December 2020

Accelerating dataset assembly

Primer and guide to the Dataset Assembly Tool

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

New Zealand Government

Author

Simon Anastasiadis

Acknowledgements

Raj Kulkarni[1], Doug Lambert[2], Integrated Data Unit[3]

1 = Social Wellbeing Agency, 2 = Inland Revenue, 3 = Statistics New Zealand

Creative Commons Licence

Liability

Citation

Social Wellbeing Agency 2020. Accelerating dataset assembly: Primer and guide to the Dataset Assembly Tool. Wellington, New Zealand.

ISBN 978-0-473-55795-9 (online)

Published in December 2020 by
Social Wellbeing Agency
Wellington, New Zealand

Summary: New tool and processes for dataset preparation

Good analytic and research projects combine information in ways that lead to new insights and actions. The preparation for such projects often involves drawing data together into a single dataset ready for analysis. However, as the number of data sources increases so does the complexity of preparation. Without a consistent method for assembling analysis-ready datasets, this process can become time-consuming, expensive, and error prone.

In response to this challenge, the Social Wellbeing Agency has developed the Dataset Assembly Tool. By standardising and automating the data preparation and dataset assembly stages of analytic projects, the tool helps staff deliver higher quality work faster. We have already found the use of this tool significant: In a recent project a panel dataset was produced from 25 sources within a week of technical work beginning; and in another project the primary dataset was updated and recreated every hour as staff collaborated checking and polishing the dataset.

The Dataset Assembly Tool is now available for other researchers and analysts to use. Alongside the tool, we have developed a project structure and workflow to support its effective use.

While the tool can be used within a single project, greater impact is anticipated from its reuse across projects and organisations. As more researchers use the tool and its patterns, the increased consistency will strengthen the entire research community by making it easier to collaborate, sharing knowledge and code.

This document describes the tool, its benefits, the associated patterns and workflow, and the principles that shaped its development. A worked example applying the tool in a simplified case is given in the appendix.

Contents

Summary: New tool and processes for dataset preparation *3*

The value of faster dataset preparation *5*

Accelerating dataset assembly *6*

The tool follows a common assembly pattern **6**

Control files ensure tool is agnostic to input data **7**

The tool is applicable to use cases other than the IDI **7**

Project structure and workflow align with tool *8*

Structure separates data preparation and analysis **8**

An iterative researcher workflow is supported **9**

Circulate event definitions for reuse **10**

Architecture details are available *10*

User input is validated before use **11**

The tool uses a single core query for assembly **12**

Format of inputs and outputs **12**

Principles guide use and future design *15*

Some effort is required to get the full benefit of a tool **15**

Design principles for tools increases their value **16**

Appendix: Worked example *18*

The value of faster dataset preparation

Good analytic and research projects combine information in ways that lead to new insights and actions. The preparation for such projects often involves drawing data together into a single dataset ready for analysis. However, as the number of data sources increases so does the complexity of preparation. Without a consistent method for creating an analysis-ready dataset, this process can become time-consuming, expensive, and error prone. This is a particular concern for researchers using the Integrated Data Infrastructure (IDI) where a single project may involve combining data from dozens of different tables, each with its own distinct structure.

In response to this challenge, the Social Wellbeing Agency has developed the Dataset Assembly Tool. By standardising and automating the data preparation and dataset assembly stages of analytic projects, the tool helps staff deliver higher quality work faster. Though it can be used by itself, the assembly tool is supported by recommended patterns and workflow that increase its overall value:

- **Accelerated assembly** – We estimate that the tool more than halves the time of creating research ready datasets, considerably reducing the cost to commence research.
- **Rapid iteration** – A faster process supports ongoing improvement, enabling researchers to update datasets multiple times a day as errors are corrected or new requirements identified.
- **Fewer errors** – A standardised process can be rerun with confidence that it will continue to perform. It also reduces the number of places where errors can occur, reducing the time spent finding faults.
- **Scalable development** – By encouraging independence between inputs, the tool enables researchers to work in steps that are easy to understand and manage.
- **Single step construction** – As the research dataset is combined together in one step, staff avoid the hazard of tangled, bespoke assembly patterns. The tool can be a starting point for data workflows.
- **Definition reuse** – The measures defined for a project represent expert knowledge and have value beyond the project they were created for. By separating definitions from assembly, the tool enables easy reuse of definitions between projects and sharing of definitions with other researchers.
- **Increased collaboration** – By providing standard patterns, the tool encourages consistency between staff, making collaboration on a single project, or handover of projects more straightforward.
- **New opportunities for innovation** – When consistent processes are adopted, opportunities arise for further improvement across staff and projects by spreading good practices and resources.

