ProgSnap 2 — A standardized representation for programming process data

Austin Cory Bart, University of Delaware
Brett Becker, University College Dublin
Luke Gusukuma, Virginia Tech
David Hovemeyer, Johns Hopkins University
Ayaan Kazerouni, Virginia Tech
Andrew Petersen, University of Toronto
Thomas Price, North Carolina State University
Kelly Rivers, Carnegie Mellon University

Document information

Specification Version: 8

Specification Date: 18 December 2020

Specification Status: Beta

This specification has now been implemented in multiple datasets and analyses and development on the spec has slowed significantly. We broadly invite interested parties to use the specification at this point. Details may still change in the future, but the probability of changes being non-backwards-compatible is low.

Change Log

Version 8, 18 December 2020:

- Officially moving to Beta status

Version 7, 21 August 2020:

- Clarification of the CodeStateID column in the Code tables
- Added Entity-Relationship diagram to appendix, referenced in Dataset Representation section.

Version 6, 31 July 2019:

- README.txt is required; clarified expectations for its contents
- Clarification of when Project.* events should be generated
- Added File.Copy event type
- Added ProjectID optional column
- Eliminated FilePath column, clarified the purpose and meaning of the CodeStateSection column
- Added DestinationCodeStateSection column (for specifying target of File.Rename and File.Copy events)
- Guidance on how SourceLocation should be interpreted for Edit.* events
- Clarification of meaning of Score column, and changed from required to recommended
- Separated Optional columns into Recommended and Event-Specific columns
- Changed CodeStateSection to be required for File.*, Compile, and Compile.* events
- Added a new recommended column, LoggingErrorID, for tracking errors in data
- Merged the EditTrigger column into the EventInitiator column, and updated EventIniator's enum values.
- Required that all Interventions have an EventIntiator value
- Changed ProgramInput and ProgramOutput from Required to Recommended for Compile events
- Clarified the description of Intervention events
- Allow hierarchical organization of code states when using the Directory format
- Deduplication of code states in the Directory format is recommended, but not required

Version 5, 17 April 2019: Added Real data type and Score/ExtraCreditScore columns.

Version 4, 29 March 2019: First Alpha version.

Table of contents

Introduction

This document describes ProgSnap 2, a standardized representation for “programming snapshot” data. Its intended purpose is to facilitate analysis of datasets representing student work on programming exercises and assignments.

Data that can be represented as part of a ProgSnap 2 dataset includes (but is not limited to):

- File contents and changes to file contents (edits)
- Compilation events
- Compiler errors and warnings
- Program execution events
- Test execution events and test results
- Interventions such as generated hints

ProgSnap 2 is based on the “DATASTAND Group Notes” document created by John Stamper, Stephen Edwards, Andrew Petersen, Thomas Price, and Ian Utting at ICER 2017.

Dataset representation

A ProgSnap 2 dataset will include two types of files: *metadata files* and *payload files*. Payload files contain data originating directly from student work. Metadata files contain data describing student work. For example, the main event table is a metadata file.

The dataset consists of a primary directory that contains three main subdirectories: CodeStates, LinkTables, and Resources. The primary data files, MainTable.csv and DatasetMetadata.csv, along with the descriptive README.txt file, are located in the primary directory, while additional files are located in their respective subdirectories.

Appendix A has an Entity-Relationship diagram that provides a structural overview.

File formats

The primary file format used for metadata files is CSV (comma-separated values.)

All CSV-format metadata files in a ProgSnap 2 dataset are required to conform to RFC 4180. In addition, they are required to be encoded using the UTF-8 character set, and they are required to include a header row. So, the effective MIME type for a CSV-format metadata file is

```
text/csv; charset=utf-8; header=present
```

Note that there is no requirement that the columns of a metadata file occur in any particular order. The mandatory header will be used to specify the column ordering.

Data types

The following data types are used in metadata files.

ID

An ID value is an identifier for any data that can be referenced, such as events, sessions, courses, etc. An ID value is formed by any sequence of Unicode characters, up to a maximum length of 1000 characters.

Two ID values are equal if they consist of exactly the same sequence of characters. No ordering is implied by ID values: the only meaningful comparison between two ID values is for equality or inequality.

Although an ID value can be any sequence of characters (up the the specified maximum length), we recommend that data providers avoid using whitespace characters other than space (U+0020), and in general refrain from using “unusual” character codes such as combining characters, non-printing characters, emojis, etc. However, data consumers should be prepared to accept any arbitrary character sequence as an ID value.

The intent of allowing free-form strings as ID values is to permit them to be self-descriptive: for example, “Fall 2018” could be used as a value in the TermID column.

Integer

An Integer value is a textual base 10 representation of an integer in the range $-2^{63}..2^{63}-1$ inclusive.

Real

A Real value is a textual base 10 representation of a real number. Three formats are allowed: *integer*, *decimal*, and *scientific notation*.

A value in the *integer* format consists of a sequence of one or more decimal digits. The value may be preceded by an optional minus sign (“-”) to indicate a negative value.

A value in the *decimal* format consists of a sequence of one or more decimal digits, followed by a decimal point (“.”), followed by a sequence of zero or more decimal digits. The value may be preceded by an optional minus sign (“-”) to indicate a negative value.

A value in the *scientific notation* format is a mantissa, followed by “E” or “e”, followed by an exponent. The mantissa has the form of either an *integer* or *decimal* as described above. The exponent starts with either plus (“+”) or minus (“-”), followed by a sequence of one or more decimal digits. If the mantissa is negative, then the overall value is negative.

Examples:

Example	**Format**	**Comments**
123	Integer	
-123	Integer	Negative value
1.23	Decimal	
-1.23	Decimal	Negative value
1.23e+3	Scientific notation	Equal to 1230
1.23e-3	Scientific notation	Negative exponent; equal to 0.00123
-1.23e+3	Scientific notation	Negative value; equal to -1230
-1.23e-3	Scientific notation	Negative value, negative exponent; equal to -0.00123

Real values should be restricted to the range allowed for IEEE 754 double precision values. Note that NaN (not a number) and +Inf/-Inf (positive and negative infinity) are *not* allowed as Real values.

The representations of Real values are intended to be directly parsed as floating-point values by most programming languages, runtime libraries, and statistical computing applications.

Boolean

A Boolean value is either `true` or `false`.

Timestamp

A Timestamp value is an ISO 8601 datetime value specifying a date and local time, *without time zone*. Example:

    2018-09-07T08:41:02

Timezone

A Timezone value is an ISO 8601 time zone offset. Example:

-0500

Enum

An Enum (enumerated) value is one of a predefined set of possible values. For the main event table columns, the specific set of possible values will be listed in the “Enum values” section of the column description.

String

A String value is a sequence of 0 or more characters.

NonemptyString

A NonemptyString value is a sequence of 1 or more characters.

