Advanced Energy System Design (AESD): Technical Manual for the Records API

Nicholas Brunhart-Lupo, Brian Bush, Kenny Gruchalla, and Michael Rossol
National Renewable Energy Laboratory

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Technical Report
NREL/TP-6A20-68924
August 2018

Contract No. DE-AC36-08GO28308

NREL
NATIONAL RENEWABLE ENERGY LABORATORY

Advanced Energy System Design (AESD): Technical Manual for the Records API

Nicholas Brunhart-Lupo, Brian Bush, Kenny Gruchalla, and Michael Rossol
National Renewable Energy Laboratory

Suggested Citation
Brunhart-Lupo, Nicholas, Brian Bush, Kenny Gruchalla, and Michael Rossol. 2018. Advanced Energy System Design (AESD): Technical Manual for the Records API. Golden, CO: National Renewable Energy Laboratory. NREL/TP-6A20-68924.
https://www.nrel.gov/docs/fy18osti/68924.pdf

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

Technical Report
NREL/TP-6A20-68924
August 2018

Contract No. DE-AC36-08GO28308

NOTICE

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. This work was supported by the Laboratory Directed Research and Development (LDRD) Program at NREL. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

Acronyms and Glossary

ACI	Application Container Image
AESD	Advanced Energy System Design
API	application programming interface
C++	a programming language
CSV	comma-separated-value file
Chrome	a web browser
Firefox	a web browser
Google Protocol Buffers	a serialization specification
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
Haskell	a programming language
IoT	Internet of Thinks
JSON	JavaScript Object Notation
JavaScript	a programming language
MySQL	a database server product
NREL	National Renewable Energy Laboratory
ODBC	Open Database Connectivity
POSIX Epoch	seconds since midnight 1 January 1970 UTC
PostgreSQL	a database server product
Project Haystack	a specification for data feeds from the Internet of Thinks (IoT)
Python	a programming language
R	a programming language
REST	representational state transfer
Rkt	a container engine (CoreOS 2017b)
SQLite3	a database server product
TSV	tab-separate-value file
URI	uniform resource identifier
UTC	Coordinated Universal Time
WebSockets	a communication protocol
YAML	YAML Ain’t Markup Language

Abstract

The Records API (application program interface) for Advanced Energy System Design (AESD) enables software that serves multidimensional record-oriented data to interoperate with software that uses such data. In the context of the Records API, multidimensional data records are simply tuples of real numbers, integers, and character strings, where each data value is tagged by a variable name, according to a pre-defined schema, and each record is assigned a unique integer identifier. Conceptually, these records are isomorphic to rows in a relational database, JSON objects, or key-value maps. Records servers might supply static data sets, sensor measurements that periodically update as new telemetry become available, or the results of simulations as the simulations generate new output. Records client software might display or analyze the data, but in the case of simulations, the client requests the creation of new ensembles for specified input parameters. It is also possible to chain records clients and servers together so that a client consuming data from a server might transform that data and serve it to additional clients.

This minimalist API avoids imposing burdensome metadata, or structural or implementation requirements on developers by relying on open source technologies that are readily available for common programming languages. In particular, the API has been designed to place the least possible burden on services that provide data. This document defines the message format for the Records API, a transport mechanism for communicating the data, and the semantics for interpreting it. The message format is specified as Google Protocol Buffers (Google Developers 2017a) and the transport mechanism uses WebSockets (Internet Engineering Task Force 2017). We discuss five major use cases for serving and consuming records data: (1) static data, (2) dynamically augmented data, (3) on-demand simulations, (4) with filters, and (5) with bookmarks. Separate implementations of the API exist in C++, Haskell, JavaScript, Python, and R.

Table of Contents

List of Figures

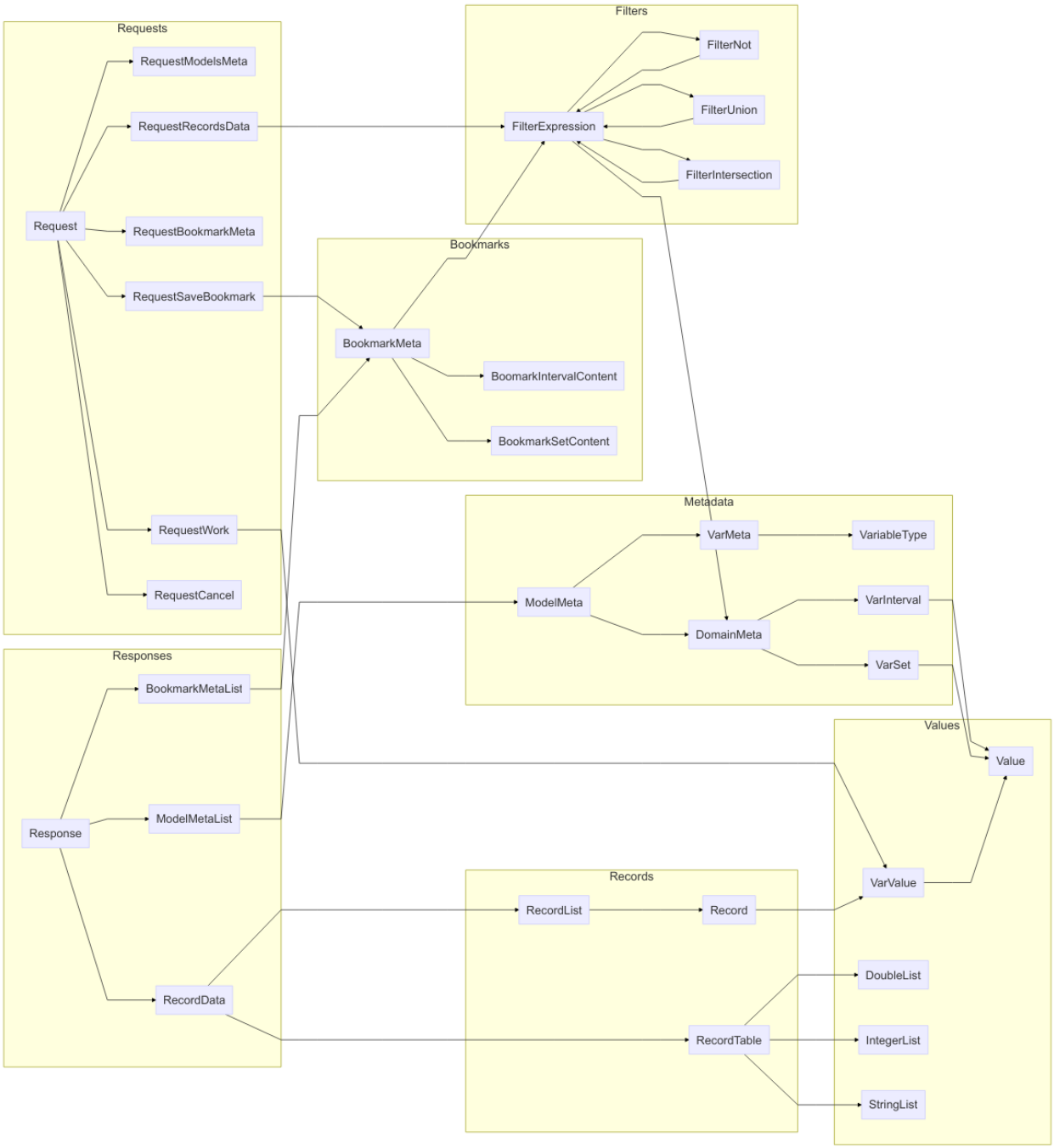
List of Tables

Overview

Client-server communication in the Records API simply consists of clients sending Request messages to the server and servers asynchronously sending Response messages to the client. The request and response messages hold the specifics of the request or response and the responses are correlated with the requests; however, it is important to note that multiple responses may occur for a single request, as when record data are chunked into multiple response, or that an error response may be sent at any time. The nested messages within `Request` and `Response` may in turn contain nested fields and messages providing further details. The table below shows the correspondence between requests and responses, while the figure following that shows the containment relationships between message types.

Correlation between Requests and Responses

Request Field	Response Field
models_metadata	models or `error`
records_data	data or `error`
bookmark_meta	bookmarks or `error`
save_bookmark	bookmarks or `error`
cancel	no response or `error`
work	data or `error`



Containment relationships between protocol buffer messages in the Records API

Metadata messages describe "models," which are just sources of data, and the variables they contain. Data record messages hold the data itself. Data records are simply tuples of real numbers, integers, and character strings, where each data value is tagged by a variable name, according to a pre-defined schema, and each record is assigned a unique integer identifier. Conceptually, these records are isomorphic to rows in a relational database, JSON objects, or key-value maps. For efficiency and compactness, RecordData may be provided in list format or tabular format, with the latter format obtained only when the contents of the table all have the same data type. The data records may be provided *in toto* or filtered using filter messages so that only certain fields or records are returned. The API contains a small embedded language for filtering via set and value operations. Sets of records may be bookmarked for sharing or later retrieval by (1) enumerating their unique record identifiers, (2) defining a range of unique record identifiers, or (3) specifying a filtering criterion.

Servers that perform computations or simulations can receive input parameters via a RequestWork message that contains those input parameters. After the server has completed its computations, it sends the results as RecordData messages.