The Dataset Assembly Tool is now available for other researchers and analysts to use. We have already found the use of this tool significant: In a recent project a panel dataset was produced from 25 sources within a week of technical work beginning; and in another project the primary dataset was updated and recreated every hour as staff collaborated checking and polishing the dataset. While useful within a single project, greater impact is anticipated from its reuse across projects and organisations. As more researchers use the tool and its patterns, the increased consistency will strengthen the entire research community by making it easier to collaborate, sharing knowledge and code.

This document describes the tool, the associated patterns and workflow, and the principles that shaped its development. A worked example applying the tool in a simplified case is given in the appendix. Technical staff are encouraged to review the separate introduction and training presentation that covers the practical aspects of configuring and running the tool.

Accelerating dataset assembly

Improving dataset preparation requires more than faster code. An effective tool must also be easy for other researchers to understand and use. This ensures it saves researcher time during and after data preparation, which is much more valuable than just reducing computer execution time when joining datasets together.

Before developing the assembly tool, we reflected on the design of existing tools, other attempts to improve data preparation, and common patterns that are followed when conducting analysis. The existing tools we considered included the Social Investment Analytical Layer (SIAL) and Social Investment Data Foundation (SIDF) that the Social Wellbeing Agency recently retired. From these reflections we formulated the following researcher needs that a suitable tool would fulfil:

1. Construct analysis ready, rectangular datasets from a range of data sources with ease.
2. Keep different project stages separate and independent from each other.
3. Design new variables without being required to fit a specific format or style.
4. Add new variables to a project dataset with ease.
5. Transfer the definitions of variables between projects as easily as copy & paste.
6. Trace where data has originated from.
7. Replicate parts of the automated assembly manually for validation and quality assurance.
8. Use and understand the tool without needing to learn a new programming language.
9. Debug only a small part of the process when an error occurs and know rapidly which part of the process is at fault.

These nine needs can be summarised as ease of use, adoption, validation, and error correction. Because we designed for these needs, new staff have become comfortable using the tool within two days with minimal input from existing users.

The tool follows a common assembly pattern

The Dataset Assembly Tool is designed around the most common dataset assembly pattern we have observed: who-when-what. This pattern is well suited to creating research-ready rectangular datasets, for both cross-section and panel data. However, it will not be ideal for research that uses non-rectangular formats.

Understanding the assembly pattern is important for knowing whether the tool is suitable for your application, and for preparing your data before assembly. The who-when-what pattern works by combining two inputs:

1. A **population** definition consisting of identities, start dates, and end dates for the time period(s) of interest.
2. An **event** definition consisting of an identity, when it occurred, and some detail about the event.

These two inputs are combined to produce a single output: For every identity, a summary of the events that occurred between the start and end dates. In a rectangular dataset, with one row per identity, this

output would appear as a single measure – one column. By combining multiple event definitions with the same population definition, a complete dataset can be constructed.

To demonstrate this pattern, consider the following two inputs:

1. The **population:** all college leavers (identities) from the date of their last attendance (start date) until one year later (end date).
2. The **event:** benefit payment records, including who received the payment (identity), when it was made (date), and how much was paid (what).

Combining these two inputs could produce summary outputs like the total number of benefit payments received, and the total amount received, for college leavers in their first year after leaving college.

By design, the filtering of events to only those identities and dates of interest happens at the time of assembly. This means that the population and event definitions can be developed independently. The population definition is specific to the research scope. The event definitions are not specific to the research scope and instead should be the highest quality definition of the event that the researcher can construct. This means that the same high-quality event definition can be reused across multiple research projects by combining it with different population definitions.

Control files ensure tool is agnostic to input data

A core part of meeting researchers' needs was ensuring that the assembly tool was general-purpose and not restricted to a specific choice or format of input data. Furthermore, we needed to minimise any requirement for users to know a specific programming language. We accomplished this via the use of control files – Excel tables with specific structures. When read by the assembly tool, these control files tell it what data to fetch, from where to fetch it, and how to combine it.

The control files have applications beyond the running of the tool. As they contain the core metadata required for assembling the analysis dataset, the control files are the primary reference for researchers seeking to review how measures relate to source data. While not mandatory, with the addition of a description column, the control files can also act as a simple data dictionary.