URL

A URL value is either

- A valid URL conforming to RFC 3986
- An “internal” URL specifying a file or directory local to the dataset

The format for an “internal” URL is

```
file:relativePath
```

where *relativePath* is the relative path of a file within the ProgSnap 2 dataset, meaning a sequence of path components, separated by slash (“/”) characters, terminating in a file or directory. Internal URLs will typically specify the path to a file in the Resources directory. As an example, the following internal URL might designate a course description document:

```
file:Resources/CS101_Description.pdf
```

RelativePath

A RelativePath identifies a specific file or resource within a CodeState. Its value should be a sequence of path components, separated by slash (“/”) characters, terminating in a file. A RelativePath should not contain any occurrences of the “.” or “..” path components (which in Unix-like systems indicate the current and parent directories.)

As an example, assume that a CodeState (collection of files and resources associated with an event) has a subdirectory called “src” with files “foo.c” and “bar.c”. The RelativePath values identifying those files within the CodeState would be

```
src/foo.c
```

and

```
src/bar.c
```

respectively.

SourceLocation

A SourceLocation value indicates a location in a source code artifact. A SourceLocation starts with either “Text:” or “Tree:”, followed by a sequence of one or more integer values separated by colon (‘:’) characters.

For text-based source languages, a SourceLocation starts with “Text:”, which will be followed by either 1 or 2 integer values, where the first value indicates a line number (1 being the first line in the source file), and the (optional) second value indicates a character position within the identified line (1 being the position of the first character in the line.) For example, the SourceLocation

```
Text:13
```

would indicate line 13 of a source file, and the SourceLocation

```
Text:13:6
```

would indicate the sixth character of line 13 of a source file.

Note that a single SourceLocation represents a single “point” in the source code. To represent a range in code, two SourceLocations should be used; one for the beginning of the range, and one for the end.

For non-text-based languages (e.g., block languages) where the source representation can be mapped to a tree structure, the SourceLocation starts with “Tree:”, and is followed by a colon-separated sequence of integers describing a path from the root of the tree to the feature of interest. The empty path indicates the root, so the SourceLocation

```
Tree:
```

would identify the root node. Integers in the path represent the ordinal of a child node, with 1 being the first child, 2 being the second, etc. So, the SourceLocation

Tree:1:3

would indicate the third child of the first child of the root node.

README.txt

Every ProgSnap 2 dataset is required to have a plain text file named README.txt in its top-level directory. This file must contain contact information (at least name and email address) for someone who can answer questions about the dataset. In addition to the required contact information, it is strongly recommended that README.txt contain the following information (if applicable):

- High-level overview of the dataset and how it was collected
- Information about the context (e.g., course) in which the dataset was collected
- Relevant papers
- Unique properties of the data
- How to interpret any columns with ambiguous values (e.g. CodeStateSelection)
- Details about tools used (to clarify the meaning of values used in the ToolInstances column of the main event table)
- Details about user-defined columns, values, and data types (those with names beginning with “X-”)

We encourage producers of datasets to consider including contextual information recommended by csedresearch.org.

Dataset Metadata

Every ProgSnap 2 dataset is required to have a metadata file named

```
DatasetMetadata.csv
```

The purpose of this metadata file is to describe the features of the dataset as a whole. The file has two columns named `Property` and `Value`. Each row in the table indicates the value of one property. Each property has a default value; any property not explicitly defined in `DatasetMetadata.csv` is assumed to have the default value. The following properties are defined.

Version

Description: This property specifies the current version of the ProgSnap 2 standard that these files adhere to. This allows the standard to change over time.

Datatype: `Integer`

Current Value: `6`

IsEventOrderingConsistent

Description: This property specifies whether the events in the main event table are predominantly ordered (within the scope specified by the EventOrderScope property) according to a single, globally-consistent clock, such that the ordering of the events in the same scope can (largely) be assumed to reflect their actual temporal order according to that clock. Datasets originating from distributed systems (including client/server systems) might not have a single clock, in which case the value of this property should be `false`.

Note that data consumers should be prepared to handle anomalies in event ordering, even if this property value is set to `true`.

Values: `true` or `false`

Default value: `false`

EventOrderScope

This property specifies the scope of Order column values within the dataset. The possible values are Global, Restricted, and None. When the value is Global, the Order column values are intended to be meaningful to determine the order of all events (globally) in the dataset. When the value is Restricted, Order column values are only comparable between events with identical values for all of the columns specified by the EventOrderScopeColumns property. When the value is None, the Order column values should never be assumed to determine an ordering for any events; in other words, the events are not ordered.

Values: Global, Restricted, None

Default value: None

EventOrderScopeColumns

This property specifies the main event table columns which define the scope of meaningful comparisons of Order column values. This property must be set to a non-empty value if the EventOrderScope property has the value “Restricted”. (This property has no significance if EventOrderScope is not “Restricted”.) (This property has no significance if EventOrderScope is not “Restricted”.) The value of this column is a semicolon-separated list of main event table column names.

As an example, if a dataset only has meaningful Order column values for events sharing the same TermID, CourseID, AssignmentID, and SubjectID values, then EventOrderScope should be “Restricted”, and the value of this property should be:

```
TermID;CourseID;AssignmentID;SubjectID
```

This would indicate that there is a meaningful order for events from *one* student attempting a given assignment, but that we cannot (or should not) meaningfully order the events of multiple students with respect to each other (e.g., because these represent independent sets of events), and we cannot order a student’s events between different assignments.

There is no significance to the order of the specified columns within this list.

Value: a semicolon-separated list of main event table column names, or empty if the EventOrderScope property is not Restricted

Default value: the default value is the empty string, i.e., no columns are specified

CodeStateRepresentation

Description: This property specifies which CodeState representation is used by the dataset. This property must be specified using one of the legal values listed below.

Values: Table, Directory, Git

Link tables

In some cases, data providers will want to link ID values or combinations of ID values with resources specifying additional information about the entity or entities identified by the ID value(s). Some examples include:

- Linking SubjectID values to documents containing more information about the subject, such as demographic information
- Linking CourseID values to course catalog descriptions
- Linking CourseID/TermID pairs to course webpages with information about specific offerings of a course

Link tables are the mechanism for providing links to resources. Note that all resource files should be stored in the Resources/ directory., while all link tables are stored in the LinkTables/ directory.

The name of a link table is constructed as follows:

1. From each column containing an ID to use as a key, strip “ID” from the end (for example, “CourseID” would become “Course”)
2. Concatenate the transformed column names in lexicographical order to form a single combined name
3. Prepend “LinkTables/”
4. Append “.csv”

So, for example, the link table for CourseID values would be called `LinkTables/Course.csv`. As another example, the link table for CourseID/TermID pairs would be called `LinkTables/CourseTerm.csv`.

Link tables are CSV files. The columns of a link table are, at a minimum

- The columns for the IDs: for example, CourseID and TermID for the `LinkTables/CourseTerm.csv` link table
- A column called “URL” containing a URL linking to the resource specified by the ID or IDs. This column is optional if additional columns are added, as described below.

Link tables may contain other columns in addition to the mandatory columns mentioned above, if needed. These columns should start with the prefix “X-”, to clarify that they are user-defined.

