ProgSnap2: A Flexible Format for Programming Process Data

Thomas W. Price[1], David Hovemeyer[2], Kelly Rivers[3], Austin Cory Bart[4], Andrew Petersen[5], Brett A. Becker[6], Jason Lefever[2]
[1]North Carolina State University, [2]York College of Pennsylvania, [3]Carnegie Mellon University, [4]University of Delaware, [5]University of Toronto, [6]University College Dublin
twprice@ncsu.edu, dhovemey@ycp.edu, krivers@andrew.cmu.edu, acbart@udel.edu, petersen@cs.toronto.edu, brett.becker@ucd.ie, jlefever@ycp.edu

ABSTRACT: In this paper, we introduce ProgSnap2, a standardized format for logging programming process data. The goal of this common format is to encourage collaboration among researchers by helping them to share data, analysis code, and data-driven tools to support students. We first highlight possible use cases for ProgSnap2 and give a high-level overview of the format. We then share two case studies of our experience using the format and outline goals for the future of ProgSnap2, including a call for collaboration with interested researchers.

Keywords: programming process data, data standards, data sharing, learning analytics

1 INTRODUCTION

Analysis of programming process data, logged as students complete programming tasks, has furthered the field of computing education research in many ways, including identifying common programming errors (Brown & Altadmri, 2014a) and detecting plagiarism (Hellas et al., 2017). However, there are few common standards for how such data should be collected, represented, or shared, making it more difficult for researchers to collaborate, replicate findings, and share tools. Initiatives such as the PSLC Datashop (Koedinger et al., 2010) provide a common data format and tools to store, analyze, and share *generic* educational data. However, programming datasets have a number of distinct, domain-specific features, which make it difficult to use generic formats. Programming datasets may track entire projects with multiple files, and interpreting them often requires specific metadata, such as the version of the IDE or compiler. Programming data collection tools, such as BlackBox (Brown et al., 2014b) and CloudCoder (Papancea et al., 2013), have addressed this problem by defining their own data formats, but these are system-specific and not widely adopted.

In this paper, we present ProgSnap2: a standardized format for logging programming process data, which we have developed and are currently refining. ProgSnap2 builds on the original Progsnap format (Hovemeyer et al., 2017)[1] by representing a richer set of event data types and using a "flat" representation more suitable for direct analysis by statistics software. The goal of ProgSnap2 is to support researchers in sharing and analyzing programming process data. The format was designed

[1] See the full specification for the original Progsnap at: http://cloudcoderdotorg.github.io/progsnap-spec/

to prioritize the needs of both the *data producer* and the *data consumer*. For the *data producer*, our goal is to make exporting data straightforward, with a default structure to encourage best practices (e.g. what to log and how), a small set of required elements, and extensibility to support a variety of datasets. For the *data consumer*, our goal is to make importing and analyzing data straightforward, while making explicit how the data were logged, and any caveats or oddities that might impact analysis.

We see three primary use cases for the ProgSnap2 format. **1) Sharing Data**: There is a high cost to sharing unstandardized data. Both parties must invest time for the consumer to understand and parse the new format. A common format lowers these barriers, while improving the quality of new and existing logging systems by defining a standard set of events and attributes to log. Efforts to standardize the format and storage of learning data in other domains have led to datasets and research efforts that spanned multiple researchers and institutions (Koedingr et al., 2010). **2) Sharing Analysis Code**: A common format also allows researchers to write analysis code that can be shared and reused on new datasets that have the same format. This enables researchers to collaborate, even when sharing data is not possible (e.g. for privacy reasons). Publishing analysis code can also increase the replicability of computing education research and encourage the development of shared analysis libraries. For example, many researchers use the Error Quotient (Jadud, 2006) to quantify learners' compilation behavior. A shared implementation of the Error Quotient, capable of operating on any dataset in the common format, would save effort and ensure a consistent definition. **3) Sharing Tools**: A number of data-driven tools have been developed to support computing classrooms, such as student models (Yudelson et al., 2014) and on-demand hints (Rivers and Koedinger 2017; Price et al, 2017). A common input data format would allow these tools to be more easily shared, reused, and composed together. This raises the possibility of publishing these tools as *services* that any researcher can utilize, for example allowing any programming environment to employ an adaptive student model by sending its data to the appropriate service.

