https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions
Open Access
EGU

Inishell 2.0: Semantically driven automatic GUI generation for scientific models

Mathias Bavay[1], Michael Reisecker[1,3], Thomas Egger[2], and Daniela Korhammer[1]

[1]WSL Institute for Snow and Avalanche Research SLF, Flüelastrasse 11, CH-7260 Davos Dorf, Switzerland
[2]Egger Consulting GmbH, Hohenstaufengasse 7, 1010 Wien, Austria
[3]Alpine Software Michael Reisecker, Schiliftstraße 504, 5753 Saalbach, Austria

Correspondence: M. Bavay (bavay@slf.ch)

Abstract. As numerical model developers, we have experienced first hand how most users struggle with the configuration of the models, leading to numerous support requests. Such issues are usually mitigated by offering a Graphical User Interface (GUI) that flattens the learning curve. This requires however a significant investment for the model developer as well as a specific skill set. Moreover, this does not fit with the daily duties of model developers. As a consequence, when a GUI has been created – usually within a specific project and often relying on an intern – the maintenance either constitutes a major burden or is not performed. This also tends to limit the evolution of the numerical models themselves, since the model developers try to avoid having to change the GUI.

In this paper we describe an approach based on an XML description of the required numerical model configuration elements and a C++/Qt tool (Inishell) that creates a GUI based on this description on the fly. This makes maintenance of the GUI very simple and enables users to easily get an up-to-date GUI for configuring the numerical model. The first version of this tool was written almost ten years ago and showed that the concept works very well for our own surface processes models. A full rewrite offering a more modern interface and extended capabilities is presented in this paper.

1 Introduction

1.1 Context

When using numerical models, one of the major issues for new users is the configuration of the model. Often the numerical models are configured by ways of multiple configuration files filled with obscure configuration parameters, making a steep learning curve. Moreover, users tend to overlook even the best written documentation (Mendoza and Novick, 2005; Ceaparu et al., 2004) and resort to copying and tweaking example files. This is not satisfactory as it leads to under-performing simulations as well as very large numbers of questions directed to the model developers. The usual solution is to implement a graphical user interface (GUI) for configuring the models that also allows to predominantly show the most common settings

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020


Geoscientific Model Development Discussions

while keeping expert settings available at deeper levels. Unfortunately, this task hardly fits the job description of the modelers and is very time consuming because of the potentially large number of configuration parameters: for example the Snowpack model (Lehning et al., 2002) and its pre-processor MeteoIO (Bavay and Egger, 2014) – two of the models Inishell was originally developed for – define more than 350 configuration keys. Developing a traditional GUI for such models, where each input widget is manually laid out, would require a significant investment. As numerical models might evolve quite fast, new configuration options would frequently be added that would also require a rework of the GUI. This is hardly sustainable and leads to either out-of-date GUIs or no GUIs at all.

Moreover, the choice of tools to develop GUIs for numerical models is less than satisfying in the long term. One possibility consists of using a Rapid Application Development environment (RAD, Spreitzhofer et al. (2004)). This is easy and can appropriately be assigned to an intern or a short term student. However this is risky in the long term since such RAD implementations are often proprietary and therefore dependent on the goodwill of its editor to maintain compatibility or even to keep the product running, potentially forcing the model developers to perform a full rewrite of the GUI. Another possibility consists of using standard toolkits and languages to develop such an interface. This requires more investment and expertise from the developer but increases the long term availability of the product. However, maintaining the product also requires some expertise that is usually not found in model developers. At the very least, it adds a considerable workload which may have to be put off until later. This practically means that upgrades (such as introducing new configuration options) will only happen when another intern or student with the proper skill set can be found and funded. Delegating this task to a temporary employee however loses first-hand knowledge about the new options. Then the graphical user interface becomes a hindrance for the model itself since it prevents the fast deployment of new configuration options.