The main event table

The core component of a ProgSnap 2 dataset is a metadata file known as the *main event table*. It is a CSV file named

```
MainTable.csv
```

Each row of the main event table represents an *event*.

Column Headers

This section describes the required, optional, and event-specific columns of the main event table. Note that additional columns may be added by the user as needed; however, users are encouraged to use pre-defined columns where possible.

Columns in the main event table are not required to be in any particular order. Data consumers must use the header of the main table to discover how the columns are ordered for a particular dataset.

Required Columns

- EventType
- EventID
- SubjectID
- ToolInstances
- CodeStateID

Optional Columns

- Order
- ServerTimestamp
- ServerTimezone
- ClientTimestamp
- ClientTimezone
- CourseID
- CourseSectionID
- TermID
- AssignmentID
- AssignmentIsGraded
- ProblemID
- ProblemIsGraded
- Attempt
- ExperimentalCondition
- TeamID
- LoggingErrorID

Event-Specific Columns

- ParentEventID
- SessionID
- ProjectID
- ResourceID
- CodeStateSection
- DestinationCodeStateSection

- EventInitiator
- EditType
- CompileResult
- CompileMessageType
- CompileMessageData
- SourceLocation
- ExecutionID
- TestID
- ExecutionResult
- Score
- ExtraCreditScore
- ProgramInput
- ProgramOutput
- ProgramErrorOutput
- InterventionCategory
- InterventionType
- InterventionMessage

Required Columns

This section documents columns that are required, meaning that they must be present and nonempty for all rows.

EventType

Datatype: Enum

Enum values: Session.Start, Session.End, Project.Open, Project.Close, File.Create, File.Delete, File.Open, File.Close, File.Rename, File.Edit, File.Focus, Compile, Compile.Error, Compile.Warning, Submit, Run.Program, Run.Test, Debug.Program, Debug.Test, Resource.View, Intervention, X-*

Description: Every line logged in a dataset must be associated with a specific event, where events can be categorized as one of several possible types. Users are encouraged to apply the built-in enum values whenever possible, but if a new event type is necessary, the coder may define a new enum type beginning with the string “X-”. The metadata of the associated dataset should define what the new EventTypes mean.

EventType value	**Description**
Session.Start	Marks the start of a work session.
Session.End	Marks the end of a work session.

Project.Open	Indicates that a project was opened.
Project.Close	Indicates that a project was closed due to an explicit user or system action. Data consumers should be prepared to handle cases where Project.Open is not terminated by an explicit Project.Close.
File.Create	Indicates that a file was created.
File.Delete	Indicates that a file was deleted.
File.Open	Indicates that a file was opened.
File.Close	Indicates that a file was closed.
File.Save	Indicates that a file was saved.
File.Rename	Indicates that a file was renamed.
File.Copy	Indicates that a file was copied.
File.Edit	Indicates that the contents of a file were edited.
File.Focus	Indicates that a file was selected by the user within the user interface.
Compile	Indicates an attempt to compile all or part of the code.
Compile.Error	Represents a compilation error and its associated diagnostic.
Compile.Warning	Represents a compilation warning and its associated diagnostic.
Submit	Indicates that code was submitted to the system.
Run.Program	Indicates a program execution and its associated input and/or output.
Run.Test	Indicates execution of a test and its associated input and/or output.
Debug.Program	Indicates a debug execution of the program and its associated input and/or output.
Debug.Test	Indicates a debug execution of a test and its associated input and/or output.
Resource.View	Indicates that a resource (typically a learning resource of some type) was viewed.
Intervention	Indicates that an intervention such as a hint was done.
X-*	Any event type beginning with “X-” is a user-defined event type, for

	events not covered by the categories above.

Note that each event type may have a set of columns that are required for that specific event. These are listed in the Event-Specific Columns section, with the relevant events listed in the *Required for* and *Recommended for* sections in each column description, and summarized in this table. In general, data providers should strive to provide as much information as possible, and avoid leaving data values empty unnecessarily.

EventID

Datatype: ID

Description: Every event must have an ID value that is distinct from (not equal to) all other events in the main event table.

SubjectID

Datatype: ID

Description: An ID representing the subject associated with the event. Whenever possible, the SubjectID should represent a single individual (i.e., a student.) A SubjectID could represent a group of individuals (i.e., a team) if the event truly originates from the group as a whole and is not directly associated with a single individual within the group.

SubjectID values and TeamID values are considered to be in the same namespace. For events where SubjectID and TeamID have the same value, it means that the event is ascribed to a team as a whole rather than any specific member of the team.

When it is not known who the subject associated with an event is, the special ID “UNKNOWN” should be used. This ID should not be used for regular subjects, and should be treated as missing information during analysis.

ToolInstances

Datatype: string

Description: a string detailing the tool(s) associated with the event. This should include any compilers, IDEs, and external tools used during the event. Tools must be separated by semicolons. For example, a submission event using the CloudCoder tool might be represented by the string “Python 3.6.5; CloudCoder 0.1.4”. Versions should be included when known, but can be omitted when less information is available. Examples of tools that should be included:

- Language compiler or interpreter version
- IDE version (this may include both a client and server version)
- Static analyzers
- Feedback/hint tools
- Student Models

In cases where multiple external tools are used, only the external tools which actively contributed to the event should be included in the string, to add clarity.

CodeStateID

Datatype: ID

Description: Each event should contain a pointer to the current state of the student's codebase. If the code has not changed since the previous event, the previous CodeStateID may be reused. More information about how to represent CodeStates can be found further below in the document.

Recommended Optional Columns

This section describes the columns of the main event table that are optional (meaning they may or may not be present for any particular dataset) but which are *recommended* for *all* event types where the value of the column is known. It is possible that only a subset of events (rows) will have a nonempty value for an optional column when the data is not known for all events.

Order

Datatype: Integer

Description: This value indicates a "best guess" chronological event order, as determined by the data provider, within the scope specified by the EventOrderScopeColumns dataset metadata property. Each event within the specified scope must have a distinct Order value. There is no requirement that Order values start with any specific minimum value, nor is there a requirement that Order values always increase by increments of one.

Because Order values are only guaranteed to be unique within the scope defined by EventOrderScopeColumns, the ordering of events in different scopes is unspecified.

In general, there is no single "true" order of events. For example, for systems that collect both server and client timestamps, there is no guarantee that these will be consistent. However, for many types of analysis, especially those that will be implemented using a "streaming" approach (where events are processed in sequence and only minimal context is directly kept in memory at any instant), it is useful to have a default ordering of events that can be expected to represent the “true” chronology with some reasonable degree of accuracy. The Order column is intended to provide that default ordering.

ServerTimestamp

Datatype: Timestamp

Description: A ServerTimestamp value indicates the time when an event was logged on a server system. In general, it is expected that servers will have clocks that are (to a reasonable degree) accurately synchronized with global time standards (e.g., using NTP), although this cannot be guaranteed. Also, in cases where there are multiple servers, their clocks may not be completely synchronized with each other.