Because the use of Excel is so widespread, and independent of the programming languages used for data preparation and analysis, it is accessible to researchers regardless of their preferred programming language. As a result, users are almost completely insulated from the programming language of the tool with only simple text – such as control file names – needing to be input directly into the tool.

A range of resources are available for new users to learn how to configure the control files: A detailed description of all the tool inputs is provided later in this document, there is a worked example in the appendix, and an example with code is distributed with the tool.

The tool is applicable to use cases other than the IDI

Although the assembly tool has been developed for assembling individual-level data in the context of the IDI, it is not restricted to this use. We note two other use cases here:

- The identities need not be individuals so long as they are consistent across input tables. This means the tool could be used to assemble information for geographic areas (e.g. regions or meshblocks) or for entities (e.g. hospitals, schools, or businesses).
- Due to differences between database languages the initial distribution of the tool requires the input data to be stored in a Microsoft Server database (as they are in the IDI). However, adjusting the tool to work with different types of database is straightforward, making it applicable outside the IDI.

Project structure and workflow align with tool

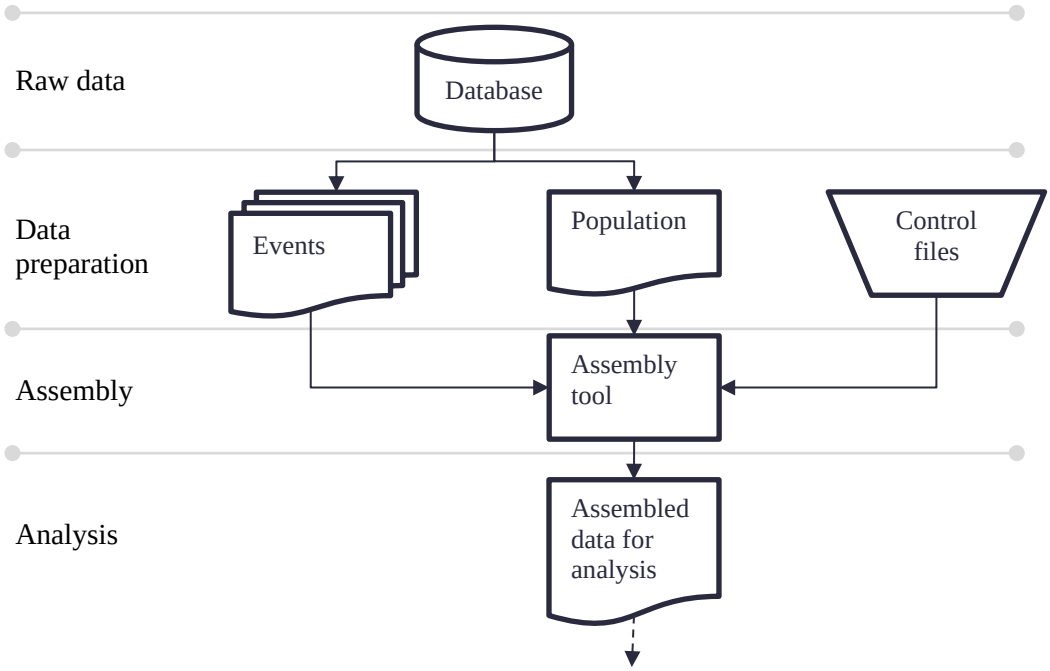
Alongside the assembly tool, we have developed a project structure and workflow to support its effective use. While the tool can be adopted without the associated processes, we recommend new users understand the intended structures we use around the assembly tool. This will help them get the most benefit from it and, as more researchers adopt these processes, will increase the value to the wider research community by making it easier to share knowledge and code.

Structure separates data preparation and analysis

We have observed significant confusion and error occur in projects where the line between data preparation and data analysis becomes blurred. To avoid this, we use the assembly tool as the boundary between the data preparation and data analysis stages of a project - everything before is data preparation and everything after is analysis.

Following this distinction, we find our projects naturally fit into the project structure demonstrated in Figure 1. Before the assembly tool, the project consists of a series of distinct data objects (population and event definitions), which are independent, with the key details of these definitions entered into the controls for the tool. When the assembly tool is run, a single rectangular table is produced as the starting point for all the subsequent analysis.

Figure 1: Recommended project structure



The arrows in the diagram represent the flow of data. While a researcher may revisit and change earlier stages of a project as a result of insights at a later stage, when code is executed data only moves in one direction. This protects against self-referential loops and means that whenever an error is encountered, analysis can resume from the last stage that was error free.