1.2 Requirements

Ideally model developers would like to have at their disposal a user friendly graphical interface for configuring the model that requires very little initial investment and expertise and where new configuration options are quickly deployed. This configuration interface should provide explanations of every configuration parameter, validate the user input (to avoid possible misconfigurations), easily integrate new options and output the complete configuration in a standard configuration file format.

Keeping the concept of a configuration file is important since such models are often configured on one system and then sent to run on some clusters to perform the heavy duty computing. Moreover, it should be manually editable in order to copy/paste some of it between similar simulations (keeping in mind that several hundred lines might be copied that represents several hours of carefully choosing the options) and to be able to modify them with text terminals through remote sessions (as is typically the case when running on a computing cluster). In order to further improve the quality of the numerical modeling work and constrain the problem, numerical model developers are strongly encouraged to rely on a single configuration file for the whole model, including as much of the pre- and post- processing as possible. This has the advantage that a copy of the said configuration file is then a reproducible description of the numerical simulation that has been performed (Bavay et al., 2020a). It represents a fixed state and keeping a changelog of the configuration files enables investigations into past simulations.

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020


Geoscientific Model Development Discussions

As a side effect, having an easy to maintain GUI at their disposal encourages model developers to document new features and even to avoid hard-coded values since making them a dynamic setting read from a user-editable configuration file is easy and quickly done.

2 Inishell overview

The Inishell Open Source software is our technical answer to the previously laid out requirements. The original version of Inishell was written in the Java programming language in 2011. However, this original version required major code restructuring in order to update some internal design decisions that did not work that well in the long term and in order to expand the feature set of the application. Moreover, as the Java environment is often not installed by default on personal computers anymore, it has started to cause more support requests related to the installation of Java as well as its configuration. Finally, the original version of Inishell missed the possibility to run the numerical models directly from within its own interface and this has been identified as a major hindrance towards having more users rely on Inishell instead of manually configuring their simulations. Thus it was decided to fully reimplement and expand Inishell in C++ with the Qt library[1] as a way to provide a cross-platform GUI that can be reused for multiple numerical models and that is sustainable over many years. It aims to feel familiar to the end users while considerably lowering the required skill set and time investment for the model developers and also shifting support requests away from IT tasks to work directly concerning the models. It remains Open Source under a GPLv3 license and works on Linux, Microsoft Windows and Apple macOS among others.

2.1 Principles

First, as it would not be feasible to support all possible configuration file syntax choices, a reasonable standard must be enforced. The INI[2] informal standard has been chosen as it is a text format that is very easy to read and parse with various programming languages. Its syntax is also supported by many text editors, making manual edition convenient on multiple platforms.

Then, based on the requirements laid out in section 1.2, the core idea is that the model developer should not spend time organizing the layout of the GUI but only provide the Inishell GUI generator with enough information to generate the right kind of interface for the end user on the fly. This GUI is then presented to the end user and is responsible for the validation of user input and the writing of a configuration file that the numerical model can rely upon to run (Fig. 1). The servicing of existing GUIs and the creation of new GUIs is therefore decoupled from the release cycle of Inishell itself.

The information that is given to the GUI generator is formatted as an XML file containing for each INI configuration key the key name itself, the data type of the value that should be provided by the end user as well as a help text. Input validation is provided by enforcing the expected data type through the use of an adequate widget, by enforcing optional range checking, and by validating the input with a regular expression. This is similar to the input validation provided by common JavaScript libraries