ServerTimezone

Datatype: Timezone

Description: A ServerTimezone value indicates the timezone (offset from UTC) to which the ServerTimestamp value is relative. Combined with the ServerTimestamp, it indicates the specific instant in time when an event was recorded on a server.

ClientTimestamp

Datatype: Timestamp

Description: A ClientTimestamp value indicates the time when an event was registered on a client system (generally, the system being used directly by the student), as reported by the client system. In general, ClientTimestamp values can be assumed to provide an accurate chronology of events within a single session (as indicated by the SessionID value), and usually can be meaningfully compared between sessions for the same student (SubjectID), but they might not be accurately synchronized with global time standards.

ClientTimezone

Datatype: Timezone

Description: A ClientTimezone value indicates the timezone to which a ClientTimestamp value is relative. Combined with the ClientTimestamp value, it identifies the precise instant in time when an event was recorded on a client system, with the caveat that clocks on client systems might not be accurately synchronized with global time standards.

CourseID

Datatype: ID

Description: Students are usually associated with a specific course that they are learning in. This course must be given an ID that is shared across all students enrolled in the course, but distinct from different courses in the same dataset. We define courses to be different when they teach different content (e.g., CS1 vs CS2). Note that a course which takes place over several terms with different students should be given the same ID across all terms; the datasets will be distinguished by their TermIDs.

We recommend that CourseIDs be at least somewhat anonymized, to avoid making student data identifiable.

CourseSectionID

Datatype: ID

Description: Courses are often split up into smaller sections of students who primarily interact with each other and a specific TA. If applicable, each section should be given a distinct ID (unique from other sections in the given course and other courses). CourseSections should *not* share IDs across terms.

We recommend that CourseSectionIDs be at least somewhat anonymized, to avoid making student data identifiable.

TermID

Datatype: string

Description: The term in which the course took place. Can be written as needed, but we recommend the format ‘<Semester> <Year>’; for example, ‘Spring 2018’.

AssignmentID

Datatype: ID

Description: CodeStates are often associated with a specific assignment that is composed of one or more programming problems. Each unique assignment must be given a distinct ID from other assignments in the associated course and other courses. If an assignment is identical to an assignment in a previous term of the course or another course, they should be given the same ID, but any changes in the assignment should result in a changed ID.

If the CodeState represents free-form student work not associated with a specific assignment or problem, this value should be empty. Stand-alone problems also do not need to be associated with assignments.

AssignmentIsGraded

Datatype: Boolean

Description: This value indicates whether or not the assignment specified by the AssignmentID was graded (`true`) or ungraded (`false`).

ProblemID

Datatype: ID

Description: The identifier for the programming problem associated with the event. Each unique problem must have its own identifier that is distinct from other identifiers in the same column that correspond to different problems. If a record specifying a ProblemID also specifies an AssignmentID, it means that the problem is part of the specified assignment. There is no requirement that problems are associated with an assignment: for example, a standalone practice problem might not be considered to be part of an assignment.

ProblemIsGraded

Datatype: Boolean

Description: This value indicates whether or not the problem specified by the ProblemID was graded (`true`) or ungraded (`false`).

Attempt

Datatype: Integer

Description: If a student attempts a problem more than once, this value is used to identify which attempt they’re on. It should start at 1, then increase by 1 on each following attempt.

ExperimentalCondition

Datatype: String

Description: If this data was logged as part of an experiment, this column can be used to specify the experimental condition that the event took place in. Condition names must be consistent for events in the same condition, and (if possible) distinct between different experiments. This can be accomplished by assigning each experiment in the dataset a distinct name. An example condition string is “02/18 Parsons Problem Study: Control”; this establishes the condition (control case), the study content (parsons problems), and when the study took place (February 2018).

TeamID

Datatype: ID

Description: This value indicates the identity of a team. There are two possible meanings of TeamID:

- If the TeamID value is different than the SubjectID value, it means that the SubjectID designates a single individual, and the TeamID value identifies the team the individual belongs to.

If the TeamID value is the same as the SubjectID value, it means that the SubjectID designates a team, and that the event is ascribed to the team as a whole rather than any individual member of the team. When this value is used, a Link Table should be created to map the TeamID to a list of SubjectIDs, when known. This should be done with two columns: the first column the TeamID, the second a single SubjectID, where the number of rows the TeamID appears in maps to the number of SubjectIDs.

LoggingErrorID

Datatype: ID

Description: Logging errors are an inevitable part of the data collection process. If a data collector finds that an error occurred during the logging process, they should leave the data in its original state, but annotate all erroneous data with IDs, where each ID corresponds to a specific logging error event. Further information about the error can then be provided in a link table (which should include the ID, error type, and an explanation).

Note that logging errors can come in many forms, including corrupted/lost data, server downtime, and tool errors that result in incorrect feedback. We define a logging error to be anything that results in the log not accurately representing the true state of the world.