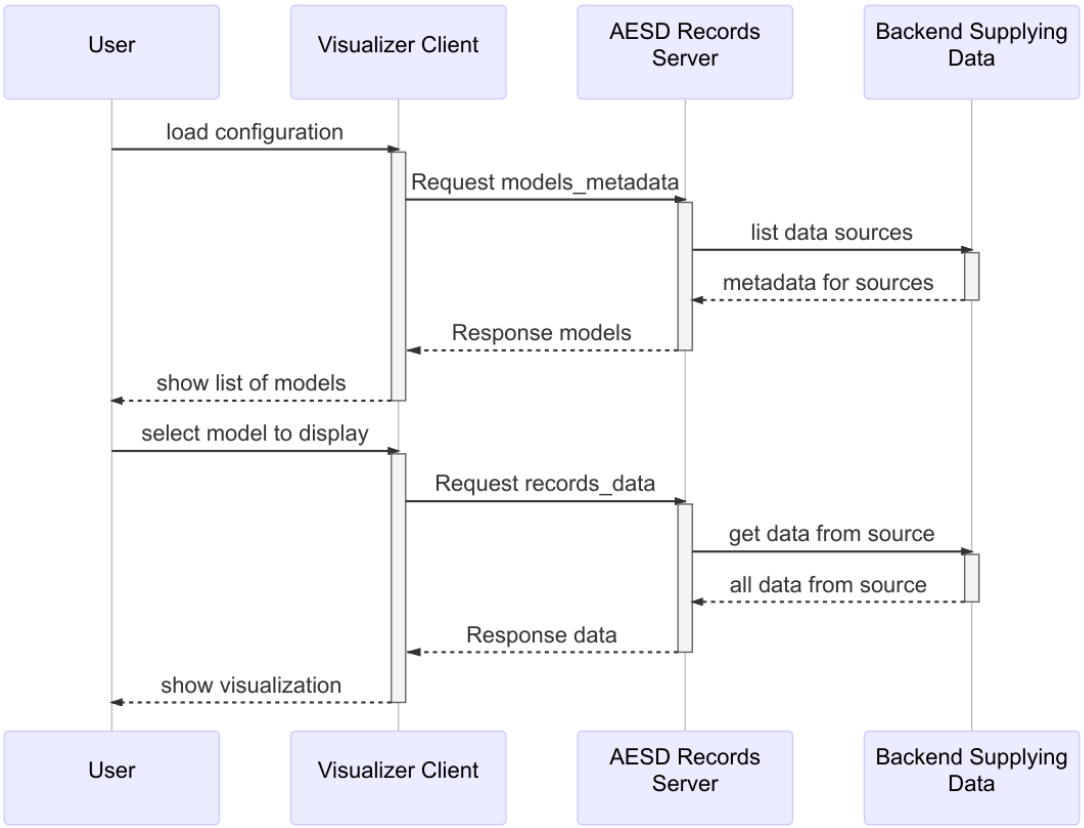
In general, the response to a request for data records comes in *chunks* numbered in sequence, where each chunk has an identifier, `chunk_id`, and the response specifies the identifier of the next chunk, `next_chunk_id`. Thus, the chunks form a linked list. The sending of additional chunks can be cancelled using a RequestCancel message. If the subscribe flag is set when making a request, the server will respond indefinitely with additional data as it becomes available, until the subscription is cancelled.

Use Cases

In this section we outline some standard use cases for the Records API. UML Sequence Diagrams (Fowler 2017) illustrate the flow of messages and the messages themselves are printed in the text format output by the Google `protoc` tool (Google Developers 2017b).

Static Data

The retrieval of static data records forms the simplest use case for the Records API. A user chooses a particular data source (a "model" in the parlance of the Records API) and then the data are retrieved and displayed. The visualization client software communicates with a Records server, which in turn accesses the static data. The figure below illustrates the process.



Visualizing data from a static source using the Records API

A Request without `model_id` specified requests the server to list all models:

```
version: 4
id: 1
models_metadata {
}
```

The Response from the server provides metadata for all of the models:

```
version: 4
id: 1
models {
  models {
    model_id: "example-model-1"
    model_name: "Example Model #1"
    model_uri: "http://esda.nrel.gov/examples/model-1"
    variables {
      var_id: 0
      var_name: "Example Real Variable"
      type: REAL
    }
    variables {
      var_id: 1
      var_name: "Example Integer Variable"
```

```
      type: INTEGER
    }
    variables {
      var_id: 2
      var_name: "Example String Variable"
      type: STRING
    }
  models {
    model_id: "example-model-2"
    model_name: "Example Model #2"
    model_uri: "http://esda.nrel.gov/examples/model-2"
    variables {
      var_id: 0
      var_name: "POSIX Epoch"
      type: INTEGER
    }
    variables {
      var_id: 1
      var_name: "Measurement"
      type: REAL
    }
  }
  models {
    model_id: "example-simulation-3"
    model_name: "Example Simulation #3"
    model_uri: "http://esda.nrel.gov/examples/simulation-3"
    variables {
      var_id: 0
      var_name: "Input"
      type: REAL
    }
    variables {
      var_id: 1
      var_name: "Time"
      type: REAL
    }
    variables {
      var_id: 2
      var_name: "Value"
      type: REAL
    }
    inputs {
      var_id: 0
      interval {
        first_value: 0
        second_value: 100
      }
    }
  }
}
```

Note that the response above is tagged with the same `id` as the request; this allows the client to correlate responses with the particular requests it makes. Next, the user might request three records from the first model:

```
version: 4
id: 2
records_data {
  model_id: "example-model-1"
  max_records: 3
}
```

The record data might be returned as two chunks, where the first chunk is

```
version: 4
id: 2
chunk_id: 1
next_chunk_id: 2
data {
  list {
    records {
      record_id: 10
      variables {
        var_id: 0
        value: 10.5
      }
      variables {
        var_id: 1
        value: -5
      }
      variables {
        var_id: 2
        value: "first"
      }
    }
    records {
      record_id: 20
      variables {
        var_id: 0
        value: 99.2
      }
      variables {
        var_id: 1
        value: 108
      }
      variables {
        var_id: 2
        value: "second"
      }
    }
```

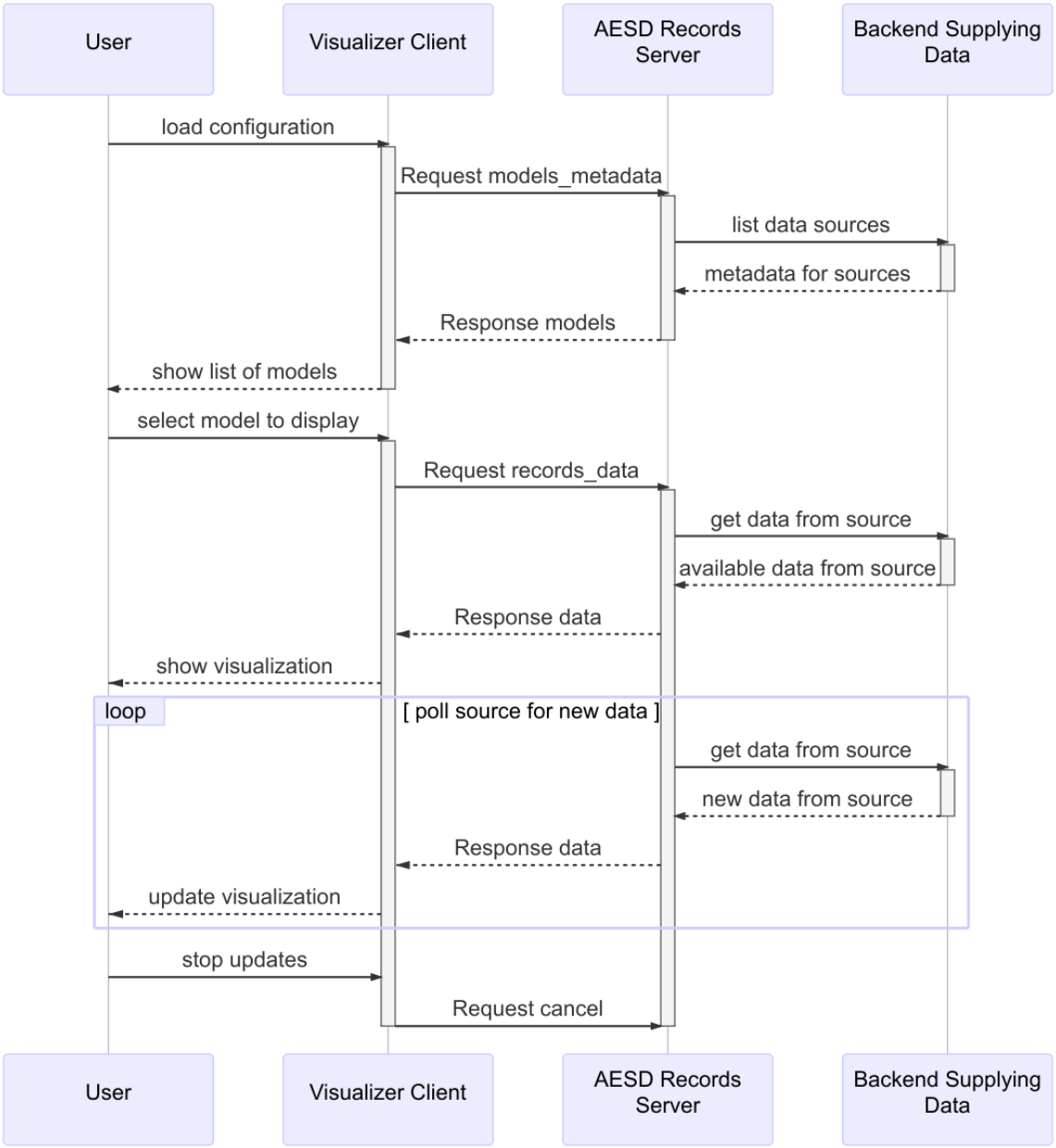
```
  }
}
```

and the last chunk is:

```
version: 4
id: 2
chunk_id: 2
next_chunk_id: 0
data {
  list {
    records {
      record_id: 30
      variables {
        var_id: 0
        value: -15.7
      }
      variables {
        var_id: 1
        value: 30
      }
      variables {
        var_id: 2
        value: "third"
      }
    }
  }
}
```

Dynamic Data

As shown in the following figure, retrieving data from a dynamic source proceeds quite similarly to retrieving data from a static source. The only essential difference is that the server repeatedly sends additional responses containing new data, until a request to cancel is sent.