2 PROGSNAP2

A ProgSnap2[2] dataset consists of logs and relevant data that capture how users interacted with a programming or learning environment. A dataset includes a *main event table*, a *metadata table* and optional *link tables* to reference outside resources, all represented as CSV files. A dataset also contains a *code repository* containing sequential snapshots of students' code and optional auxiliary *resources* (e.g. assignment descriptions). We chose to define most elements of the dataset as directly parsable CSV files, rather than using a database, with the goal of making analysis as straightforward as possible.

2.1 Main Event Table

The central component of a dataset is the *main event table*, which represents a collection of events that took place in the programming environment. These events can represent both fine-grained interactions, such as individual keystrokes, and high-level actions, such as entire problem attempts,

[2] The full specification for ProgSnap2 is available at: http://bit.ly/ProgSnap2

depending on the granularity of the logging system. Each row in the table represents one event, and each column represents an event property. ProgSnap2 defines a small set of mandatory columns:

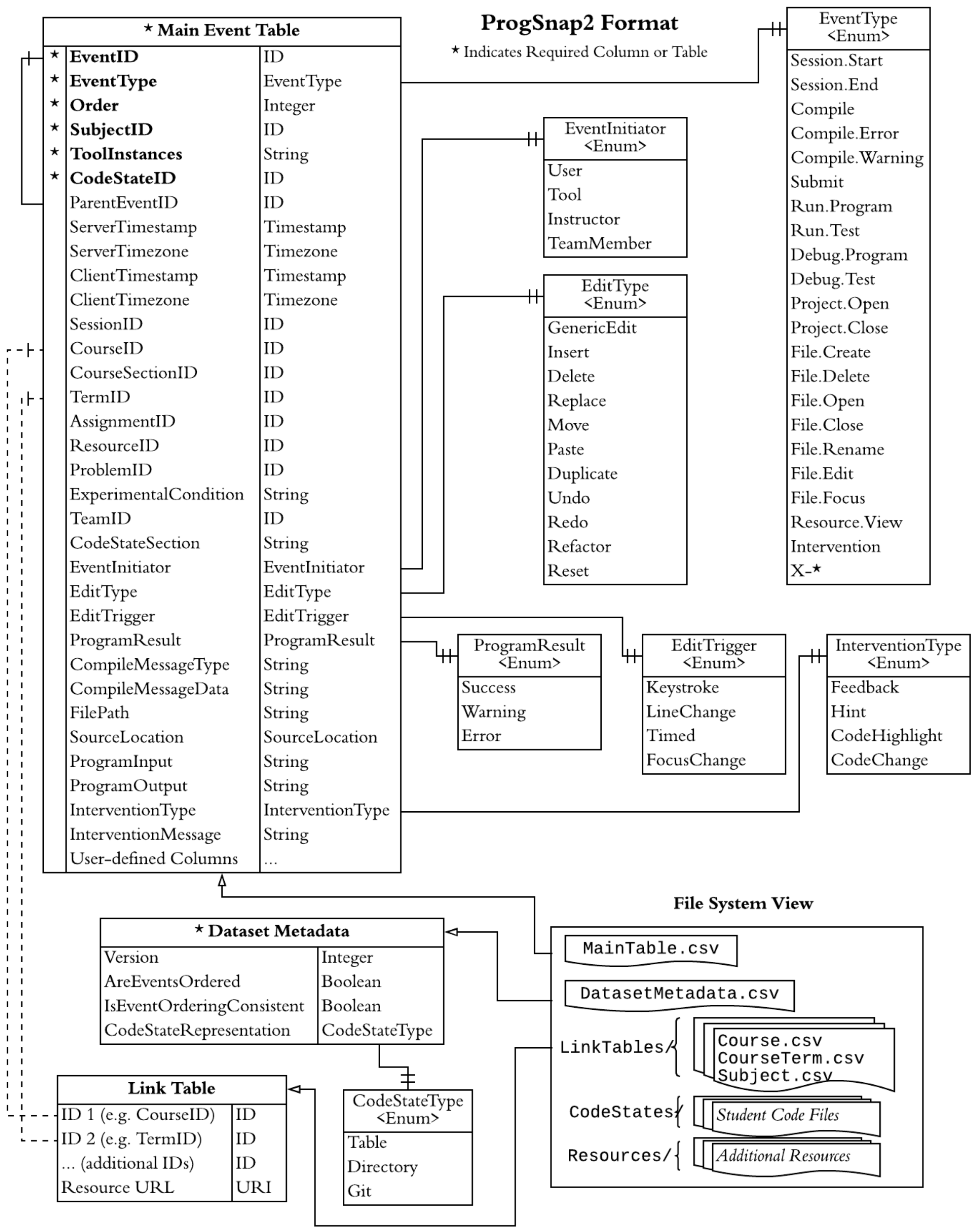
- **EventType**: an enumeration value indicating the type of event; examples include “Session.Start”, “File.Edit”, “Compile”, “Compile.Error”, “Submit”, and “Run.Program”
- **EventID**: the unique ID of the event
- **Order**: the chronological ordering of the event compared to others (may be approximate)
- **SubjectID**: the ID of the human subject (or group) associated with the event
- **ToolInstances**: a string indicating the names and versions of tools associated with the event
- **CodeStateID**: the ID of a snapshot of the source code and resources when the event occurred