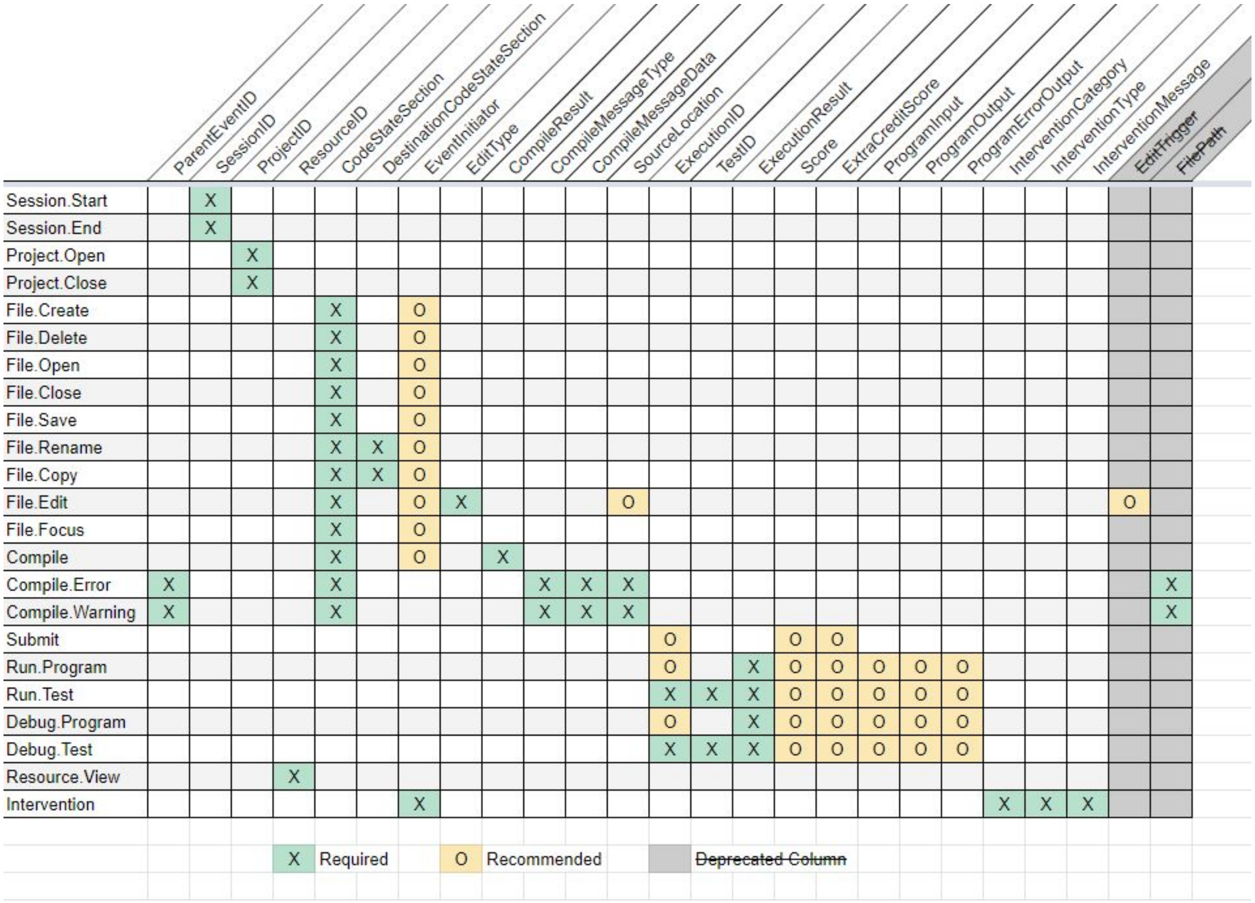
Event-Specific Columns

This section describes the columns of the main event table that are associated with *specific* event types and therefore may not be used in every row of the dataset. An event-specific column only needs to be included in a dataset if an event which requires it appears in some row in the dataset as well. Each event-specific column description includes Required-for and Recommended-for sections.

The Required-for section lists event types for which the column must have a meaningful value. In other words, a data consumer can safely assume that the value of the column is meaningful for all of the Required-for event types that appear in the dataset. Note that “meaningful” does not necessarily imply nonempty: it is possible, for example, that an empty string could be a meaningful value depending on the purpose of the column.

The Recommended-for section lists event types for which data providers should provide a meaningful value if possible, but where a meaningful value is not absolutely required. If a Recommended-for section is not included, the column may still be used for any event types where the column data is relevant.

A table matching EventTypes to suggested and required columns can be found here.



	ParentEventID	SessionID	ProjectID	ResourceID	CodeStateSection	DestinationCodeStateSection	EventInitiator	EditType	CompileResult	CompileMessageType	CompileMessageData	SourceLocation	ExecutionID	TestID	ExecutionResult	Score	ExtraCreditScore	ProgramInput	ProgramOutput	ProgramErrorOutput	InterventionCategory	InterventionType	InterventionMessage	~~EditTrigger~~	~~FilePath~~
Session.Start		X																							
Session.End		X																							
Project.Open			X																						
Project.Close			X																						
File.Create					X		O																		
File.Delete					X		O																		
File.Open					X		O																		
File.Close					X		O																		
File.Save					X		O																		
File.Rename					X	X	O																		
File.Copy					X	X	O																		
File.Edit					X		O	X				O												O	
File.Focus					X		O																		
Compile					X		O		X																
Compile.Error	X				X					X	X	X													X
Compile.Warning	X				X					X	X	X													X
Submit													O			O	O								
Run.Program													O		X	O	O	O	O	O					
Run.Test													X	X	X	O	O	O	O	O					
Debug.Program													O		X	O	O	O	O	O					
Debug.Test													X	X	X	O	O	O	O	O					
Resource.View				X																					
Intervention							X														X	X	X		

X Required O Recommended ▨ ~~Deprecated Column~~

ParentEventID

Datatype: ID

Description: Certain events are *hierarchical*, where multiple child events might be associated with a single parent event. In these cases, the parent event should be referenced in this column by its EventID value.

Note that this column is still under design; more ‘required for’ event types will be added with time, after the format has been tested.

Required for: Compile.* events (must reference parent Compile event).

Recommended for: Any X-* events where it is desirable to indicate a relationship to a parent event.

SessionID

Datatype: ID

Description: A session is generally defined as a distinct period of time during which a student is interacting with a tool/program. Sessions are somewhat ill-defined and may vary across datasets. Session IDs must be unique across subjects and across distinct sessions. This ID may be the EventID of the SessionStart event that initiated the session, or it may be derived independently.

Required for: Session.* events

ProjectID

Datatype: ID

Description: A project is a collection of source files that can be opened and closed (in Project.* events). Note that a project may be distinct from an assignment or problem. For example, one assignment might extend another, in which case the student will load the same project and continue working on it.

Data producers should only generate Project.* events and ProjectID values if the underlying data source has an explicit concept of “project”.

Required for: Project.* events

ResourceID

Datatype: ID

Description: Often students access resources while working on problems. Example resources include API documentation, online textbooks, and demo videos. In a dataset which logs student access to resources, each resource must be assigned a distinct ID. If resources are not changed across terms, their IDs should be reused.

Required for: Resource.View events

CodeStateSection

Datatype: RelativePath

Description: A CodeStateSection value names a single file or resource within a CodeState which is specifically associated with the event. Examples:

- In a File.Create event, the CodeStateSection identifies the file created
- In a Compile.Error event, the CodeStateSection identifies the source file in which the compilation error occurs

Note that for events where there is both a "source" file/resource and a "destination" file/resource, the CodeStateSection value indicates the "source". For example, for File.Copy and File.Rename events, the CodeStateSection names the "original" file. (Note that in the case of File.Rename events, the CodeStateSection value identifies a file or resource in the *previous* CodeState.)

Note that a CodeStateSection may only refer to a single file. Cases where multiple resources are accessed or modified at the same time (such as using "Save All" to save all files) should be represented as multiple events, each with its own distinct CodeStateSection.

Also note that CodeStateSections should not be used for CodeStates in the Table format, as all table data is contained in the same file.

Required for: File.*, Compile, and Compile.* events where the CodeState is not in the Table format

DestinationCodeStateSection

Datatype: RelativePath

Description: For events associated with two files or resources — a "source" and a "destination" — the DestinationCodeStateSection value specifies the destination resource. For example, for File.Copy and File.Rename events, the DestinationCodeStateSection value specifies the "new" file or resource.

Note that this column should only contain a nonempty value if the CodeStateSection column contains a nonempty value.

Required for: File.Copy and File.Rename events

EventInitiator

Datatype: Enum

Enum values: UserDirectAction, UserIndirectAction, ToolReaction, ToolTimedEvent, InstructorDirectAction, InstructorIndirectAction, TeamMemberDirectAction, TeamMemberIndirectAction, X-*

Description: Events are typically performed by either the user, the tool, or the instructor. When known, this column should specify which one instigated the event.

Note that user, instructor, and team members can initiate actions either directly or indirectly. A direct action is one the person purposefully makes (like typing or editing a program with mouse clicks); an indirect action is one that is caused by a user action, but not done directly by the user (like when a user accepts an autocomplete recommendation and the text is filled in).