This structure is also reflected in the choice of folders we use to store our projects. Six primary folders are common in projects using the assembly tool. Though their names may vary, their purposes are consistent across projects:

- **Data exploration** stores all work done to inform event or population definitions. While not always necessary, some datasets require detailed investigation before they can be used with confidence. Code, notes, and findings from such work are saved for reference.
- **Documentation** stores project and data documentation. Data dictionaries, project overviews, workplans, quality assurance review records, and other reference material are saved here.
- **Data preparation** stores completed definitions of both events and populations. It may feature subfolders to separate events and populations, or to separate different types of events.
- **Dataset assembly tool** stores the tool. While initial projects using the tool stored it in the same folder as the analysis, this clutters the workspace. Our recommendation is to store the tool separately, and to place only the file that executes the tool, along with the control files, in the analysis folder.
- **Analysis** stores all the work involved delivering the research results once a rectangular dataset has been produced. Some data cleaning is necessary at this step, as it is often more efficient to define events generally during data preparation and refine them to the specific format required for the research question early in the analysis. This is especially the case for composite measures.
- **Output** stores completed research output. Within the IDI environment, this includes copies of output submitted for release.

The structure we describe above is general and may need to be adapted to fit a specific project or researcher style. When adapting, the key aspects from this structure that we recommend preserving are the independence of different event definitions during data preparation, and the separation of the preparation, assembly tool, and analysis stages.

An iterative researcher workflow is supported

The linear structure demonstrated in Figure 1 is not intended to imply that a researcher's process is linear. Our experience suggests the opposite, that the use of the assembly tool enables researchers to use a more iterative, or circular, workflow. This is most obvious when adding new measures or adjusting existing ones, where we observe the following pattern:

1. Identify the need for a new or updated measure (or population)
2. Explore data
3. Propose and validate a new or updated measure
4. Construct the events for the new measure as a database object (such as a table or a view in SQL)
5. Add the new measure to control file
6. Rerun the assembly tool

7. Load dataset into analysis
8. Resume analysis

An important part of this workflow is how the development of a new measure (steps 2-4) is separated from the analysis (steps 7 onwards) by the addition of the new measure to the control file and the rerunning of the assembly (steps 5 and 6). This separation supports increased collaboration as researchers can define new measures in parallel, or a researcher can explore the existing analysis dataset while another develops measures.

We encourage IDI researchers to use the native database language – SQL – for constructing and defining events. This improves accessibility and reuse of events as most researchers will have the skill required to understand a definition because they use the database language to explore the data.

If the logic required to create a definition is straightforward, then we also encourage researchers to use views in SQL, as opposed to tables. This saves database space as views fetch data from source and reformat it when required, instead of writing reformatted data to disc. We consider a definition to be straightforward if it contains at most one join and at most one nested query.

Circulate event definitions for reuse

When event definitions are constructed independent of the study population and time period, they are easy to reuse. Rather than redeveloping the same measure for multiple projects, a single definition can be constructed and used across projects. In addition to saving effort, this helps refine existing work resulting in a single, well-established, high-quality definition.

Where researchers belong to a wider community (as they do with the IDI), the publishing and circulating of these definitions brings further benefits. As existing definitions reflect the subject matter expertise of their authors (and if refined by the community, of multiple authors and peer review) they provide researchers with rapid access to expertise it would otherwise be costly to find or develop themselves. Authors who share definitions can provide a single resource instead of managing a series of individual requests for support, and such definitions act as another form of publication, increasing the uptake of their work.

Whether circulating definitions for use by a wider research community, or storing definitions for personal reuse, we recommend recording metadata along with the code that constructs the definition. While each research community will develop their own standards, at a minimum we recommend this includes the definition's intended purpose, dependencies, specific design decisions, and limitations. Establishing such standards is a significant step towards the creation of a common library of definitions.