ProgSnap2 also defines a variety of optional columns with standard names. Some columns may not apply to all datasets (e.g. CourseID) and can be omitted. Others apply to a specific subset of events, and can be included for only these events (e.g. CompileMessageType is only appropriate for “Compile.Error” events), creating a *sparse* table. Data producers are encouraged to include as many optional columns and as much detail as possible. They can also define new columns when needed. Examples of optional columns include:

- **ParentEventID**: the EventID of a “parent” event, to represent causal relationships; for example, a “Compile.Error” event would typically have a “Compile” event as its parent
- **TermID**, **CourseID**, **CourseSectionID**, **AssignmentID**, **ProblemID**: these provide contextual information for the associated event, which may be found in a "Link Table" (described below)
- **EditType**, **EditTrigger**, **CodeStateSection**: these respectively describe how code was edited (e.g. typing, paste, undo), the reason it was recorded, and where the edit took place
- **ProgramInput**, **ProgramOutput**, **CompileMessageType**, **CompileMessageData**: these record relevant information about how the code was compiled and run
- **ExperimentalCondition**, **InterventionType**, **InterventionMessage**: these record data about experimental conditions and interventions used in research studies

2.2 Metadata, Link Tables and Resources

The **Dataset Metadata** is a mandatory CSV file specifying the global properties of the dataset as key/value pairs. Currently, only a few global properties are defined, including the ProgSnap2 version number, whether event ordering is known, and which code state representation is used. **Link Tables** are *optional* files used to associate contextual ID values (or combinations of IDs) with a *resource* (defined below) providing more information. For example, a link table could associate a TermID/CourseID pair with the URL of a course website for that course and term. AAnother optional file, a **Resource,** is an arbitrary data blob, identified by a URL in a Link Table, which can be either external (accessed via the internet) or internal (local to the dataset). As with Link Tables, the inclusion of Resources is *optional* but encouraged. Where possible, we also encourage data producers to use *internal* resources (e.g. saving a static version of the course website in the example above) to ensure they are not lost or changed.



