints can be validated either by ensuring their syntax conformance or by evaluation of validation expression criteria. Correlation constraints can be defined on a field as well, demanding a condition when constraints on one field depend on the value(s) of other fields. There can be a scale of constraint rigorousness depending on the impact due to violation of such constraints. Soft constraint nonconformity can signify potential devaluation of a particular field value pair while hard constraint violation in a single field value pair would invalidate the entirety of the data as a whole.

Existence of static and dynamic constraints is necessitated by the changing nature of the interdependencies of local and global information. Static constraints are those whose validation expression depends only on the data itself while dynamic constraints depend on external or dynamically changeable information from local or global sources. An example of an explicit, dynamic, and correlated constraint-carrying field type is XREF (cross-reference) for which both the

ID and the cross-referenced source must be included and the validity of the field can be evaluated only by inclusion of potentially changeable dynamic content of the external source.

It is important to note that constraints may limit the universe of possible values to discrete or continuous subsets of a finite or infinite enumerable list of values. Actual implementations of constraints may vary from a form of numerical open (]min,max[, closed ([min,max] or mixed ([min,max[,]min,max]) ranges, lists of ranges, lists of strings and possible values to mathematical expressions producing true or false Boolean values based on the computation of a formula.

Single and multi-value

An additional level of empowering complexity can be achieved if fields are allowed to have multiple values. Unlike single-value fields, a multi-value field carries multiple values associated with a single name-value or location-value assignation. Using this paradigm, one can introduce notions of a list and an array into data format containers. The difference between these types is that an array's children elements are topologically bound to their location in a row and therefore, are related to each other. In the list however, children elements of the same or of a different kind are simply a collection of non-coupled values.

Public and private fields

Security and access control rights play an important role when considering data access patterns in modern collaborative information systems. Data can be shared in such environments for viewing and modifications. One may want to share only a subset of fields containing certain descriptive values broadcasting information about the existence of a particular metadata instance through a search and browsing engine. Thus, the researcher or collaborator who is interested in having access to your data may then ask for more complete (read or write) access to the dataset that was found through browsing its public properties. To satisfy this need, we can have an attribute called public for a field that declares it to be available for searching and browsing by other users.

Uniqueness and key values

The importance of any particular field's role in data format definition can vary within a local or global context. Some fields are unique within the given instance (object) of the data file and some are required to be unique within the global scope of such objects in some database. Such fields can be used as unique key identifiers of a whole data file within a known context.

Incompleteness

Data formats can support both mandatory and optional fields. A valid data file must include all its mandatory fields without exception. Such fields are typically essential in representation of the underlying information and their absence can devalue or introduce ambiguities into interpretation of

other fields. Optional fields are those carrying non-critical values; absence of those values does not devalue the interpretation of other fields. There are two possible approaches to optional fields: default value and undefined value. An empty optional field can be treated as having an agreed upon predefined default value or can simply call the value undefined or NULL which must be interpreted as non-equal to any other value, even to another undefined. Any operation performed with undefined values must also be assigned the value of undefined.

Arbitrary key-value pairs

Certain file formats allow incorporation of arbitrary key value pairs without specific designation with the objective of adopting unstructured, descriptive, or information that may be useful in the future. Although the ultimate goal of such arbitrary fields is to maintain some level of extensibility in formats, those arbitrary fields usually end up containing a significant amount of ill structured and unverifiable, non-canonical information. By providing other means to extensibility, we strongly discourage using arbitrary key-value pairs for anything other than purely descriptive, non-critical information.

3.4.6 Extensibility through inheritance and inclusion of data types

It is of the utmost importance to generate extensible metadata formats capable of providing the basis for more complex new types. There are two proposed ways to extend a data format: inheritance and inclusion.

The concept of inheritance assumes that a more complex data type inherits all the field value pairs from another, simpler data type and extends the content only with additional field value pairs or customizes (redefines some characteristics) of existing fields. The concept of inclusion assumes that a particular field of an object is of a previously declared complex type and that it contains all the fields of a simpler data type. A single data type can inherit from multiple data types and can include multiple data types multiple times.

Using these two paradigms, one can design a number of layered standard objects based on predefined objects and extend their functionality with specific fields. For example: imagine a metadata object of type bio-sample which has a predefined fundamental description applied to a generalized sample. This object can have its properties repeatedly inherited to create a human-sample object with increasingly specific information about particularities of that sample description.

The two proposed extensibility models allow avoiding the overuse of optional field attributes that are present in conglomerated flat data type designs. Instead of designing wide and flat data types with all possible fields for different use-cases, one may choose to design more targeted types with specifically mandated fields inside.

For example, having an optional tissue-location field in all biological-sample objects might lead to sparse population of the field as it will be unpopulated for all environmental, metagenomic, bacterial, and viral samples where the notion of a tissue is irrelevant. However, designing an inherited animal-sample data type can have a mandatory tissue location field for instances when it is important to know from which part of the animal a particular sample was collected.

The power of the inheritance and inclusion methods to extend and implement new data types is evident when one considers the need to create new subtypes or a branch of existing types after the initial data -type structure is established. This step can be accomplished without modification of existing database objects by defining the new intermediates within the framework of the pre-defined metadata type hierarchy.

The relationship implemented by inheritance subtyping is a is-a relationship. For example, the type "fish" can have three subtypes "eel", "shark" and "salmon". Each subtype is a variety of the "fish" supertype and inherits all "fish" characteristics but has some specific differences.

3.4.7 Data lifecycle timeline

Data objects are typically records stored in an information system: a file system or a database. The life of such a record starts at the moment of metadata file submission. Typical preprocessing steps include: files being parsed and validated regarding their conformity with data standards, application of quality control processes and designation of appropriate permissions for later use. Depending on the size and complexity of the data, as well as the load on the data processing subsystems, this period may take seconds to days for NGS data. After this initial preprocessing stage, objects become visible to the owner/submitter of the information.

The user can then specify the validity start time before which the object is not to be accessed by anyone other than the user. This feature is useful for providing pre-publication delays or time fixed processing procedures. The user can also specify the validity end period after which the object is not to be used by anyone other than the owner of the record.

Optional soft and hard expiration periods can be set. These properties signify when the object should become "expired" from the database and should be treated as "deleted," and when the record is actually deleted from the database, respectively. The timespan between these two time periods is the potential recovery period; during this period the deletion can be reverted by manual or electronic inquiry from the user to the DB administrator/manager.

Another important milestone of the data existence is set by FDA’s mandate to maintain an archival copy of any review data used to make regulatory decisions. This copy does not necessarily reside in any easily accessible database or file system and is managed by a different set of regulations, the description of which lies outside the scope of this document.

3.5 Appendix VI Acknowledgements

This document began development during the 2017 HTS-CSRS workshop. The discussion during the workshop facilitated the refinement and completion of this document. The workshop participants were a major part of the initial BCO community, and the comments and suggestions collected during the sessions were incorporated into this document. The people who participated in the 2017 workshop, and therefore made significant contributions are listed here: https://osf.io/h59uh/