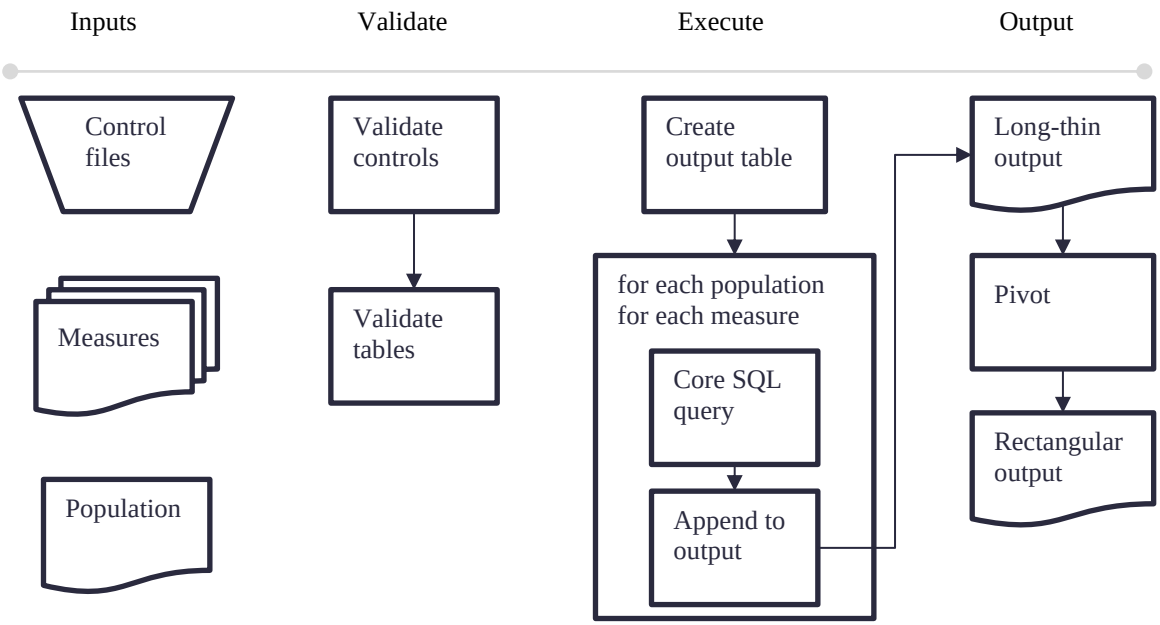
Architecture details are available

Developers and researchers seeking to understand the design or execution of the tool in more detail are welcome to review the code. For those wanting to understand how the tool operates without inspecting it closely, this section provides additional details and a worked example can be found in the appendix.

Figure 2 provides an overview of the tool architecture. This shows the key tasks and database objects involved at each stage. Note that the initial assembly produces a long-thin table – one row per person, time

period, and measure. The standard rectangular table preferred for analysis – one row per person and time period, and one column per measure – is created at the very end via a pivot operation.

Figure 2: Architecture overview of dataset assembly tool



While the tool is written in the R programming language, when the tool is run it executes code in both R and SQL. This is deliberate – SQL is well suited for manipulating large quantities of data, but less suited for validating and iterating over user input. To use each language for its strength, R creates a connection to the database server. We can then use R to manage the running of the tool and to generate SQL code that is passed to the database server for execution. This connection between R and SQL can be used for other applications. Interested researchers may find it helpful to review the helper/supporting functions that are provided as part of the tool.

User input is validated before use

For a general-purpose tool to function smoothly, it must not only assemble data but also validate all user inputs. This helps provide meaningful feedback to users, reduces the likelihood of error during assembly, and provides clarity and confidence to users.

Each time the tool is run it first validates both control files, before accessing each SQL table given in the control files and checking the contents are consistent with the control file. Any errors or inconsistencies at this stage produce warning messages for the user and prevent the assembly running. The following checks are carried out:

1. Format of the input tables is correct
2. The contents of input tables are formatted correctly
3. Database objects specified in the input tables exist
4. Database objects specified in the input tables are accessible
5. Database objects specified in the input tables have the required columns
6. Summary type of events is consistent with input data format (e.g. ‘sum’ only applied to numeric values)

Several controls exist to support researchers' use of the tool's validation features. The simplest of these controls enable a researcher to run only the checks without the assembly. An additional control exists to support the checking of output without having to run the entire assembly. When activated, '*Development mode*' limits the number of records returned of each output type. This lets researchers create only a small sample of the data, useful for validating end-to-end execution.

The tool uses a single core query for assembly

At the heart of the assembly execution is a single SQL query. The purpose of this query is to fetch all events that correspond to the identities and data ranges given in the population table and return them in a consistent format.