⋆ Main Event Table		
⋆	**EventID**	ID
⋆	**EventType**	EventType
⋆	**Order**	Integer
⋆	**SubjectID**	ID
⋆	**ToolInstances**	String
⋆	**CodeStateID**	ID
	ParentEventID	ID
	ServerTimestamp	Timestamp
	ServerTimezone	Timezone
	ClientTimestamp	Timestamp
	ClientTimezone	Timezone
	SessionID	ID
	CourseID	ID
	CourseSectionID	ID
	TermID	ID
	AssignmentID	ID
	ResourceID	ID
	ProblemID	ID
	ExperimentalCondition	String
	TeamID	ID
	CodeStateSection	String
	EventInitiator	EventInitiator
	EditType	EditType
	EditTrigger	EditTrigger
	ProgramResult	ProgramResult
	CompileMessageType	String
	CompileMessageData	String
	FilePath	String
	SourceLocation	SourceLocation
	ProgramInput	String
	ProgramOutput	String
	InterventionType	InterventionType
	InterventionMessage	String
	User-defined Columns	...

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-⋆

EventInitiator <Enum>
User
Tool
Instructor
TeamMember

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

ProgramResult <Enum>
Success
Warning
Error

EditTrigger <Enum>
Keystroke
LineChange
Timed
FocusChange

InterventionType <Enum>
Feedback
Hint
CodeHighlight
CodeChange

⋆ Dataset Metadata	
Version	Integer
AreEventsOrdered	Boolean
IsEventOrderingConsistent	Boolean
CodeStateRepresentation	CodeStateType

CodeStateType <Enum>
Table
Directory
Git

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

Figure 1: A diagram of the ProgSnap2 Format. Lines connect Enum types to their possible values, files to their respective tables, and IDs to their definitions in the Main Event Table.

2.3 Code State Representations

ProgSnap2 is intended to represent data from a variety of programming activities, from single-function exercises to complex final projects to block-based programs. To capture this diverse data, ProgSnap2 supports three source code representations: *Git*, *Directory*, and *Table*. This choice allows data producers to use the most appropriate representation, while constraining that choice to formats which are easily processed. Each format maps a CodeStateID value to a *code state*, which is simply a collection of one or more files with an optional directory structure. In the *Git* format, code states are represented as commits in a Git[3] repository stored within the dataset. This format is appropriate for datasets where code states may consist of a relatively large number of files. In the *Directory* format, each CodeStateID maps to the name of a directory stored within the dataset which contains a collection of all files that are part of the code state. This format is appropriate for datasets where code states contain a small number of files. In the *Table* format, a dedicated CSV file maps CodeStateID values to text strings. This format is only appropriate for datasets where each code state consists of a single text file, and where the amount of data per code state is small.

3 CASE STUDIES

To explore and evaluate the standard, we implemented ProgSnap2 data exporters for two open source autograding systems, Virtual Programming Lab (VPL)[4] and CloudCoder (Papancea et al., 2013).

Virtual Programming Lab (VPL): As learners make submissions and receive feedback, the VPL maintains a downloadable log stored as a zip file of directories, with each directory representing one student. These directories contain a timestamped series of paired folders representing student code submissions and their associated compilation information. Our tool[5] consumes these logs and produces ProgSnap2 compliant archives. During conversion, each submission is decomposed into a sequence of events (“Submission”, “Compile”, “Compile.Error”, etc.). Each event is assigned a numerically ascending, unique Event ID, and necessary fields are assigned, such as the Server Timestamp, Event Type, and event-specific data like the code for a “Submission” event or the compiler’s output during a “Compile.Error” event.

We faced a few challenges during development, such as mapping VPL’s data to ProgSnap2 events. For example, a number of event types relate to compilation. Given that Python is not truly “compiled”, should we use a “Compile” event or a “Run.Program” event? When students run their code and receive autograder feedback, would a “Run.Test” event be appropriate, since autograding is more than just unit testing? If there is an error, VPL will still offer feedback to the student. Is this an “Intervention” or just the “ProgramResult”? When a Grade is assigned, is that also an “Intervention,” or does the standard need a new Event Type? Most of these challenges were easily resolved, though some led to ongoing conversations that may be settled as we develop other conversion tools.