Further information is provided in the following table:

EventInitiator value	**Description**
UserDirectAction	Indicates that the user directly instigated the action.
UserIndirectAction	Indicates that the user indirectly instigated an action.
ToolReaction	Indicates that a tool caused the edit as a reaction to something the user did.
ToolTimedEvent	Indicates that a tool caused the edit as part of a time-based event (such as automatically saving every five minutes).
InstructorDirectAction	Indicates that the instructor directly caused the action, potentially remotely.
InstructorIndirectAction	Indicates that the instructor indirectly caused the action, potentially remotely.
TeamMemberDirectAction	Indicates that a team member directly caused the event; for example, if two students are pair-programming on a shared screen.
TeamMemberIndirectAction	Indicates that a team member indirectly caused the event; for example, if two students are pair-programming on a shared screen.
X-*	Any initiator beginning with “X-” is a user-defined type, for special initiators not covered by the categories above.

When a tool initiated the event, the column ToolInstances should include that tool for clarity.

Required for: Intervention events

Recommended for: File.* events, Compile events

EditType

Datatype: Enum

Enum values: GenericEdit, Insert, Delete, Replace, Move, Paste, Undo, Redo, Refactor, Reset, X-*

Description: This value indicates the type of edit which caused the file to change. Specific values are described in the table below.

EditType value	**Description**
Insert	Indicates that one or more characters or values have been added.
Delete	Indicates that one or more characters of values have been deleted.
Replace	Indicates that one or more characters or values have been replaced by new characters/values.
Move	Indicates that one or more characters or values have been moved to a new location.
Paste	Indicates that one or more characters or values have been pasted into the program.
Undo	Indicates that the most recent edit not of an undo/redo type was undone.
Redo	Indicates that the most recent edit that had been undone was re-done.
Refactor	Indicates that the program has been refactored in some way.
Reset	Indicates that the program has been reset to its start state.
GenericEdit	Any generic edit that can not be described by the edits listed above. We recommend that this is not used for special edits; those should be classified as X-* edits.
X-*	Any event type beginning with “X-” is a user-defined event type, for special edit events not covered by the categories above.

Required for: File.Edit events

CompileResult

Datatype: Enum

Enum values: Success, Warning, Error

Description: Compile events can either result in an error, a warning, or a general success.

Required for: Compile events

CompileMessageType

Datatype: String

Description: The type/ID of compile message provided. If no error or warning was given, the string “Success” should be used. The types of errors and warnings used will otherwise vary by language; for example, a Python compile message type might be a ‘SyntaxError’ or an ‘IndentationError’.

Required for: Compile.* events

CompileMessageData

Datatype: String

Description: The specific compiler message shown to the student.

Required for: none

Recommended for: Compile.* events

SourceLocation

Datatype: SourceLocation

Description: A SourceLocation value represents a location or region within a source file, associated with a compiler diagnostic, static analysis warning, or other message about program source. It can also describe the location of an edit in source code during File.Edit events. Note that due to the large number of ways file contents could change as a result of a File.Edit event, the SourceLocation value associated with a File.Edit event (if any) should be considered to be a “hint” regarding the location of the change(s) represented by the event. The true change corresponding to a File.Edit event is indicated by the changes to the event's CodeState relative to the previous CodeState.

Required for: Compile.* events

Recommended for: File.Edit events

ExecutionID

Datatype: ID

Description: This ID value is used to group Run.Test events that were part of the same overall test execution. For example, if multiple unit tests were executed, resulting in one Run.Test event for each unit test, all of the Run.Test events in the group should share a common ExecutionID.

If the code execution is associated with a submission, then the Submit event should have an ExecutionID value, and the associated Run.Test, Debug.Test, and/or Run.Program events should share the same ExecutionID value.

For consistency, this ID value may also be specified for Run.Program events.

Required for: Run.Test and Debug.Test events

Recommended for: Submit events (if applicable), Run.Program and Debug.Program events

TestID

Datatype: ID

Description: An ID indicating which test case is associated with the event. If desired, a link table may map IDs to further information about the individual test cases. Note that TestID values may be human-readable: for example, the names of JUnit tests could be used as TestID values.

Required for: Run.Test and Debug.Test events

ExecutionResult

Datatype: Enum

Enum values: Success, Timeout, Error, TestFailed

Description: Run.Program events can result in Success (the program runs fully to completion), Timeout (the program's execution is interrupted by the user or the system), or Error (the program execution is terminated by a compiler or runtime error).

Run.Test events can result in Success (the test passes), Timeout (the test failed to complete in the allotted time), Error (the test failed due to a fatal runtime exception), or TestFailed (the test produces the incorrect output). Note that assertion errors should be classified as TestFailed, not Error.

Required for: Run.* and Debug.* events

Score

Datatype: Real

Description: A Score value ranges between 0.0 and 1.0, and indicates the normalized degree of correctness of the submitted code with respect to a specific test (in the case of a Run.Test event) or with respect to all tests and correctness criteria (in the case of a Submit event), excluding extra credit criteria.

A completely incorrect test result or submission should be assigned a score of 0.0, and a completely correct test result or submission should be assigned a score of 1.0. If a test result/submission is partially incorrect, it may either have a number in the range [0.0, 1.0) or may be set to 0.0 automatically; this should be specified in the README. In general, it is expected that

1. A Run.Test event will have a Score of 1.0 if the ExecutionResult is Success, and 0.0 otherwise
2. A Submit event will have a Score that is the average (possibly weighted) of the Score values of the Run.Test events associated with the Submit event (i.e., those having the same ExecutionID value)

In some sense Score values are redundant, because they could be inferred from analyzing Run.Test events. However, for many types of analysis, having a single Score value directly associated with a Submit event is highly valuable, and data providers are strongly encouraged to include Score values.

Note that while a Score could be the basis of an assigned grade, there is no implication that a Score is *necessarily* a grade. It is simply intended to capture the normalized degree of correctness of submitted code.

Note also that Run.Test events and potentially even Submit events could omit the Score value if they are intended exclusively as extra credit. Also, events can omit the Score value if it is not possible for a score to be calculated immediately (as is the case for creative or manually graded problems). When a manual grade is provided, an EarnedGrade Intervention should be used to log the grade.

Required for: none

Recommended for: Submit, Run.* and Debug.* events

ExtraCreditScore

Datatype: Real

Description: An ExtraCreditScore value ranges between 0.0 and 1.0, and indicates the degree to which a single test (in the case of Run.Test events) or submission (in the case of Submit events) satisfies extra credit criteria. This column should not contain any value for Run.Test and Submit events that have no extra credit criteria.

Required for: none

Recommend for: Submit, Run.*, and Debug.* events that are intended at least partially as extra credit

ProgramInput

Datatype: URL

Description: Programs are often provided with input at the beginning of a run or test. The ProgramInput value specifies the URL which records the program input. There are two possibilities for the resource identified by the URL:

1. If the URL refers to a file, the file's contents are the program input. This possibility is intended to handle the case where the program is receiving input via its standard input channel (`stdin` in C, `System.in` in Java, etc.)
2. If the URL refers to a directory, the directory contains one or more files that constitute the program's input. This possibility is intended to handle the case where the program is receiving input from some combination of files and standard input. The naming and meaning of these files is unspecified; data producers are encouraged to use descriptive names.