[1] https://www.qt.io/
[2] https://en.wikipedia.org/wiki/INI_file

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions

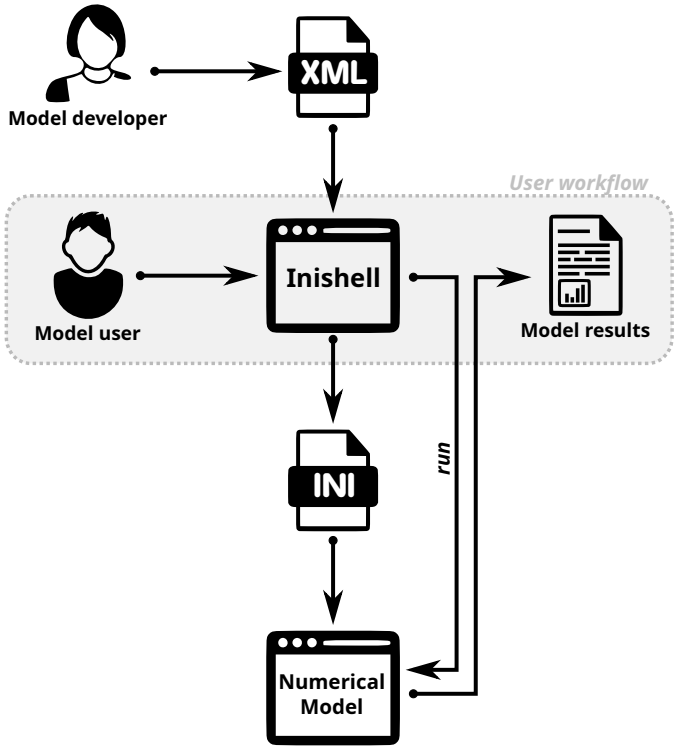

Figure 1. Inishell principle of operations

such as Angular[3] or React[4]: data type, min, max, required or not, pattern. This means that the model developer does not look into low level interface attributes (exact positioning, complex layout) but only provides a high level semantic description of the configuration parameters. Laying out all the widgets that have been generated according to the XML file is handled internally with the help of a layout manager. This is then a higher level view of the GUI than in previous efforts such as XUL (XML User Interface Language, Goodger et al. (2001)) that is still focused on low level widgets or even UIML (User Interface Markup Language, Abrams et al. (1999)) that still kept low level widgets as basic building blocks. It can be best compared to Atomic Design (Frost, 2016) since it is also built around a hierarchical point of view but keeping in mind that here the focus is not the entry widget type but data semantics of the data that has to be retrieved from the user.

2.2 Overview of the general interface

The overall interface (Fig. 2) is made of three areas, as well as a standard top menu bar. Area 1 controls the whole user workflow: In the upper panel (Applications), the user selects which model he or she wants to configure. Below (Simulations), it is possible to open a preconfigured pair of numerical model profile and specific configuration file (i. e. a model with loaded

[3]https://angular.io/api/forms/Validators
[4]https://react-hook-form.com

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions

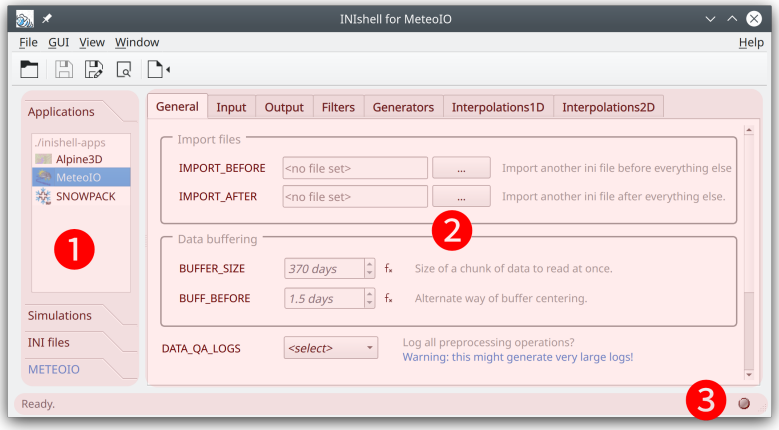

Figure 2. Overview of the Inishell software with the MeteoIO numerical model profile loaded (here on Linux).

settings ready to run). One drawer lower (INI files), it is possible to open an existing configuration file. Finally, for numerical models that support it, the lowest drawer (METEOIO, shown in blue) enables running the simulation and potentially opening the simulated results.

Area 2 in Fig. 2 contains all configuration widgets for the selected numerical model profile. This is where the end user fills a configuration file. Area 3 is a status bar that shows error messages or warnings (such as for missing mandatory configuration keys) or the status of a currently running simulation. Clicking on status messages will open a more extensive log window.