Figure 3: Core SQL assembly query

```
APPEND TO [OUTPUT TABLE]

SELECT p.identity
    , p.population_label
    , p.start_date
    , p.end_date
    , p.period_label
    , m.measure_label
    , FUNCTION(m.value) AS measure_value

FROM [population_table] AS p
INNER JOIN [measure_table] AS m
ON p.identity = m.identity
AND p.start_date <= m.end_date
AND m.start_date <= p.end_date

GROUP BY p.identity, p.population_label, p.start_date,
         p.end_date, p.period_label, m.measure_label
```

Figure 3 gives a pseudo-SQL query that captures the core of the code that is run for every population/measure pair. This produces the long-thin output table, with the rectangular output table produced using standard pivot code. The summary `FUNCTION` (last row under the `SELECT` keyword) changes depending on how events are summarised into measures (for example, counting records or summing values).

Format of inputs and outputs

This section sets out the details for each input to, and output from, the dataset assembly tool. Its purpose is to guide users preparing tool inputs and to set expectations for tool output.

For all inputs we use the same convention to distinguish between text and SQL object names. This distinction is important because many inputs can take both formats. We have chosen to use specific delimiters to distinguish between them:

- Text input is delimited using double quotes: "example"
- SQL input is delimited using square brackets: [example]

To highlight the difference, consider the difference between:

- Label = "before"
- Label = [before_after]

The first example will use the text ‘before’ as the label. The second example will look for a column called ‘before_after’ and use the value found in this column as the label. If the column [before_after] exists and only contains the text ‘before’ then both approaches are equivalent.

Note that text strings which are not delimited, or that use some other choice of delimiter, will produce an error. Also take care when copy-and-pasting as Excel and most plain text-editors default to straight quotes ("accepted"), while MS Word defaults to curly quotes (“rejected”). These are different symbols and the tool only accepts straight double quotes.

Table 1 and Table 2 give the mandatory columns for the study population and measure control files respectively, along with their accepted inputs. In addition to these mandatory columns, we often include two further columns to record the file that constructs the source definition and a description of the intended measure.

Table 1: Required columns for population and period control file

Column name	Accepted inputs
Database_name	SQL only
Schema_name	SQL only
Table_name	SQL only
Identity_column (e.g. snz_uid)	SQL or text
Label_Identity (e.g. control, treatment)	SQL or text
Summary_period_start_date (type = date)	SQL or text
Summary_period_end_date (type = date)	SQL or text
Label_summary_period (e.g. year_1, year_2)	SQL or text

Table 2: Required columns for measures control file

Column name	Accepted inputs
Database_name	SQL only
Schema_name	SQL only
Table_name	SQL only
Identity_column (e.g. snz_uid)	SQL or text
Measure_period_start_date (type = date)	SQL or text
Measure_period_end_date (type = date)	SQL or text

Label_measure	SQL or text
Value_measure	SQL or text
Measure_summarised_by	MIN / MAX / SUM/ COUNT / EXISTS/ DURATION / HISTOGRAM / DISTINCT / MEAN
Proportional (true / false)	TRUE / FALSE

Table 3 gives a summary of the user controls that are entered into the run file (`run_assembly.R`) along with their accepted inputs. Note that the tool requires the absolute file path to work reliably. This is the path that starts from the root directory. An example of such a path can be produced in R using the 'get working directory' command: `getwd()`.

Table 3: User controls entered in code file

Column name	Accepted inputs
Absolute_path_to_tool	Location of the tool
Absolute_path_to_analysis	Location of run file and control files
Population_file	Name and extension of population and period control file
Measures_file	Name and extension of measures control file
Output_database	SQL only
Output_schema	SQL only
Long_thin_table_name	SQL only
Rectangular_table_name	SQL only
Overwrite_existing_tables	TRUE / FALSE
Development_mode	TRUE / FALSE
Run_checks_only	TRUE / FALSE
Info_to_print_to_console	default / all / details / headings / none

Table 4 gives the format of the long-thin output table produced by the assembly tool. Note that by default two output tables are produced: a long-thin table and a standard rectangular table. The rectangular table has the same first five columns as the long-thin table, but then has one additional column per unique value in the Label_measure column (the rectangular table is made by turning the labels in the Label_measure

column into column headings). Researchers who need only the rectangular table are encourage to delete the long-thin table to save database space.

Table 4: Structure of the long-thin output table

Column name	Format
Identity_column	Numeric
Label_Identity	Text
Summary_period_start_date	Date
Summary_period_end_date	Date
Label_summary_period	Text
Label_measure	Text
Value_measure	Numeric

Some users may find the three different labels (identity, period, and measure) provided by the tool unnecessary. As labels are user-defined, placeholder values can be assigned to unneeded labels. The use of three labels is to support the creation of panel data. For example: a project might consider the blood pressure and weight (two measures labels) for treatment and control groups (two identity labels), one week, one month, and one year after their initial appointment (three summary period labels).