Visualizing data from a dynamic source using the Records API

When requesting dynamic data, it is advisable to set the `subscribe` flag in the request for data:

```
version: 4
id: 2
subscribe: true
records_data {
```

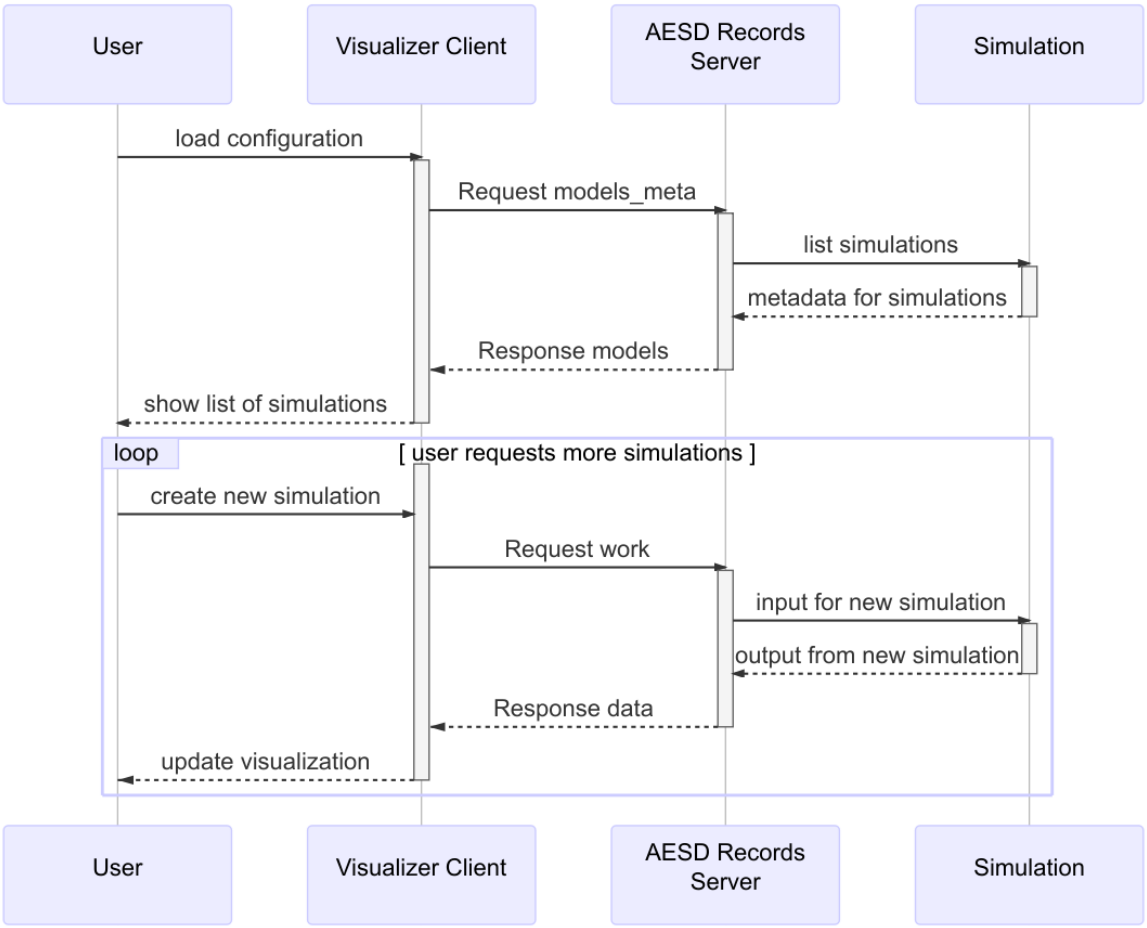
```
  model_id: "example-model-2"
}
```

The RequestCancel message is the `cancel` field Request and must include the `id` of the request to be cancelled:

```
version: 4
cancel {
  id: 2
}
```

Simulations

The model `Example Simulation #3` in the Static Data use case is a simulation model, as evidenced by the presence of the `inputs` field in its metadata. The following figure shows a typical interaction with a simulation-based model via the Records API.



Steering and visualizing simulation results using the Records API

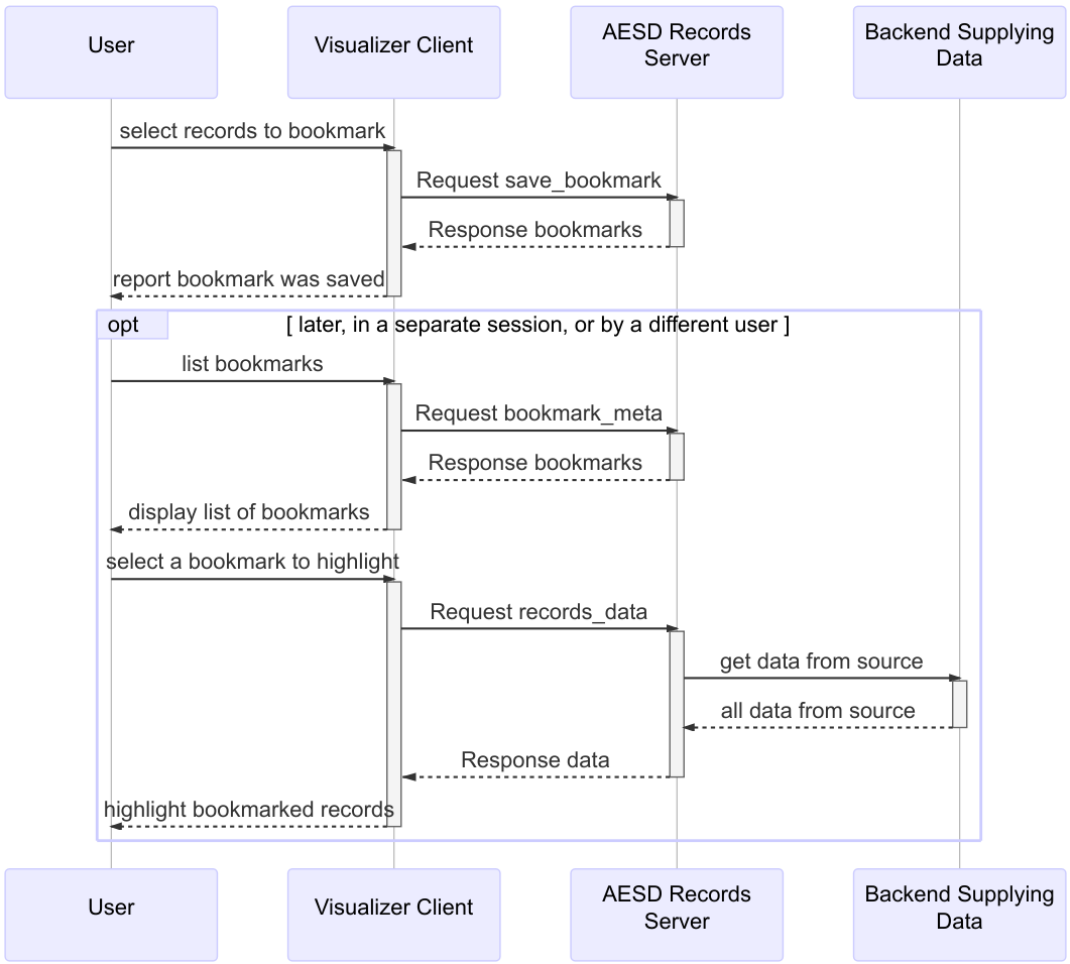
The RequestWork message, which is contained in the `work` field of a Request, specifies the input for a simulation to be run:

```
version: 4
id: 3
work {
  model_id: "example-simulation-3"
  inputs {
    var_id: 0
    value: 50
  }
}
```

The response to this message will be data for the result of the simulation.

Bookmarks

Once data from a model is loaded, it may be bookmarked. One simply supplies a description of the data to be bookmarked. Bookmarks can be listed and loaded, as shown in the following figure.