In order to further encourage end users to rely on Inishell to configure and run their simulations, a text editor supporting various configuration file features is integrated within Inishell under the name *Preview Editor* in the menu. Besides syntax highlighting, it can sort configuration keys, perform various kind of case or white space conversions, comment or uncomment whole selections, add all missing mandatory keys or a copyright header, and keep snapshots of the file throughout the editing process. Hence, the Preview Editor incorporates several Inishell features into a text editor like every user will have seen and used while still minimizing classical user errors (such as by marking unrecognized or deprecated keys as unknown).

Rudimentary command line options are available; a system with Inishell installed can use it as a command line agent to modify configuration keys within an automated simulation toolchain. Inishell has multi language capabilities; so far English and German are shipped. It is documented extensively both for end users and model developers.

3 Implementation

3.1 Supported INI file syntax

Although best practices have emerged that make the INI informal standard reasonably usable as a configuration file syntax, it is too loosely specified to be easily automatically generated and therefore has been defined more strictly for this work as well as extended to better suit the needs of numerical models.

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions

```
[General]                          ; entering the "General" section
#meteo data input settings           this whole line is a comment
BUFF_CHUNK_SIZE = 370              ; this is an inline comment
METEO           = SMET             ; value as string
METEOPATH       = ./input/meteo    ; a path is also a string
STATION1        = FLU2             ; providing two station IDs
STATION2        = FIR2

[Filters]                          ; entering the "Filters" section
TA::filter1     = min_max          ; namespace for key "filter1" is "TA"
TA::arg1        = 240 320

RH::filter1     = min_max          ; another "filter1" key, but in namespace "RH"
RH::arg1        = 0.01 1.2
```

Figure 3. Syntax of the INI file

The general principle consists of a list of key/value pairs, delimited by an '=' sign. The values can be of type doubles, integers, boolean (*true*/*false* or 0/1) or strings. It is possible to add comments: all characters following '#' or ';' will be considered to be comments until the end of the line is reached. The keys can be grouped by sections in order to bring more clarity and structure to the configuration file, each section being marked by a section name between square brackets. Spaces and tabs can be used freely between words (either keys or values). Each key must appear only once per section but the same key can appear in several sections: for example a time zone information can appear in an input and an output section.

In order to keep the uniqueness of the keys in each section while allowing semantically identical keys to coexist, several extensions have been defined. A first possibility is to simply add a number after the key, making it in effect unique but clearly showing the user that all these keys participate to the same concept. Another possibility is that a key may receive several values by providing the different values space-delimited after the equals sign. Finally, a weak concept of namespaces has been introduced: a key can be prefixed by a namespace so multiple keys belonging to different namespaces can coexist in the same section. This makes it possible for example to declare keys for specific meteorological parameters by using the meteorological parameter abbreviation as namespace.

A commented example of the syntax described above is given in Fig. 3.

3.2 General architecture

The hierarchical approach to interface design is seen both in the GUI itself and in the underlying architecture (where Inishell mirrors the XML structure) and defines the roles of each contributor to the GUI for any particular numerical model. The atomic elements (*atoms* in Atomic Design) are the widgets provided by the Qt toolkit. These are never exposed to the model developer, instead they are grouped into higher level elements (*molecules* in Atomic Design) by Inishell for each parameter type in the

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020


Geoscientific Model Development Discussions

XML file. In effect, by writing a succession of parameters belonging to sections in the XML file, the model developer sets up all parameters necessary for the configuration of a module of his or her model, distributed over one or more tabs in the GUI that act as the next hierarchical level and are mapped to sections in the resulting INI file. These will then be grouped together under an application name that might also receive a workflow (step-by-step instructions to configure and run some model) and an icon. This is the highest hierarchical level as it matches a specific numerical model.

