Progsnap 2 — A standardized representation for programming snapshot data

TODO: authors, contact info

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, are located in the primary directory, while additional files are located in their respective subdirectories.

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.

Timestamp

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

```
2018-09-07T08:41
```

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 local to the dataset (TODO: specify what this looks like more completely)

SourceLocation

A SourceLocation value indicates a location in a source code artifact. A SourceLocation is a sequence of one or more integer values separated by colon (‘:’) characters.

For text-based source languages, a SourceLocation will have 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

    13

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

    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, such as block languages, the mapping of SourceLocation values to code artifact features will depend on the specific code representation used in the dataset. In general, we recommend that SourceLocation values represent a path, with the path elements being ordered from least specific to most specific. If a tree-structured code representation is used, the SourceLocation values could describe a path from the root of the tree to a node. For example, the SourceLocation

```
0:1:3
```

would indicate the 3rd child of the 1st child of the root node.

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`, in that order. 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: `3`

AreEventsOrdered

Description: This property specifies whether or not the events in the main event table are ordered. Most datasets will have ordered events. However, some datasets could consist of isolated examples of student work without any ordering information.

Values: `true` or `false`

Default value: `false` ~~`true`~~

IsEventOrderingConsistent

Description: This property specifies whether the events in the main event table are predominantly ordered according to a single, globally-consistent clock, such that the ordering of the events in the table 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`

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

Link tables may contain other columns in addition to the mandatory columns mentioned above.

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 and optional 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 (required and optional) 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
- Order
- SubjectID
- ToolInstances
- CodeStateID

Optional Columns

- ParentEventID
- ServerTimestamp
- ServerTimezone
- ClientTimestamp
- ClientTimezone
- SessionID
- CourseID
- CourseSectionID
- TermID
- AssignmentID
- ResourceID
- ProblemID
- ExperimentalCondition
- TeamID
- CodeStateSection
- EventInitiator
- EditTrigger
- EditType
- ProgramResult
- CompileMessageType
- CompileMessageData
- FilePath
- Location
- ProgramInput
- ProgramOutput
- 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.
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.Rename	Indicates that a file was renamed.
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 for the optional columns, there may be explicit requirements or recommendations for how particular event types are handled. These are described by the *Required for* and *Recommended for* sections in each optional column description. 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.

Order

Datatype: Integer

Description: This value indicates a “best guess” chronological event order, as determined by the data provider. Each event must have a distinct Order value. The events (rows) must be physically ordered to match the order value (such that an event with an earlier Order value

always is physically positioned before an event with a later 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.

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.

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:

- Compiler or interpreter version
- IDE version (this may include both a client and server version)
- Version of any additional tools (e.g. static analysis, hints, student model)

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. TODO: how are we actually storing CodeStates?

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.) In addition, it is possible that only a subset of events (rows) will have a nonempty value for an optional column: for example, a nonempty value might only occur for a subset of event types.

Each optional 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, if the column is present in a dataset, a data consumer can safely assume that the value of the column is meaningful for all of the *Required for* event types. 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.

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.

Required for: Compile.Error and Compile.Warning (must reference parent Compile event) TODO: what other kinds parent events are there? Are Debug.* or Run.* events the children of something?

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

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.

Required for: none

Recommended for: all events where a server timestamp was recorded

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.

Required for: none

Recommended for: all events where a server timestamp was recorded

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.

Required for: none

Recommended for: all events where a client timestamp was recorded

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.

Required for: none

Recommended for: all events where a client timestamp was recorded

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.Start and Session.End events

Recommended for: all events known to have occurred in a particular session

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.

Required for: none

Recommended for: all events

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.

Required for: none

Recommended for: all events

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’.

Required for: none

Recommended for: all events

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.

Required for: none

Recommended for: all 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

Recommended for: any event where a resource is accessed

ProblemID

Datatype: ID

Description: The identifier for the programming problem associated with the event, if there are multiple problems in the current assignment. Each unique problem must have its own identifier that is distinct from other identifiers in the same column that correspond to different problems.

Required for: none

Recommended for: all events

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).

Required for: none

Recommended for: all events

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.

Required for: none

Recommended for: all events that were recorded in a team context

TODO: need a way to represent team membership.

CodeStateSection

Datatype: String

Description: CodeStates are generally composed of multiple parts, which may be files, scripts, or problems. A CodeState and CodeStateSection, when combined, must allow the user to retrieve a specific subset of the CodeState from an external source. CodeStateSection strings should be constant within a single subject; whether they are kept distinct or shared across subjects/terms may be decided based on the project.

Required for: none

Recommended for: File.*, Compile, Compile.Error, and Compile.Warning events

EventInitiator

Datatype: Enum

Enum values: User, Tool, Instructor, TeamMember

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.

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

Required for: none

Recommended for: File.* events, Compile events

EditType

Datatype: Enum

Enum values: GenericEdit, Insert, Delete, Replace, Move, Paste, Duplicate, Undo, Redo, Refactor, Reset

Description: This value indicates the type of edit which caused the file to change. TODO: describe the meaning of the Enum values.

Required for: File.Edit events

Recommended for: none (except for File.Edit events, for which a value is required)

EditTrigger

Datatype: Enum

Enum values: Keystroke, LineChange, Timed, FocusChange (TODO: do these values make sense? Should there be others?)

Description: What triggered this edit to be recorded. This gives an idea of the granularity of the edit. TODO: document what the values mean.

Required for: File.Edit events

Recommended for: none (except for File.Edit events, for which a value is required)

ProgramResult

Datatype: Enum

Enum values: Success, Warning, Error

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

Required for: Compile events, Run.* 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.Error and Compile.Warning events

Recommended for: none (other than Compile.Error and Compile.Warning, in which case a value is required)

CompileMessageData

Datatype: String

Description: The specific compiler message shown to the student.

Required for: none

Recommended for: Compile.Error and Compile.Warning events

FilePath

Datatype: NonemptyString

Description: A FilePath value names a specific source file associated with a compiler diagnostic, static analysis warning, or other message about program source.

Required for: Compile.Error and Compile.Warning events

Recommended for: none (other than Compile.Error and Compile.Warning, in which case a value is required)

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.

Required for: Compile.Error and Compile.Warning events

Recommended for: none (other than Compile.Error and Compile.Warning, in which case a value is required)

ProgramInput

Datatype: String

Description: Programs are often provided with input at the beginning of a run or test. This input may either be represented directly as a string, or the string may be a reference that can be combined with an external file to fetch the actual input. We recommend that the latter method be used for large inputs and inputs that cannot easily be represented as strings. TODO: need a way to distinguish when the data is linked rather than stored directly.

Required for: Run.* events

Recommended for: none (other than Run.* events, in which case a value is required)

ProgramOutput

Datatype: String

Description: Programs often produce output at the end of a run or test. This output may either be represented directly as a string, or the string may be a reference that can be combined with an external file to fetch the actual output. We recommend that the latter method be used for large outputs and outputs that cannot easily be represented as strings. TODO: need a way to distinguish when the data is linked rather than stored directly.

Required for: Run.* events

Recommended for: none (other than Run.* events, in which case a value is required)

InterventionType

Datatype: Enum

Enum values: Feedback, Hint, CodeHighlight, CodeChange

Description: An Intervention 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 types here, but new ones may be added when new interventions are performed.

Required for: Intervention events

Recommended for: none (other than Intervention events, in which case a value is required)

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

Recommended for: none (other than Intervention events, in which case a value is required)