Creating and retrieving a bookmark and its associated data

To create a bookmark for a specific list of records, simply supply their record identifiers as part of a BookmarkMeta message in the `save_bookmark` field of Request:

```
version: 4
id: 4
save_bookmark {
  model_id: "example-model-1"
  new_bookmark {
    bookmark_name: "Sample Bookmark"
    set {
      record_ids: 10
      record_ids: 30
    }
  }
}
```

The response will be the same bookmark but with the `bookmark_id` field added:

```
version: 4
id: 4
bookmarks {
  bookmark_metas {
    bookmark_id: "bookmark-1"
    bookmark_name: "Sample Bookmark"
    set {
      record_ids: 10
      record_ids: 30
    }
  }
}
```

The user, or another user, can retrieve the records corresponding to the bookmark:

```
version: 4
id: 5
records_data {
  model_id: "example-model-1"
  bookmark_id: "bookmark-1"
}
```

This will return precisely the bookmarked records:

```
version: 4
id: 5
data {
  list {
    records {
      record_id: 10
      variables {
        var_id: 0
```

```
        value: 10.5
      }
      variables {
        var_id: 1
        value: -5
      }
      variables {
        var_id: 2
        value: "first"
      }
    }
    records {
      record_id: 30
      variables {
        var_id: 0
        value: -15.7
      }
      variables {
        var_id: 1
        value: 30
      }
      variables {
        var_id: 2
        value: "third"
      }
    }
  }
}
```

Filtering

Filtering records can be used to select particular records for retrieval, via the RequestRecordsData message, or in defining bookmarks via the BookmarkMeta message. Filtering of records is accomplished through expressions (FilterExpression), combining values for variables (DomainMeta), and the set operators *not*, *union*, and *intersection*, encoded in the messages FilterNot, FilterUnion, and FitlerIntersection respectively. For example, the expression $x \leq 20$ would be expressed as the following FilterExpression:

```
filter_domain {
  interval {
    var_id: 0
    last_value: 20
  }
}
```

provided that x has `var_id = 0`. The expression $(10 \leq x \leq 20) \cup (y \notin \{4,7\})$ would be expressed as

```
filter_union {
  filter_expressions {
```

```
    filter_domain {
      var_id: 0
      first_value: 10
      last_value: 20
    }
    filter_not {
      filter_expression {
        filter_domain {
          var_id: 1
          set {
            elements: 4
            elements: 7
          }

      }
    }
  }
}
```

provided that x has `var_id = 0` and y has `var_id = 1`.

Records API, Version 4

The Records API consists of Google Protobuf 3 (Google Developers 2017b) messages used to request and provide data and metadata for record-oriented information. This section contains the complete specification for Version 4 of the Records API. Clients send `Request` messages and servers send `Response` messages, which are typically transported via WebSockets (Internet Engineering Task Force 2017).

Message Groups

The message types in the Records API are organized into thematic groups below.

Requests and Responses

`Request` messages are sent from client to server, and `Response` messages are sent from server to client. Request messages contain a specific type of request and response messages contain a corresponding specific type of response.

- Request
- RequestModelsMeta
- RequestRecordsData
- RequestWork
- RequestBoomarkMeta
- RequestSaveBookmark
- RequestCancel
- Response

Metadata

Metadata messages describe data sources ("models") and variables.

- ModelMeta
- ModelMetaList
- DomainMeta
- VarMeta
- VariableType
- VarSet

- VarInterval

Data Records

Data are represented as either lists of records or tables of them.

- Record
- VarValue
- Value
- RecordData
- RecordList
- RecordTable

Filtering

Records can be filtered by logical operations on conditions for values of variables in the records.

- FilterExpression
- FilterNot
- FilterIntersection
- FilterUnion
- DomainMeta

Bookmarks

Bookmarks record particular sets or records or conditions for record data.

- BookmarkMeta
- BookmarkMetaList
- BookmarkIntervalContent
- BookmarkSetContent

Miscellaneous

The following messages wrap data types for the content of records.

- DoubleList
- IntegerList

- StringList
- OptionalInt32
- OptionalUInt32
- OptionalString

General Conventions

All fields are technically optional in ProtoBuf 3, but some fields may be required in each message type in order for the message to be semantically valid. In the following specifications for the messages, fields are annotated as *semantically required* or *semantically optional*. Also, the specification notes when field in the protobuf oneof construct are required or mutually exclusive.

Furthermore, one cannot determine whether an optional value has been set if it is just a value, as opposed to a message. That is not true for fields that are messages, where the absence of the field truly indicates the value is absent and is not just a default or unset value. The message `OptionalString`, for example, is used in the API to indicate whether a character string value is truly present. Thus RequestModelsMeta has a `model_id` field that indicates whether the request is for all models, when the field has not been set, or for a specific one, when the field has been set.

Throughout this specification, the following types are used for identifiers: * `var_id` is int32 * `model_id` is string * `record_id` is int64

This specification conforms to Protocol Buffers Version 3.

Messages

BookmarkIntervalContent

A range of record identifiers can specify the content of a bookmark. Bookmark interval content provides a convenient means to bookmark a contiguous selection of records in a model.

Both fields in this message are optional:

- If neither field is present, the bookmark interval designates all records in the model.
- If only `first_record` is present, the bookmark interval designates all records starting from that record identifier.
- If only `last_record` is present, the bookmark interval designates all records ending at that record identifier. For a dynamic model, such a bookmark interval includes all “future” records.
- If both fields are present, the bookmark interval designates all records between the two identifiers, inclusively.

Field	Type	Label	Description
first_record	int64	optional	[semantically optional] The identifier for the first record in the interval.
last_record	int64	optional	[semantically optional] The identifier for the last record in the interval.

BookmarkMeta

A bookmark is metadata defining a subset of records in a model.

There are three options for specifying a bookmark:

1. Interval content specifies a range of records in the bookmark.
2. Set content specifies a list of records in the bookmark.
3. A filter expression defines a set of logical conditions for determining whether a record is in the bookmark.

Exactly one of *`interval`*, *`set`*, or *`filter`* must be specified in this message.

Field	Type	Label	Description
bookmark_id	string	optional	[semantically optional] When creating a new bookmark, this field must be empty; the server will create a unique identifier for the bookmark. This identifier uniquely identifies the bookmark *on the particular server*.
bookmark_name	string	optional	[semantically required] a name for the bookmark, which is useful for displaying the bookmark to users; this need not be unique, although it is recommended to be so.
interval	BookmarkIntervalContent	optional	the range of records in the bookmark
set	BookmarkSetContent	optional	the list of records in the bookmark
filter	FilterExpression	optional	logical conditions for defining which records are in the bookmark

BookmarkMetaList

Bookmarks may be grouped into lists (sets).

Field	Type	Label	Description
bookmark_metas	BookmarkMeta	repeated	[semantically optional] the bookmarks in the list

BookmarkSetContent

A list (set) of record identifiers can specify the contents of a bookmark. Bookmark-set content provides a convenient means to bookmark a specific selection of non-continuous records in a model.

Field	Type	Label	Description
record_ids	int64	repeated	[semantically optional] the list of record identifiers in the set

DomainMeta

The domain (set of valid values) for a variable.

There are two options for specifying a domain:

1. An interval specifies a range of values in the domain.
2. A set specifies a list of values in the domain.

Exactly one of *interval* or *set* must be specified in the message.

Field	Type	Label	Description
var_id	int32	optional	[semantically required]
interval	VarInterval	optional	the interval of values in the domain
set	VarSet	optional	the list of values in the domain

DoubleList

A list of real numbers.

Field	Type	Label	Description
values	double	repeated	[semantically required] the real numbers

FilterExpression

A filtering expression is a composition of logical conditions on a record. It can be used to filter records. There are four options for specifying a filter expression:

1. The logical negation of another filtering expression
2. The set union of multiple filtering expressions
3. The set intersection of multiple filtering expressions
4. Particular values of variables in a record.

Exactly one of *filter_not*, *filter_union*, *filter_intersection*, or *filter_domain* must be specified in this message.

Field	Type	Label	Description
filter_not	FilterNot	optional	logical negation of an expression
filter_union	FilterUnion	optional	set union of expressions
filter_intersection	FilterIntersection	optional	set intersection of expressions
filter_domain	DomainMeta	optional	particular values of variables

FilterIntersection

Set intersection of filtering expressions. A record satisfies this expression if it satisfies all `filter_expressions`.

Field	Type	Label	Description
filter_expressions	FilterExpression	repeated	[semantically required] the expressions to be intersected

FilterNot

Logically negate a filtering expression. A record satisfies this expression if it does not satisfy `filter_expression`.

Field	Type	Label	Description
filter_expression	FilterExpression	optional	[semantically required] the expression to be negated

FilterUnion

Set union of filtering expressions. A record satisfies this expression if it satisfies any of `filter_expressions`.

Field	Type	Label	Description
filter_expressions	—	repeated	[semantically required] the expressions to be “unioned”

IntegerList

A list of integers.

Field	Type	Label	Description
values	sint64	repeated	[semantically required] The integers

ModelMeta

Metadata for a model.

Field	Type	Label	Description
model_id	string	optional	[semantically required] the unique identifier for the model *on the particular server*
model_name	string	optional	[semantically required] a name for the model, useful for display the model to users; this need not be unique, although it is recommended to be so.
model_uri	string	optional	[semantically required] the unique URI for the model; additional metadata may be obtained by dereferencing that URI.
variables	VarMeta	repeated	[semantically required] metadata for the variables
inputs	DomainMeta	repeated	[semantically optional] metadata for input values to the model, if any

ModelMetaList

A list of metadata for models.

Field	Type	Label	Description
models	ModelMeta	repeated	[semantically optional] the metadata for the models

OptionalInt32

Wrapper for an optional signed integer.

Field	Type	Label	Description
value	int32	optional	[semantically required] the signed integer value

OptionalString

Wrapper for an optional string.

Field	Type	Label	Description
value	string	optional	[semantically required] the character string value

OptionalUInt32

Wrapper for an optional unsigned integer.

Field	Type	Label	Description
value	uint32	optional	[semantically required] the unsigned integer value

Record

A record is a list of variables and their associated values.

Field	Type	Label	Description
record_id	int64	optional	[semantically required] a unique identifier for the record
variables	VarValue	repeated	[semantically optional] the values for variables in the record

RecordData

A collection of records.

There are two options for specifying record data:

1. A list specifies a heterogeneously typed list.
2. A table specifies a homogeneously typed table.

Exactly one of *`list`* or *`table`* must be present in the message.

Field	Type	Label	Description
list	RecordList	optional	a heterogeneously typed list of records
table	RecordTable	optional	a homogeneously typed table of records

RecordList

A list of records. The list is heterogeneous in the sense that each variable may have a different type.

Field	Type	Label	Description
records	Record	repeated	[semantically optional] The list of records.