[3]Standard libraries for extracting a commit from a Git repository (git-scm.com) can be found at libgit2.org
[4] http://vpl.dis.ulpgc.es/
[5] Source code for the tool is available at: https://github.com/CSSPLICE/progsnap2

CloudCoder: Exporting CloudCoder data to the ProgSnap2 format was fairly straightforward and mostly involved mapping CloudCoder's internal event representation to that of ProgSnap2. One challenge we encountered is that a single CloudCoder event can yield multiple ProgSnap2 events in some cases. To address this, the CloudCoder event IDs are multiplied by a constant to create a gap in the namespace where multiple derived events can be situated without conflict. We also had difficulty defining Session.Start and Session.End events, as CloudCoder does not directly record sessions. We considered defining them based on how much time elapsed between recorded CloudCoder events, but eventually decided to omit them. We felt it would be more appropriate for the data consumer to develop his or her own heuristics to reconstruct sessions during analysis.

4 FUTURE WORK AND CALL FOR COLLABORATION

We are currently working to refine ProgSnap2 by exporting datasets from additional programming environments, including PCRS (Zingaro et al., 2013) and iSnap (Price et al., 2017). However, our primary goal for the format is to facilitate collaboration through the sharing of data, analysis code, and data-driven tools. We plan to evaluate the utility of ProgSnap2 through these efforts, and we invite researchers interested in sharing programming data for collaboration to contact the authors.

5 REFERENCES

Brown, N. C. C., & Altadmri, A. (2014a). Investigating Novice Programming Mistakes: Educator Beliefs vs Student Data. In Proceedings of the Tenth International Computing Education Research Conference (pp. 43–50). https://doi.org/10.1145/2632320.2632343

Brown, N. C. C., Kölling, M., McCall, D., & Utting, I. (2014b). Blackbox: A Large Scale Repository of Novice Programmers' Activity. In Proceedings of the ACM Technical Symposium on Computer Science Education (pp. 223–228). https://doi.org/10.1145/2538862.2538924

Hellas, A., Leinonen, J., & Ihantola, P. (2017). Plagiarism in Take-home Exams : Help-seeking , Collaboration , and Systematic Cheating. In Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education (pp. 238–243). https://doi.org/10.1145/3059009.3059065

Hovemeyer, D., Hellas, A., Petersen, A., & Spacco, J. (2017). Progsnap: Sharing Programming Snapshots for Research. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (p. 709).

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. In Proceedings of the Third International Workshop on Computing Education Research (pp. 73–84). https://doi.org/10.1145/1151588.1151600

Koedinger, K. R., Baker, R. S. J., Cunningham, K., & Skogsholm, A. (2010). A Data Repository for the EDM community: The PSLC DataShop. In C. Romero, S. Ventura, M. Pechenizkiy, & R. Sj. Baker (Eds.), Handbook of Educational Data Mining (pp. 43–55). CRC Press. https://doi.org/doi:10.1201/b10274-6

Papancea, A., Spacco, J., & Hovemeyer, D. (2013). An Open Platform for Managing Short Programming Exercises. In Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (pp. 47–52). New York, NY, USA: ACM. https://doi.org/10.1145/2493394.2493401

Price, T. W., Dong, Y., & Lipovac, D. (2017). iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In Proceedings of the ACM Technical Symposium on Computer Science Education.

Price, T. W., Zhi, R., & Barnes, T. (2017). Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming. In Proceedings of the International Conference on Educational Data Mining.

Rivers, K., & Koedinger, K. R. (2017). Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. International Journal of Artificial Intelligence in Education, 27(1), 37–64. Retrieved from http://link.springer.com/10.1007/s40593-015-0070-z

Yudelson, M., Hosseini, R., Vihavainen, A., & Brusilovsky, P. (2014). Investigating Automated Student Modeling in a Java MOOC. In Proceedings of the International Conference on Educational Data Mining (pp. 261–264).

Zingaro, D., Cherenkova, Y., Karpova, O., & Petersen, A. (2013). Facilitating Code-writing in PI Classes. In Proceeding of the 44th ACM Technical Symposium on Computer Science Education (pp. 585–590). https://doi.org/10.1145/2445196.2445369