ProgramInput and ProgramOutput are all listed as recommended, not required, for Run.* events. However, data collectors are strongly encouraged to provide information on the input/output whenever possible. These values should only be left blank when it is impossible to present the data directly (for example, if the output is an interactive animation that cannot be stored statically).

Required for: none

Recommended for: Run.* and Debug.* events

ProgramOutput

Datatype: URL

Description: Programs often produce output at the end of a run or test. The ProgamOutput value specifies the URL which records the program output. The URL will typically refer to an "internal" file within the dataset's Resources directory. Note that ProgramOutput is intended to capture the "standard" output channel of the program, i.e., `stdout` in C, `cout` in C++, `System.out` in Java, etc.

ProgramInput and ProgramOutput are all listed as recommended, not required, for Run.* events. However, data collectors are strongly encouraged to provide information on the input/output whenever possible. These values should only be left blank when it is impossible to present the data directly (for example, if the output is an interactive animation that cannot be stored statically).

Required for: none

Recommended for: Run.* and Debug.* events

ProgramErrorOutput

Datatype: URL

Description: Programs often produce error output at the end of a run or test. The ProgamErrorOutput value specifies the URL which records the program's error channel output. The URL will typically refer to an "internal" file within the dataset's Resources directory. Note that ProgramErrorOutput is intended to capture the "error" output channel of the program, i.e., `stderr` in C, `cerr` in C++, `System.err` in Java, etc.

Required for: none

Recommended for: Run.* and Debug* events which produced non-empty error output

InterventionCategory

Datatype: Enum

Enum values: Feedback, Hint, CodeHighlight, CodeChange, EarnedGrade, X-*

Description: An Intervention event is an interaction with the subject initiated during the programming process; for example, showing the students a targeted feedback message when they fail a specific test case. We include common intervention categories here, but new ones with names starting with the prefix “X-” may be used. Common interventions should be recommended for inclusion in future versions of ProgSnap 2.

Note that Compile and Run events are *not* interventions; these events are ubiquitous enough that they have been given their own event types.

Required for: Intervention events

InterventionType

Datatype: String

Description: System-level information about the type of intervention being performed. For feedback, this might be the type of error or code state that was detected; for CodeHighlight, this might be the starting and ending coordinates of the highlighted code. This can be organized freely by the logger, but the format should be consistent within datasets, and should state the information as succinctly as possible.

Required for: Intervention events

InterventionMessage

Datatype: String

Description: The actual intervention message shown to the student, when applicable. If no message is shown but a visual effect occurs, the effect should be described (possibly using a dataset-specific coding scheme).

Required for: Intervention events

CodeStates

CodeStates are used to represent student code in the ProgSnap 2 format. Unlike the main table, CodeState representation can and should vary across different datasets, based on the type of work being stored. However, we recommend that developers represent their code using the following guidelines, to increase shareability.

CodeState Directory

The CodeState directory should hold the files containing the code data, where files can be referenced by ID in the main table. How that data is stored varies based on the type of data that has been logged. We consider varying types of data along the following dimensions:

- Program size (small or large)
- Number of files per CodeState (one or multiple)
- Code format (text or other)
- Size of dataset

The recommended representation format can then be decided based on the factors mentioned above:

- The **Git Format** might be most appropriate for large datasets and datasets with snapshots consisting of large numbers of files
- The **Directory Format** might be most appropriate for medium size datasets and datasets where the snapshots contain a small number of files (including single files)
- The **Table Format** might be most appropriate for datasets where each snapshot is a single file of a relatively small size and the source code representation is text-based

Producers of ProgSnap 2 datasets must use one of the following CodeState representations. In general, it is left up to producers to decide which representation is most appropriate (considering the factors described above.) Producers should specify the CodeStateRepresentation property of the dataset's `DatasetMetadata.csv` to Git, Directory, or Table as appropriate.

Consumers of ProgSnap 2 datasets should be prepared to work with any of the following CodeState representations.

All code data should be stored in the dataset's `CodeStates` directory.

Table Format

In the Table Format, all CodeStates are stored in a single CSV file for ease of access. This CSV should be named `CodeStates.csv` (in the `CodeStates` directory), and should contain at least two columns: CodeStateID and Code. We strongly recommend that developers generate the CSV in RFC 4180 format using a standard library, to avoid parsing problems.

The CodeStateID column should hold each code state's ID, which should follow the ID datatype described by the main table. The Code column should hold the complete program text referenced by the CodeStateID. The whole code text should be included, not just a diff.

If there are multiple useful representations of a code state, additional columns can be used to store these as well. For example, for a block-based language, the Code column might store the original XML project file that the programming environment can load, while a second JSON column contains a JSON representation of the AST, allowing the data consumer to skip the step of parsing the XML. A further Pseudocode column could contain a human-readable version of the code.

Directory Format

In the Directory Format, each CodeState is stored in a different subdirectory within the dataset's `CodeStates` directory, where the name of the directory is the CodeState ID. The directory then contains all files associated with the CodeState in their full (non-diff) form. Each CodeState directory may have an internal directory structure, and files may be located anywhere within the internal directory structure.

Care must be taken to ensure that CodeStateID values are legal directory names for common operating systems. For this reason, it is recommended that Directory Format CodeStateID values consist of the characters a-z, A-Z, 0-9, underscore ("_"), hyphen ("-"), and forward slash ("/"). When a slash ("/") character appears in a CodeStateID, it is considered to be a path separator, such that the CodeStateID describes a path to the specific CodeState directory within the dataset's overall `CodeStates` directory. Note that a CodeStateID value should not begin with a slash ("/"), since that would indicate an absolute path. Because a ProgSnap2 dataset could have a large number of code states, it is recommended that data producers use a hierarchical organization to avoid the `CodeStates` directory having an unreasonably large number of immediate subdirectories.

It is recommended (but not required) that directories representing code states should be **deduplicated**: that is, it should not be the case that two directories representing code states have identical contents. If a unique code state occurs across multiple events or students (for example, the starter code for an assignment), it should be referred to by the same CodeStateID in each event where it occurs.

Git Format

In the Git Format, a single Git repository contains all of the dataset's CodeStates, with each CodeState being represented as a commit.