RecordTable

A homogeneously typed table of records, where each variable has each type, with a row for each record and a column for each variable.

This message represents the following table:

Record Identifier	`var_id[0]`	`var_id[1]`	`...`	`var_id[N]`
rec_id[0]	list[0][0]	list[0][1]	. . .	list[0][N]
rec_id[1]	list[1][0]	list[1][1]	. . .	list[1][N]
. . .	. . .	. . .	. . .	. . .
rec_id[M]	list[M][0]	list[M][1]	. . .	list[M][N]

The underlying list is a **single** array, addressable using the following row-major index formula list[row][var] = array[var + NY * row] where `NX` = length of `rec_ids` and `NY` = length of `var_ids`.

Exacly one of *`reals`*, *`integers`*, or *`strings`* must be specified in the message.

Field	Type	Label	Description
var_ids	int32	repeated	[semantically required] the identifiers of the variables (columns) in the table
rec_ids	int64	repeated	[semantically required] the identifiers of the records (rows) in the table
reals	DoubleList	optional	the real numbers comprising the values of the variables, in row-major order
integers	IntegerList	optional	the integers comprising the values of the variables, in row-major order
strings	StringList	optional	the character strings comprising the values of the variables, in row-major order

Request

A request. There are six types of requests:

Request	Response
Metadata for model(s)	ModelMetaList
Data records	RecordData
Metadata for bookmark(s)	BookmarkMetaList
Saving a bookmark	BookmarkMetaList
Canceling a previous request	n/a
New work, such as a simulation	RecordData

Exactly one of `models_metadata`, `records_data`, `bookmark_meta`, `save_bookmark`, `cancel`, or `work` must be specified in the message.

Field	Type	Label	Description
version	uint32	optional	[semantically required] the version number for the API; *this must be the number **four**.*
id	OptionalUInt32	optional	[semantically optional, but recommended] an identifier that will be used to tag responses, so that responses can be correlated with requests
subscribe	bool	optional	[semantically optional] whether to continue receiving responses indefinitely, as new records become available; this is useful, for example, when a sensor is reporting measurements periodically or when simulations are reporting a series or results. Use RequestCancel to end the subscription.
models_metadata	RequestModelsMeta	optional	request metadata for model(s)
records_data	RequestRecordsData	optional	request data records
bookmark_meta	RequestBookmarkMeta	optional	request metadata for bookmark(s)
save_bookmark	RequestSaveBookmark	optional	request save a new bookmark or update an existing one
cancel	RequestCancel	optional	request to cancel a previous request)
work	RequestWork	optional	request work (e.g., simulation results)

RequestBookmarkMeta

A request for one or more bookmarks for a model.

The response to this request is BookmarkMetaList

Field	Type	Label	Description
model_id	string	optional	[semantically required] which model for which to list bookmarks
bookmark_id	OptionalString	optional	[semantically optional] If empty, list all bookmarks for the model. Otherwise, list just the bookmark metadata for this specific bookmark identifier.

RequestCancel

Cancel a previous request.

Field	Type	Label	Description
id	OptionalUInt32	optional	[semantically required] which request to cancel

RequestModelsMeta

A request for metadata about model(s).

The response to this request is ModelMetaList.

Field	Type	Label	Description
model_id	OptionalString	optional	[semantically optional] If absent, the request is for metadata for all models. Otherwise, the request is for the specifically identified model.

RequestRecordsData

Request record data for a model.

There are three options for requesting record data:

1. Request all records.
2. Request records in a bookmark.
3. Filter records according to a criterion.

The response to this request is RecordData.

No more than one of *bookmark_id* or *expression* may be present in the message.

Field	Type	Label	Description
model_id	string	optional	[semantically required] the identifier for the model
max_records	uint64	optional	[semantically optional] If specified, this is the maximum number of records to return. Otherwise, all records are returned, although they may be returned as multiple responses, each with a chunk of records.
var_ids	int32	repeated	[semantically optional] which variables to include in the response; if this is not specified, all variables will be included.
bookmark_id	string	optional	[semantically optional] Only respond with records in a specified bookmark.
expression	FilterExpression	optional	[semantically optional] Only respond with records matching a specified criterion.

RequestSaveBookmark

A request to create or update a bookmark.

The response to this request is BookmarkMetaList.

Field	Type	Label	Description
model_id	string	optional	[semantically required] which model for which to save the bookmark
new_bookmark	BookmarkMeta	optional	[semantically optional] If empty, create a new bookmark. (In which case, leave the `bookmark_id` empty, so that the server will create a unique identifier for the new bookmark.) Otherwise, update an existing bookmark.

RequestWork

Request that the server compute new records based on input values.

The response to this request is RecordData.

Field	Type	Label	Description
model_id	string	optional	[semantically required] the identifier for the model
inputs	VarValue	repeated	[semantically optional] which input variables to set to which values

Response

A response to a request.

Note that a server may send multiple responses to a single request, expressed as a linked list of chunks. It is strongly recommended that servers chunk by `record_id` so that each record is kept intact. A chunk may be empty.

Field	Type	Label	Description
version	uint32	optional	[semantically required] the version number for the API; *this must be the number **four**.*
id	OptionalUInt32	optional	[semantically optional] a response without an identifier is a notification; otherwise, the response identifier matches the response identifier for the original request.
chunk_id	int32	optional	[semantically optional, but recommended] the identifier for this chunk; it is recommended that chunks are numbered sequentially starting beginning with the number one.
next_chunk_id	int32	optional	[semantically optional] the identifier of the next chunk, or zero if this is the last chunk
error	string	optional	an error message
models	ModelMetaList	optional	a list of model metadata
data	RecordData	optional	a list of record data
bookmarks	BookmarkMetaList	optional	a list of bookmark metadata

StringList

A list of character strings.

Field	Type	Label	Description
values	string	repeated	[semantically required] the character strings

Value

Value that may be a real number, an integer, or a character string

Exactly one of *`real_value`*, *`integer_value`*, or *`string_value`* must be specified in this message.

Field	Type	Label	Description
real_value	double	optional	the real number
integer_value	int64	optional	the integer
string_value	string	optional	the character string

VarInterval

A range of values of a variable.

Both fields in this message are optional:

- If neither field is present, the interval designates all values in the domain.
- If only `first_value` is present, the interval designates all values starting from that value.
- If only `last_value` is present, the bookmark interval designates all values ending at that value.
- If both fields are present, the interval designates all values between the two values, inclusive.

Field	Type	Label	Description
first_value	Value	optional	[semantically optional] the first value in the interval
last_value	Value	optional	[semantically optional] the last value in the interval

VarMeta

Metadata for a variable.

Field	Type	Label	Description
var_id	int32	optional	[semantically required] an integer identifying the variable
var_name	string	optional	[semantically required] the name of the variable
units	string	optional	[semantically optional] the name of the unit of measure for values of the variable
si	sint32	repeated	[semantically optional] the unit of measure expressed as a list of the exponents for the eight fundamental SI quantities [meter, kilogram, second, ampere, kelvin, mole, candela, radian]; for example, the unit of acceleration m/s^2 would be express as `[1, 0, -2, 0, 0, 0, 0, 0]` because meters has an exponent of positive one and seconds has an exponent of negative two.
scale	double	optional	[semantically optional] An overall scale relative to the fundamental SI scale of the unit of measure; for instance, kilometers would have a scale of 1,000 because the fundamental unit of distance is meters.
type	VariableType	optional	[semantically optional] the data type for values of the variable; The default type is real number.

VarSet

A set of values for a variable.

Field	Type	Label	Description
elements	Value	repeated	[semantically optional] the list of values in the set

VarValue

The value of a variable.

Field	Type	Label	Description
var_id	int32	optional	[semantically required] the identifier for the variable
value	Value	optional	[semantically required] the value of the variable

VariableType

The data type for a value.

Name	Number	Description
REAL	0	a real number
INTEGER	1	an integer
STRING	2	a character string

Scalar Value Types

.proto Type	Notes	C++ Type	Java Type	Python Type
double		double	double	float
float		float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers—if your field is likely to have negative values, use sint32 instead.	int32	int	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers; if your field is likely to have negative values, use sint64 instead.	int64	long	int/long
uint32	Uses variable-length encoding.	uint32	int	int/long
uint64	Uses variable-length encoding.	uint64	long	int/long
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long
sfixed32	Always four bytes.	int32	int	int
sfixed64	Always eight bytes.	int64	long	int/long
bool		bool	Boolean	Boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str

Implementations

This section provides an overview of the variety of libraries and applications implementing the Records API (see the table below). In particular, pre-built applications are available for serving text-based data sources, database queries, and sensor data feeds. Application Container Images (ACIs) (CoreOS 2017a) of each have been packed for use with the rkt container engine (CoreOS 2017b).

Available Client and Server Applications and Libraries for the Records API

Client or Server ?	Library or Application ?	Data Source	Implementation Language	Computing Platforms	URL
client	GUI application	any	C++	Mac, Winodws, Linux	https://github.nrel.gov/d-star/cpp-records
server	GUI/CLI applications	CSV files	C++	Mac, Winodws, Linux	https://github.nrel.gov/d-star/cpp-records
client	library	any	Haskell	Mac, Windows, Linux	https://github.com/NREL/AESD/lib/haskell
server	CLI application	TSV files	Haskell	Mac, Windows, Linux	https://github.com/NREL/AESD/lib/haskell
server	CLI application	PostgreSQL	Haskell	Mac, Windows, Linux	https://github.com/NREL/AESD/lib/haskell
server	CLI application	MySQL	Haskell	Mac, Windows, Linux	https://github.com/NREL/AESD/lib/haskell
server	CLI application	SQLite3	Haskell	Mac, Windows, Linux	https://github.com/NREL/AESD/lib/haskell
server	CLI application	ODBC	Haskell	Mac, Windows, Linux	https://github.com/NREL/AESD/lib/haskell
server	CLI application	Haystack	Haskell	Mac, Windows, Linux	https://github.com/NREL/AESD/lib/haskell
client	library, web application	any	JavaScript	Chrome, Firefox	https://github.com/NREL/AESD/lib/javascript
client	library	any	Python	any	https://github.com/NREL/AESD/lib/python
client	library	any	R	any	https://github.nrel.gov/d-star/r-records

Haskell Client and Server Library and Applications

Both client and server applications in Haskell are available for the Records API. Full documentation resides at https://github.com/NREL/AESD/lib/haskell.