Principles guide use and future design

We anticipate that the publication of the Dataset Assembly Tool will create further opportunities for innovation. Researchers and analysts will develop further tools and discover new ways to use existing ones. Our development and use of the tool have been shaped by a collection of principles. We provide these below to guide others as they identify and act upon new opportunities.

Some effort is required to get the full benefit of a tool

Analytic tools often work by shifting effort from researchers and analysts onto computers. In this way they benefit from the speed at which computers can carry out repetitive tasks. However, adopting a tool creates new tasks for staff to setup and oversee the running of the tool. The benefit of a tool is not that it eliminates all staff effort, but that it replaces a large amount of staff effort with much smaller effort, resulting in a net saving in staff time and effort.

From this perspective, we recommend the following three principles for researchers adopting a new tool:

1. Accept up-front effort for longer term gains

   Some up-front effort will always be required for staff to familiarise themselves with a new tool.

For the assembly tool, we have provided a range of documentation and training material to support staff to become comfortable using the tool. This material has brought new staff up to speed on the tool within two days. We encourage researchers adopting the tool to work through all the available material, including the worked example at the end of this document, the training presentation, and the example provided alongside the code.

2. Provide information on use and feedback for improvement

   Without evidence that the tool is of benefit to users, it will be difficult to justify its ongoing support, improvement, and the development of further resources that benefit the wider community.

   For the assembly tool, each time the tool is run it saves a small amount of information about the execution in a local log file. Sharing this log file at the end of each project is an easy way for researchers to provide information on their use of the tool back to the developer.

3. Use the tool as developed

   While the tool is open source and can be edited by users, this is discouraged. When staff use the same tool, then all can benefit from updates and improvements to the tool. However, if each researcher edits the code to fit their own needs, then the community no longer has a shared tool and instead has a mixture of bespoke code that shares a common ancestor.

   For the assembly tool, we recommend that staff build new tools, adapt the inputs to the tool, or apply further processing to the outputs from the tool instead of editing the code in the tool. As part of the tool, we provide a run file (`run_assembly.R`) that accepts user parameters and executes the core of the tool. This file is a suitable location where other code could be added to extend the usefulness of the tool to a researcher while keeping the core code of the tool consistent across all users.

Design principles for tools increases their value

The Dataset Assembly Tool aims to improve the data preparation phase of analytic projects, but there are other project phases that would benefit from the creation of further resources. While individual staff may create patterns, code, or templates that they reuse, when a tool is published it becomes available to other staff who are not known by the creator. This means that a successful tool needs to be easy for staff to learn and use independent of the developer. With this goal in mind, we recommend the following principles for staff creating new tools:

- **Build for the general purpose** – The need for a tool is often identified in a specific context, but the reuse of a tool requires it to be applicable to other contexts. Consider how the problem you are solving might vary and craft a solution that can handle (at least some of) this variation.
- **Use interfaces to simplify inputs and outputs** – An interface is a pattern for communicating with a programme without needing to know how the programme works. At its simplest level it says: "if your inputs match *this* pattern then I will work and will give you output according to *this* pattern."
- **One tool for one purpose** – It can be tempting to develop a single tool that meets all the needs for a project from beginning to end. However, many smaller independent tools (that communicate via interfaces) are better than a single all-encompassing tool. This reduces the complexity of the overall project, making development, maintenance, and understanding each tool easier.
- **Design for incomplete adoption** – While the full value of a tool may only be realised when users adopt processes that support its use, tools need to provide value even if associated practices and patterns are

not used. The same principle applies when publishing multiple, related tools: Each tool must provide value if adopted by itself.

- **Design for incomplete understanding** – If staff must fully understand a tool in order to use it, this creates a barrier to adopting the tool. New users to a tool may only want part of its functionality or only have time to learn some of its features. Guidance for inexperienced users and clear default settings support the progressive adoption of the tool, allowing users to benefit even while building their understanding.
- **Robust error handling** – Publishing a tool implies a minimum standard of quality. Error handling is part of this standard. Staff using a tool should not need to debug or inspect the workings of the tool in order to resolve errors. An effective way to do this is to validate the input interface and provide immediate corrective feedback to users when their input does not conform to the interface.
- **Tools are self-documenting** – Having a record of the inputs to a tool that produced a specific output helps staff validate outputs, identify and correct errors, and reproduce results.