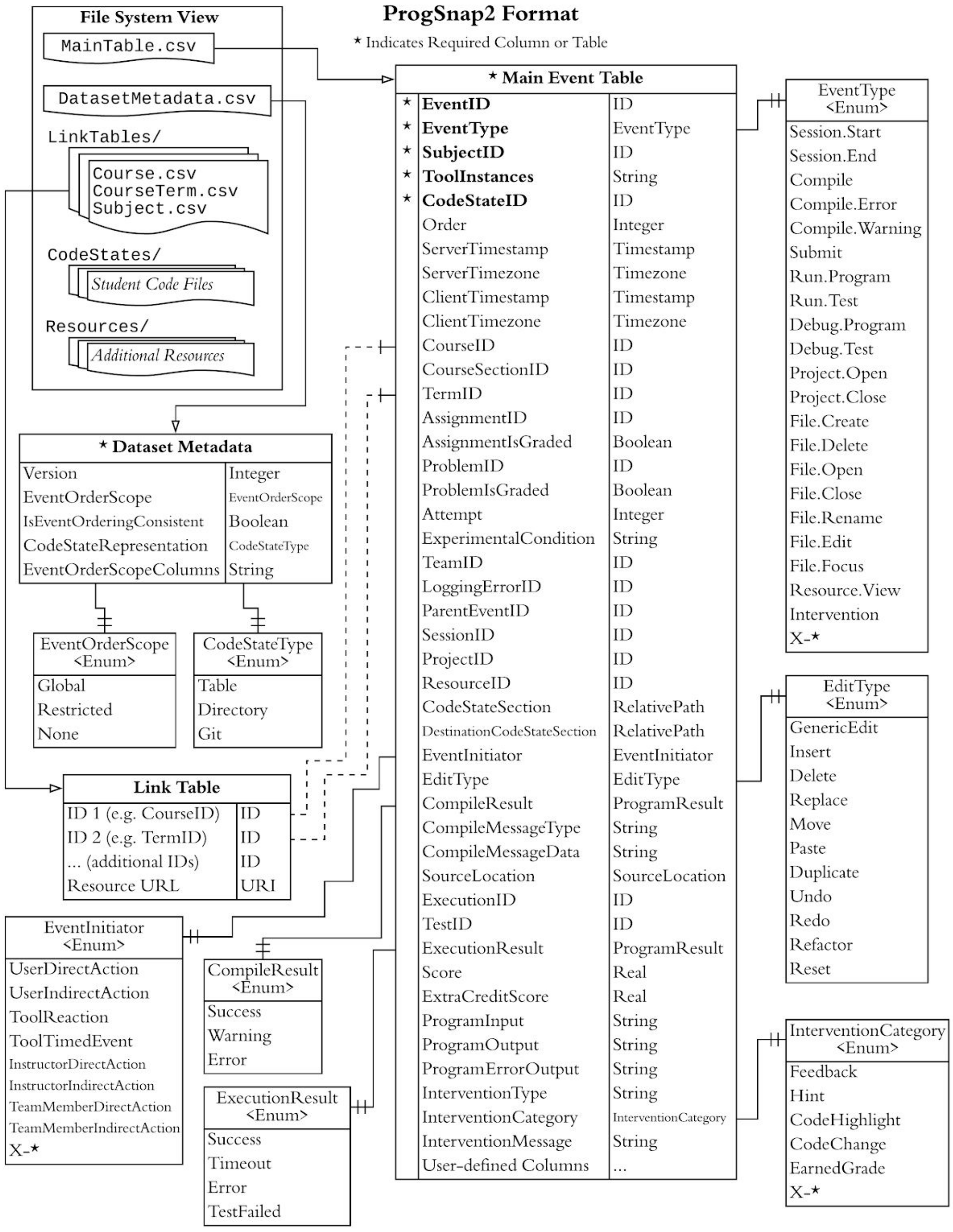
In this representation, the CodeStates directory should be a bare Git repository (i.e., one created with the `git init --bare` command.) Each CodeState ID should name a commit in the Git repository. It is strongly recommended for dataset producers to use parent/child relationships between commits to represent code histories. For example, consider a sequence of edits made by one student working on one project. Assume there are CodeStates C1, C2, C3, etc., representing the edit sequence. Each CodeState is a commit in the Git repository. In order to model the connections between the CodeStates in the sequence, C1 should be C2's parent, C2 should be C3's parent, etc.

Open Questions

- Should information about a CodeState's language be included in the CodeState, or in the main table?
- Should more structured formats (like an AST) be stored in the CodeState, or generated by a script later on?

Appendix A: Entity-Relationship Diagram

This image provides a structural overview of the ProgSnap2 format.



⋆ Main Event Table		
⋆	**EventID**	ID
⋆	**EventType**	EventType
⋆	**SubjectID**	ID
⋆	**ToolInstances**	String
⋆	**CodeStateID**	ID
	Order	Integer
	ServerTimestamp	Timestamp
	ServerTimezone	Timezone
	ClientTimestamp	Timestamp
	ClientTimezone	Timezone
	CourseID	ID
	CourseSectionID	ID
	TermID	ID
	AssignmentID	ID
	AssignmentIsGraded	Boolean
	ProblemID	ID
	ProblemIsGraded	Boolean
	Attempt	Integer
	ExperimentalCondition	String
	TeamID	ID
	LoggingErrorID	ID
	ParentEventID	ID
	SessionID	ID
	ProjectID	ID
	ResourceID	ID
	CodeStateSection	RelativePath
	DestinationCodeStateSection	RelativePath
	EventInitiator	EventInitiator
	EditType	EditType
	CompileResult	ProgramResult
	CompileMessageType	String
	CompileMessageData	String
	SourceLocation	SourceLocation
	ExecutionID	ID
	TestID	ID
	ExecutionResult	ProgramResult
	Score	Real
	ExtraCreditScore	Real
	ProgramInput	String
	ProgramOutput	String
	ProgramErrorOutput	String
	InterventionType	String
	InterventionCategory	InterventionCategory
	InterventionMessage	String
	User-defined Columns	...

⋆ Dataset Metadata	
Version	Integer
EventOrderScope	EventOrderScope
IsEventOrderingConsistent	Boolean
CodeStateRepresentation	CodeStateType
EventOrderScopeColumns	String

Link Table	
ID 1 (e.g. CourseID)	ID
ID 2 (e.g. TermID)	ID
... (additional IDs)	ID
Resource URL	URI

EventOrderScope <Enum>
Global
Restricted
None

CodeStateType <Enum>
Table
Directory
Git

EventInitiator <Enum>
UserDirectAction
UserIndirectAction
ToolReaction
ToolTimedEvent
InstructorDirectAction
InstructorIndirectAction
TeamMemberDirectAction
TeamMemberIndirectAction
X-⋆

CompileResult <Enum>
Success
Warning
Error

ExecutionResult <Enum>
Success
Timeout
Error
TestFailed

EventType <Enum>
Session.Start
Session.End
Compile
Compile.Error
Compile.Warning
Submit
Run.Program
Run.Test
Debug.Program
Debug.Test
Project.Open
Project.Close
File.Create
File.Delete
File.Open
File.Close
File.Rename
File.Edit
File.Focus
Resource.View
Intervention
X-⋆

EditType <Enum>
GenericEdit
Insert
Delete
Replace
Move
Paste
Duplicate
Undo
Redo
Refactor
Reset

InterventionCategory <Enum>
Feedback
Hint
CodeHighlight
CodeChange
EarnedGrade
X-⋆