Client Library

The client library described below provides the basic functions for interacting with any Records API server.

Types

`data State`

State information for a client.

Entry Point

`clientMain`

Run a client.

Argument Type	Description
`:: String`	the WebSocket host address
`-> Int`	the WebSocket port number
`-> String`	the WebSocket path
`-> (State -> IO ())`	customize the client
`-> IO ()`	action for running the client

`close`

Close a client.

Argument Type	Description
`:: State`	the state of the client
`-> IO ()`	action for closing the client

Server Requests

`fetchModels`

Fetch model metadata.

Argument Type	Description
`:: State`	the state of the client
`-> IO (Either String [ModelMeta])`	action returning either an error or the models

`fetchRecords`

Fetch records from the server.

Argument Type	Description
`:: State`	the state of the client
`-> ModelIdentifier`	the model identifier
`-> Maybe Int`	the maximum number of records to request
`-> IO (Either String [RecordContent])`	action returning either an error or the records

`fetchBookmarks`

Fetch bookmark(s).

Argument Type	Description
`:: State`	the state of the client
`-> ModelIdentifier`	the model identifier
`-> Maybe BookmarkIdentifier`	the bookmark identifier, or all bookmarks
`-> IO (Either String [BookmarkMeta])`	action returning either an error or the bookmark(s)

`storeBookmark`

Save a bookmark.

Argument Type	Description
`:: State`	the state of the client
`-> ModelIdentifier`	the model identifier
`-> BookmarkMeta`	the bookmark metadata
`-> IO (Either String BookmarkMeta)`	action returning either an error or the bookmark

Server Library

The server library provides two options for implementing a Records API server. The `AESD.Records.Server` module provides a main entry point `serverMain`, a type class `ModelManager`, and a monad `ServiceM` that implement a skeletal server, which handles all of the WebSocket communication and Protocol Buffer serialization; an implementer need only create an instance of `ModelManager`. Furthermore, the `AESD.Records.Server.Manager` module provides such an instance `InMemoryManager` of the type class `ModelManger` to handle in-memory caching of data and on-disk persistence of bookmarks; here, an implementer just calls the function `makeInMemoryManager` and provides several functions that retrieve content:

`makeInMemoryManager`

Construct an in-memory model manager.

Argument Type	Description
`:: Maybe FilePath`	the name of the journal file
`-> a`	the initial state
`-> (a -> IO ([ModelMeta], a))`	list models in an action modifying the state
`-> (a -> ModelMeta -> IO ([RecordContent], a))`	load record data in an action modifying the state
`-> (a -> ModelMeta -> [VarValue] -> IO ([RecordContent], a))`	performing work in an action modifying the state
`-> IO (InMemoryManager a)`	action constructing the manager

Server Back Ends

As previously mentioned, prebuilt servers have been implemented for standard types of data sources.

Tab-Separate-Value Files

Serving tab-separated-value (TSV) files is as simple as placing the TSV files in a directory and starting a server at the command line, with the arguments specified in the table below:

```
aesd-file-server <host> <port> <directory> <persistence> <chunkSize>
```

Command-Line Arguments for Serving TSV Files

Parameter	Description
host	host address to which to bind the service
port	port to which to bind the service
directory	directory with TSV files to be served
persistence	filename for bookmark data
chunkSize	number of records return in each chunk

Database Queries

The Records API servers have been implemented for the most common database back ends. Each server takes a single command-line argument specifying a YAML (Oren Ben-Kiki, Clark Evans, Ingy döt Net 2017) configuration file with the parameters in the table below.

Parameters for Database Back Ends Serving the Records API

Parameter	Description	PostgreSQL	MySQL	SQLite3	ODBC
host	host address to which to bind the service	required	required	required	required
port	port to which to bind the service	required	required	required	required
directory	directory with queries to be served	required	required	required	required
persistence	filename for bookmark data	optional	optional	optional	optional
chunkSize	number of records return in each chunk	optional	optional	optional	optional
database	database connection information	required connection string	required connection string	required filename	required connection string

Haystack Sensor Measurements and the "Internet of Things"

Furthermore, a server for Project Haystack (Project Haystack 2017) data feeds, typically sensor measurements from devices in the "internet of things," has been implemented. The server takes command-line arguments specified in the table below.

```
aesd-haystack-server <configuration> <host> <port> <startTime> <persistence>
<chunkSize>
```

Command-Line Arguments for Serving Haystack Data Feeds

Parameter	Description
configuration	YAML configuration file for accessing the Haystack service
host	host address to which to bind the service
port	port to which to bind the service
startTime	earliest time to serve, specified in seconds of the POSIX Epoch
persistence	filename for bookmark data
chunkSize	number of records return in each chunk

The parameters in the YAML configuration file like the one below and are described in the following table:

```
siteAccess     :
  server       : xv11skys01.nrel.gov
  root         : /api/nrel_wt_V7
  authorization: ["my username","my password"]
  secure       : false
  timeZone     : [-360, true, Denver]
siteIdentifier : NWTCv4
siteURI        : http://aesd.nrel.gov/records/v4/nwtc/
siteName       : NREL NWTC
siteDescription: Sensors from NREL National Wind Technology Center
siteTags       :
               ! 'DC.source'     : https://xv11skys01.nrel.gov/proj/nrel_wt_v7
               ! 'DC.creator'    : Brian W Bush <brian.bush@nrel.gov>
               ! 'DC.description': NREL NWTC sensors
siteMeters     :
               - 1dca834e-c6af46d6 NWTC Alstom Turbine Electricity Meter Turbine-Alstom kW Demand Forward
               - 1dca834e-69a3e57e NWTC Alstom Turbine Electricity Meter Turbine-Alstom kW Demand Reverse
               - 1dca834e-f56e11f0 NWTC Alstom Turbine Electricity Meter Turbine-Alstom kWh Delivered Forward
```

YAML Configuration Parameters for Haystack-Based Records API Servers

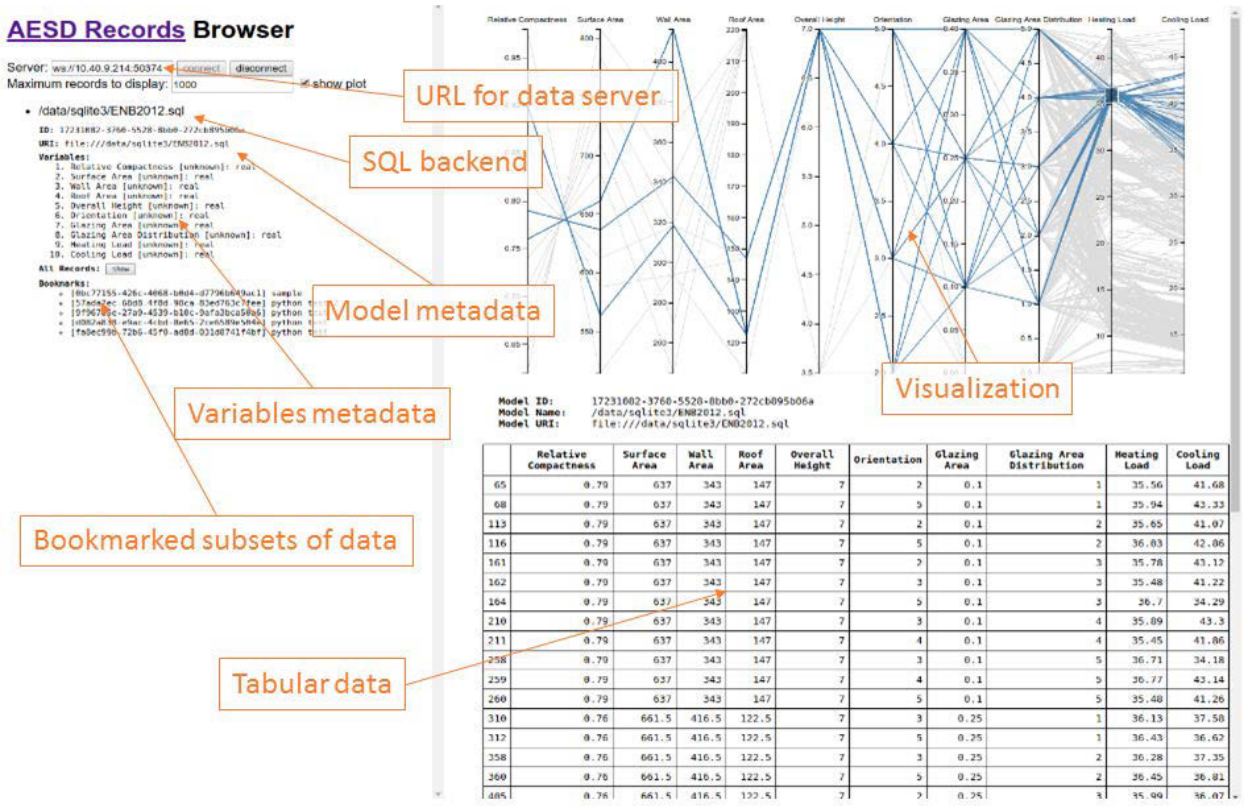
Parameter	Description	Required?
server	hostname and port for the Haystack server	required
root	path to the Haystack REST service	required
authorization	the username and password for accessing the Haystack REST service	optional
secure	whether to use HTTPS instead of HTTP	optional
time zone	time zone information: minutes offset from UTC, whether to use daylight savings time, and the geographic location	required
siteIdentifier	identifier for the Records API server	required
siteURI	URI for the Records API server metadata	required
siteName	name of the Records API server	required
siteTags	key-value pairs tagging the server with additional information	optional
siteMeters	list of meters to expose on the Records API server; the Haystack ID is followed by a space and textual description.	required

C++ Server and Client

Both client and server applications have been implemented in C++ for the Records API. See https://github.nrel.gov/d-star/cpp-records for details. There are GUI and command-line applications for serving comma-separated-value files and a GUI application for browsing Records API data sources.