Appendix: Worked example

Consider a database containing the following source tables:

Table 5: Example source table: [IDI_Sandpit].[project-code].[project_population]

snz_uid	registration_date	appointment_date	appointment_type
001	2012-03-01	2012-04-01	'fast appointment'
002	2018-01-01	2019-01-01	'delayed appointment'

Table 6: Example source table: [IDI_UserCode].[project-code].[accidents]

snz_uid	accident_date	accident_cause
001	2012-05-01	'fall'
001	2012-06-01	'vehicle'
002	2016-04-01	'April fools trick'
002	2017-04-01	'April fools trick'
002	2018-04-01	'April fools trick'
002	2019-04-01	'April fools trick'

Table 7: Example source table: [IDI_Clean_20190420].[msd_clean].[benefit_payment]

snz_uid	benefit_start_date	benefit_end_date	payments
001	2012-02-01	2012-04-01	100
001	2013-05-01	2014-02-01	1000
002	2018-02-01	2018-03-01	20
002	2018-04-01	2018-05-01	20
002	2018-06-01	2018-07-01	20
002	2018-06-15	2018-07-15	20
002	2019-06-01	2019-09-01	300

Then this input in the control files:

Table 8: Example population and period control file: population_and_period_table_example.xlsx

Database _name	Schema _name	Table _name	Identity _column	Label_ identity	Summary_period _start_date	Summary_period _end_date	Label_summary _period
[IDI_ Sandpit]	[project-code]	[project_ population]	[snz_uid]	"before"	[registration_date]	[appointment_date]	"registration to appointment"
[IDI_ Sandpit]	[project-code]	[project_ population]	[snz_uid]	"after"	[appointment_date]	"2019-12-31"	[appointment_type]

Table 9: Example measures to assemble control file: measures_to_assemble_table_example.xlsx

Database _name	Schema _name	Table _name	Identity _column	Measure_period _start_date	Measure_period _end_date	Label_ measure	Value_ measure	Measure_ Summarised _by	Propor-tional
[IDI_UserCode]	[project-code]	[accidents]	[snz_uid]	[accident_date]	[accident_date]	"num_ accidents"	"1"	COUNT	FALSE
[IDI_Clean]	[msd_clean]	[benefit_ payment]	[snz_uid]	[benefit_ start_date]	[benefit_ end_date]	"total_ benefit"	[payments]	SUM	TRUE

With these code settings:

```
Output_database = '[IDI_Sandpit]'
Output_schema = '[project-code]'
Long_thin_table_name = '[project_population_prepared]'
Rectangular_table_name = '[project_population_ready]'
Input_population_and_period_table = 'population_and_period_table_example.xlsx'
Input_measures_to_assemble_table = 'measures_to_assemble_table_example.xlsx'
```

Produces the following output:

Table 10: Example long-thin output table: [IDI_Sandpit].[project-code].[project_population_prepared]

Identity _column	label_identity	summary_period _start_date	summary_period _end_date	label_summary_period	label _measure	value _measure
001	'before'	2012-03-01	2012-04-01	'registration to appointment'	'num_accidents'	0
001	'before'	2012-03-01	2012-04-01	'registration to appointment'	'total_benefit'	50
002	'before'	2018-01-01	2019-01-01	'registration to appointment'	'num_accidents'	1
002	'before'	2018-01-01	2019-01-01	'registration to appointment'	'total_benefit'	80
001	'after'	2012-04-01	2019-12-31	'fast appointment'	'num_accidents'	2
001	'after'	2012-04-01	2019-12-31	'fast appointment'	'total_benefit'	1000
002	'after'	2019-01-01	2019-12-31	'delayed appointment'	'num_accidents'	1
002	'after'	2019-01-01	2019-12-31	'delayed appointment'	'total_benefit'	300

Note that the top row of Table 10 would not be produced by the tool as there are no accidents in the before period for identity 001. We show it here for clarity.

Table 11: Example rectangular output table: [IDI_Sandpit].[project-code].[project_population_ready]

Identity _column	label_identity	summary_period _start_date	summary_period _end_date	label_summary_period	num_accidents	total_benefit
001	'before'	2012-03-01	2012-04-01	'registration to appointment'	0	50
002	'before'	2018-01-01	2019-01-01	'registration to appointment'	1	80
001	'after'	2012-04-01	2019-12-31	'fast appointment'	2	1000
002	'after'	2019-01-01	2019-12-31	'delayed appointment'	1	300

A similar, but more extensive, worked example is run as part of the automated tests for checking the performance of the tool. It can be found in the testing folder distributed with the tool.