JavaScript Client Library and Web-Based Browser

The client library for JavaScript relies on a few simple functions to interact with a Records API server. Full documentation for the JavaScript client library is available at http://github.com/NREL/AESD/lib/javascript. The figure below shows the user interface of the general purpose Records API browser using this JavaScript library.



	Relative Compactness	Surface Area	Wall Area	Roof Area	Overall Height	Orientation	Glazing Area	Glazing Area Distribution	Heating Load	Cooling Load
65	0.79	637	343	147	7	2	0.1	1	35.56	41.68
68	0.79	637	343	147	7	5	0.1	1	35.94	43.33
113	0.79	637	343	147	7	2	0.1	2	35.65	41.07
116	0.79	637	343	147	7	5	0.1	2	36.03	42.86
161	0.79	637	343	147	7	2	0.1	3	35.78	43.12
162	0.79	637	343	147	7	3	0.1	3	35.48	41.22
164	0.79	637	343	147	7	5	0.1	3	36.7	34.29
210	0.79	637	343	147	7	3	0.1	4	35.89	43.3
211	0.79	637	343	147	7	4	0.1	4	35.45	41.86
258	0.79	637	343	147	7	3	0.1	5	36.71	34.18
259	0.79	637	343	147	7	4	0.1	5	36.77	43.14
260	0.79	637	343	147	7	5	0.1	5	35.48	41.26
310	0.76	661.5	416.5	122.5	7	3	0.25	1	36.13	37.58
312	0.76	661.5	416.5	122.5	7	5	0.25	1	36.43	36.62
358	0.76	661.5	416.5	122.5	7	3	0.25	2	36.28	37.35
360	0.76	661.5	416.5	122.5	7	5	0.25	2	36.45	36.81

User interface for the Records API browser

Connect to a server

```
connect(wsURL)
```

Here, `wsURL` is simply the URL of the server (e.g., `ws://10.40.9.214:503761`). This returns a connection object.

Disconnect from a server

```
disconnect(connection)
```

Here, `connection` is the connection object returned by the `connect` function.

Retrieve list of data models

```
requestModelsMetadata(connection, modelId, notify, notifyError)
```

Here, `connection` is the connection object returned by the `connect` function and `modelId` is either the string identifying the model or `null` if metadata for all models is requested. After all of the model metadata have been retrieved, the `notify` function is called with the list of model metadata objects as its argument; if an error occurs, `notifyError` is called with the error message as its argument. The function `requestModelsMetadata` returns a result object that contains a field `done` indicating whether all model metadata have been retrieved and a field `models` listing the model metadata retrieved so far.

Retrieve data records

```
requestRecordsData(connection, modelId, maxRecords, variableIds, bookmarkId,
notify, notifyError)
```

Here, `connection` is the connection object return by the `connect` function and `modelId` is the string identifying the model. After all of the data records have been retrieved, the `notify` function is called with the list of data records as its argument; if an error occurs, `notifyError` is called with the error message as its argument. The `maxRecords` argument specifies the maximum number of records to retrieve, `variableIds` may list the variables of interest, and `bookmarkId` restricts the results to bookmarked records. The function `requestRecordsData` returns a result object that contains a field `done` indicating whether all data records have been retrieved and a field `data` listing the data records retrieved so far.

Retrieve list of bookmarks

```
requestBookmarkMeta(connection, modelId, bookmarkId, notify, notifyError)
```

Here, `connection` is the connection object returned by the `connect` function, `modelId` is the string identifying the model, and `bookmarkId` is either the string identifying the bookmark or `null` if metadata for all bookmarks is requested. After all of the bookmark metadata have been retrieved, the `notify` function is called with the list of bookmark metadata as its argument; if an error occurs, `notifyError` is called with the error message as its argument. The function `requestBookmarkMeta` returns a result object that contains a field `done` indicating whether all bookmark metadata have been retrieved and a field `bookmarks` listing the bookmark metadata retrieved so far.

Create/update a bookmark

```
requestSaveBookmark(connection, modelId, name, filter, notify, notifyError)
```

Here, `connection` is the connection object returned by the `connect` function, `modelId` is the string identifying the model, and `bookmarkId` is either null for a new bookmark or the identifier for a bookmark being updated. The `name` field names the bookmark and the `filter` object describing the filtering operation for the bookmark. After the bookmark metadata has been created or updated, the `notify` function is called with the list of bookmark metadata as its argument; if an error occurs, `notifyError` is called with the error message as its argument. The function `requestSaveBookmark` returns a result object that contains a field `done` indicating whether all bookmark metadata have been retrieved and a field `bookmarks` listing the bookmark metadata retrieved so far.

Python Client Library

Full documentation for the Python client library is available at http://github.com/NREL/AESD/lib/python.

Client API

new_server(self, server_url)

```
Change server URL to which WebSocket will connect
Parameters
----------
server_url : 'string'
    server url

Returns
---------
self.url : 'string'
    server url
```

send(self, request)

```
Closes event_loop
Parameters
----------
request : 'proto.request'
    proto request message
timeout : 'int'
    timeout in seconds for connection

Returns
---------
response : 'list'
    List of responses from the server, each response is a proto message
```

get_model_info(self, model_id)

```
Sends request of model metadata and extracts response
Parameters
----------
model_id : 'string'
    Id of model for which to request models_metadata
    if None requests all models

Returns
-------
model_info : 'list'|'dict'
    List of model's metadata dictionaries for each model in models or
    dictionary for model_id
```

get_data(self, model_id, max_records=1000, variable_ids=None, bookmark_id=None)

```
Sends request of model metadata and extracts response
Parameters
----------
```

```
model_id : 'string'
    Id of model for which to request records_data
max_records : 'int'
    Number or records being request (0 will return all records)
variable_ids : 'list'
    List of variable ids (ints) to be requested
    Will be returned in same order as request
    Default=None, all variables will be returned (order?)
bookmark_id : 'int'
    Request records_data based on bookmark id

Returns
-------
data : 'pd.DataFrame'
    Concatenated data from each response message
    Variable ids replaced with names from model_info
```

do_work(self, model_id, inputs)

```
Sends request of model metadata and extracts response
Parameters
----------
model_id : 'string'
    Id of model for which to request records_data
inputs : 'dict'
    Dictionary of {var_id: value} pairs

Returns
-------
data : 'pd.DataFrame'
    Concatenated data from each response message
    Variable ids replaced with names from model_info
```

get_bookmark_info(self, model_id, bookmark_id)

```
Sends request of model metadata and extracts response
Parameters
----------
model_id : 'string'
    Id of model for which to request bookmark_meta
bookmark_id : 'string'
    Id of bookmark for which to request models_metadata
    if None request all bookmarks

Returns
-------
model_info : 'list'|'dict'
    List of model's metadata dictionaries for each model in models or
    dictionary for model_id
```

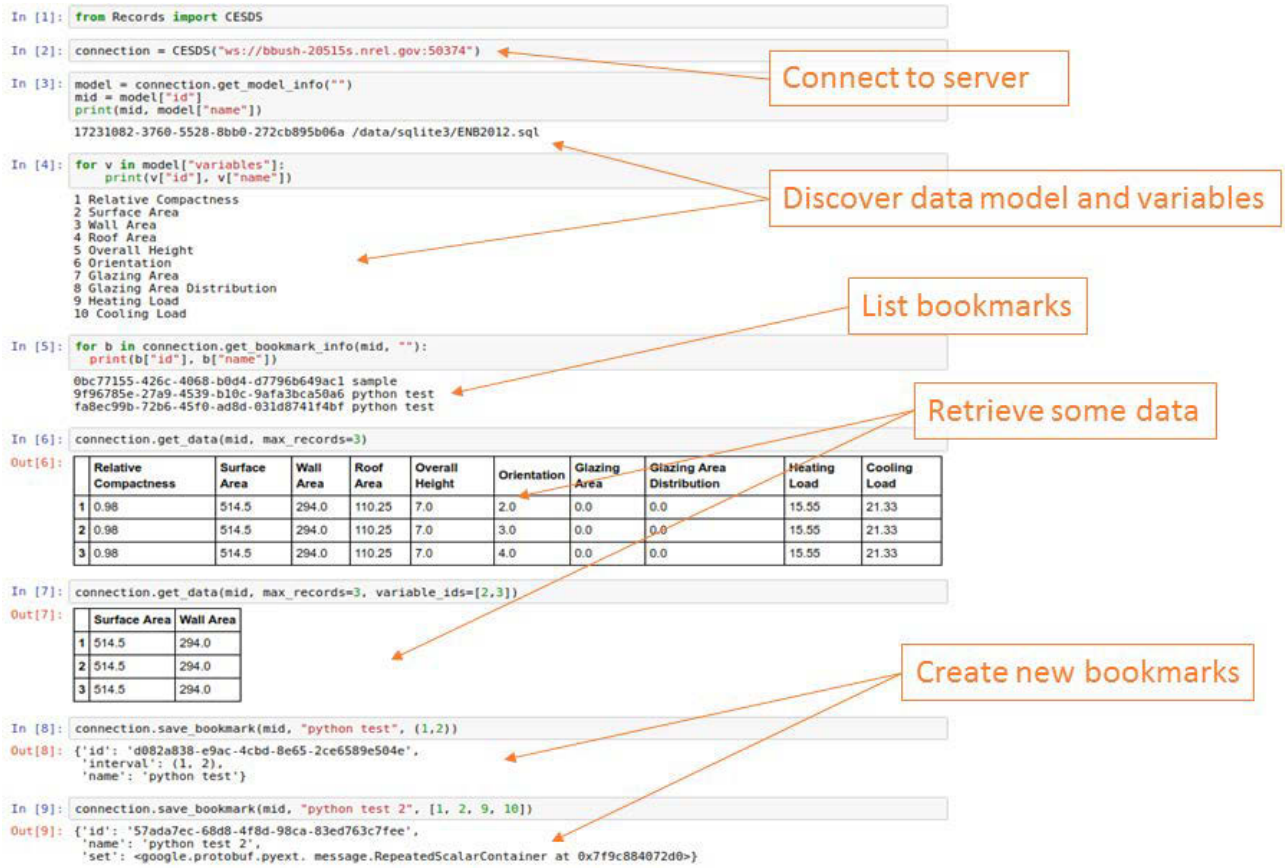
```
save_bookmark(self, model_id, name, content)

Sends request to save new bookmark
Parameters
----------
model_id : 'string'
    Id of model for which to request bookmark_meta
name : 'string'
    Name for new bookmark
content : 'list'|'tuple'
    Contents of bookmark
    list is a bookmark set
    tuple is a bookmark interval

Returns
-------
model_info : 'list'|'dict'
    List of model's metadata dictionaries for each model in models or
    dictionary for model_id
```

Example

The figure below shows example usage of the Python Records API client.



Example of a Python session using the Records API

Appendix

Protocol Buffers for Records API Version 4

```
syntax = "proto3";
package AesdRecords;

option optimize_for = LITE_RUNTIME;

message OptionalInt32 {
    int32 value = 1; /// [semantically required]
}

message OptionalUInt32 {
    uint32 value = 1; /// [semantically required]
}

message OptionalString {
    string value = 1; /// [semantically required]
}

message Value {
    oneof    value              /// [semantically required]
    {
      double real_value    = 1;
      int64  integer_value = 2;
      string string_value  = 3;
    }
}

message DoubleList {
    repeated double values = 1; /// [semantically required]
}

message IntegerList {
    repeated sint64 values = 1; /// [semantically required]
}

message StringList {
    repeated string values = 1; /// [semantically required]
}

message BookmarkIntervalContent {
    int64 first_record = 1; /// [semantically optional]
    int64 last_record  = 2; /// [semantically optional]
}

message BookmarkSetContent {
    repeated int64 record_ids = 1; /// [semantically optional]
```

```
}

message BookmarkMeta {
    string                      bookmark_id   = 1; /// [semantically optional
]
    string                      bookmark_name = 2; /// [semantically required
]
    oneof                       content           /// [semantically required
]
    {
        BookmarkIntervalContent interval      = 3;
        BookmarkSetContent      set           = 4;
        FilterExpression        filter        = 5;
    }
}

message BookmarkMetaList {
    repeated BookmarkMeta bookmark_metas = 1; /// [semantically optional]
}

message RequestBookmarkMeta {
    string         model_id    = 1; /// [semantically required]
    OptionalString bookmark_id = 2; /// [semantically optional]
}

message RequestSaveBookmark {
    string       model_id     = 1; /// [semantically required]
    BookmarkMeta new_bookmark = 2; /// [semantically optional]
}

message FilterExpression {
  oneof                expression               /// [semantically required]
  {
    FilterNot          filter_not          = 1;
    FilterUnion        filter_union        = 2;
    FilterIntersection filter_intersection = 3;
    DomainMeta         filter_domain       = 4;
  }
}

message FilterNot {
  FilterExpression filter_expression = 1; /// [semantically required]
}

message FilterUnion {
  repeated FilterExpression filter_expressions = 1; /// [semantically require
d]
}
```

```
message FilterIntersection {
  repeated FilterExpression filter_expressions = 1; /// [semantically require
d]
}

enum VariableType
{
    REAL            = 0;
    INTEGER         = 1;
    STRING          = 2;
}

message VarMeta {
    int32           var_id   = 1; /// [semantically required]
    string          var_name = 2; /// [semantically required]
    string          units    = 3; /// [semantically optional]
    repeated sint32 si       = 4; /// [semantically optional]
    double          scale    = 5; /// [semantically optional]
    VariableType    type     = 6; /// [semantically optional]
}

message ModelMeta {
    string              model_id   = 1; /// [semantically required]
    string              model_name = 2; /// [semantically required]
    string              model_uri  = 3; /// [semantically required]
    repeated VarMeta    variables  = 4; /// [semantically required]
    repeated DomainMeta inputs     = 5; /// [semantically optional]
}

message ModelMetaList {
    repeated ModelMeta models = 1; /// [semantically optional]
}

message RequestModelsMeta {
    OptionalString model_id = 1; /// [semantically optional]
}

message VarInterval {
    Value first_value = 1; /// [semantically optional]
    Value last_value  = 2; /// [semantically optional]
}

message VarSet {
    repeated Value elements = 1; /// [semantically optional]
}

message DomainMeta {
    int32           var_id   = 1; /// [semantically required]
    oneof           domain        /// [semantically required]
```

```
    {
        VarInterval interval = 2;
        VarSet      set      = 3;
    }
}

message RequestWork {
    string model_id          = 1; /// [semantically required]
    repeated VarValue inputs = 2; /// [semantically optional]
}

message VarValue {
    int32 var_id = 1; /// [semantically required]
    Value value  = 2; /// [semantically required]
}

message Record {
    int64    record_id          = 1; /// [semantically required]
    repeated VarValue variables = 2; /// [semantically optional]
}

message RecordList {
    repeated Record records = 1; /// [semantically optional]
}

message RecordTable {
    repeated int32  var_ids    = 1; /// [semantically required]
    repeated int64  rec_ids    = 2; /// [semantically required]
    oneof           list            /// [semantically required]
    {
        DoubleList  reals      = 3;
        IntegerList integers   = 4;
        StringList  strings    = 5;
    }
}

message RecordData {
    oneof           style      /// [semantically required]
    {
        RecordList  list  = 1;
        RecordTable table = 2;
    }
}

message RequestRecordsData {
    string               model_id    = 1; /// [semantically required]
    uint64               max_records = 2; /// [semantically optional]
    repeated int32       var_ids     = 3; /// [semantically optional]
    oneof                filter           /// [semantically optional]
```

```
    {
        string           bookmark_id = 4; /// [semantically optional]
        FilterExpression expression  = 5; /// [semantically optional]
    }
}

message Response {
    uint32               version       = 1; /// [semantically required]
    OptionalUInt32       id            = 2; /// [semantically optional]
    int32                chunk_id      = 3; /// [semantically optional, but r
ecommended]
    int32                next_chunk_id = 4; /// [semantically optional]
    oneof                type               /// [semantically optional]
    {
        string           error         = 5;
        ModelMetaList    models        = 6;
        RecordData       data          = 7;
        BookmarkMetaList bookmarks     = 8;
    }
}

message RequestCancel {
    OptionalUInt32 id = 1; /// [semantically required]
}

message Request {
    uint32                   version         = 1; /// [semantically required]
    OptionalUInt32           id              = 2; /// [semantically optional,
but recommended]
    bool                     subscribe       = 3; /// [semantically optional]
    oneof                    type                 /// [semantically required]
    {
        RequestModelsMeta    models_metadata = 4;
        RequestRecordsData   records_data    = 5;
        RequestBookmarkMeta  bookmark_meta   = 6;
        RequestSaveBookmark  save_bookmark   = 7;
        RequestCancel        cancel          = 8;
        RequestWork          work            = 9;
    }
}
```

References

CoreOS. 2017a. “App Container Basics: Coreos.” Accessed September 6, 2017. https://coreos.com/rkt/docs/latest/app-container.html.

CoreOS. 2017b. “Rkt Container Engine with Coreos.” Accessed September 6, 2017. https://coreos.com/rkt.

Fowler, Martin. 2017. “UML Distilled.” Accessed April 11, 2017. http://my.safaribooksonline.com/book/software-engineering-and-development/uml/0321193687/sequence-diagrams/ch04.

Google Developers. 2017a. “Protocol Buffers | Google Developers.” Accessed April 11, 2017. https://developers.google.com/protocol-buffers/.

Google Developers. 2017b. “Protocol Buffers: Google’s Data Interchange Format.” Accessed April 11, 2017. https://github.com/google/protobuf/blob/master/README.md.

Internet Engineering Task Force. 2017. “RFC 6455 - The Websocket Protocol.” Accessed April 11, 2017. https://tools.ietf.org/html/rfc6455.

Oren Ben-Kiki, Clark Evans, Ingy döt Net. 2017. “YAML Specification Index.” Accessed September 6, 2017. http://www.yaml.org/spec/.

Project Haystack. 2017. “Home - Project Haystack.” Accessed September 6, 2017. http://project-haystack.org/.