Atomic design	Qt	Inishell	Model developer
atom	widgets (textfield, combobox...)		
molecule		basic building block	
module	layout manager	create and populate the tabs	declare the parameters
application		multiple tabs, workflow & INI editing / writing	gather all parameters, describe the workflow

Table 1. Role distribution for Inishell

This hierarchical approach is simplified by relying on two modularity constructs: parameter groups and includes. Parameter groups allow giving an internal name to any group of parameters. This internal name can then be referred to later on to call this group one or multiple times. This is even more meaningful when used with the built-in inclusion system: a full file will be included but it is possible to only select a subset of the file thanks to parameter groups. Several applications sharing most of the same configuration keys for any subset of their configuration can then include one file that defines all possibilities and only call the parameter groups that are relevant. In the same way models that rely on other models (e. g. in the form of libraries) can simply include this lower level model and freely extend upon it.

3.3 Basic building blocks

Inishell supports the following data types: strings, dates and times, paths to files and paths, decimal numbers, integral numbers and booleans, usually with several display options. Strings are less strictly defined as this type can accommodate free text entry or a selection among a preset list of choices (that can potentially be extended by the end user). Geographic coordinates are matched within strings through a regular expression that triggers the generation of an additional button that shows the provided coordinates on an online map. Strings can be validated by means of a regular expression, as well as through an expression parser to make them suitable for mathematical formulas.

For each data type, Inishell generates a low level entry widget prefixed with a label that shows the matching INI configuration key (or another, better suited label chosen by the model developer) and followed by a help text (that may also contain hyperlinks to a more exhaustive online documentation). Hence, Inishell manages several abstraction layers for the programmer

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions

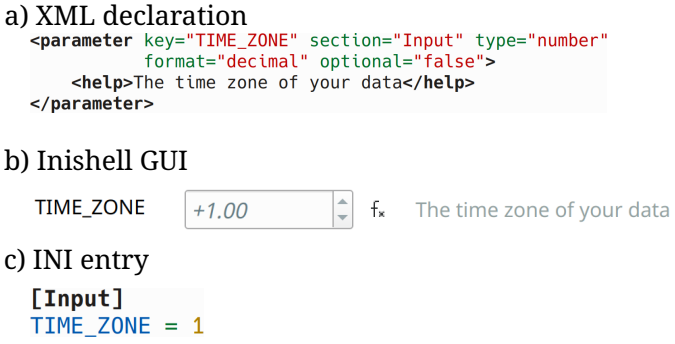

Figure 4. Basic building block: integer entry

and adequately adding and describing a model setting in the right place is now as easy as adding an XML text node without the need to recompile any software. Several properties for each INI configuration key can be declared. Among those, the XML property *optional* when set to *false* makes the matching widget appear in red and display a warning message when saving the file without setting it. In such a case, all the mandatory keys that have not been set by the end user will be listed in a log window. These error messages don't prevent saving the file, though. Manual styling of the text displayed in the various widgets is possible, such as font family, size, weight or style. Colors can be chosen freely with an RGB hexadecimal representation, but Inishell also offers a set of predefined colors with semantic names (such as *warning*, *info* . . .) which have been designed to keep good visibility if the end user changes the GUI theme, for example when using the dark theme or system wide accessibility settings.

3.4 Grouping elements

The first grouping element is matched to an INI structure: sections. It is either expressly declared in the XML or indirectly as the basic building blocks can declare which section they belong to. In the GUI, this is represented by a tab, so all INI keys belonging to a given section will have their matching widgets appear in the same tab. The end user has an overview of all the sections with the list of tabs on the top of Inishell (Fig. 2, on top of area 2).

Another grouping element is available that does not match any INI structure: frames. A frame is used to graphically group basic elements that belong together, for example a set of configuration parameters all related to the same concept in the numerical model. A frame can have its own help text which can be convenient to describe in details the feature that is configured by the keys within the frame.

3.5 Templates

Some fragments of the INI configuration file might have to be repeated multiple times, for example to iterate over multiple input files or over meteorological parameters. In this case, a base key is defined (for example "STATION") and multiple versions derived from this base key will be generated on-demand as requested by the user (for example by clicking on a "+" button

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions
Open Access
EGU

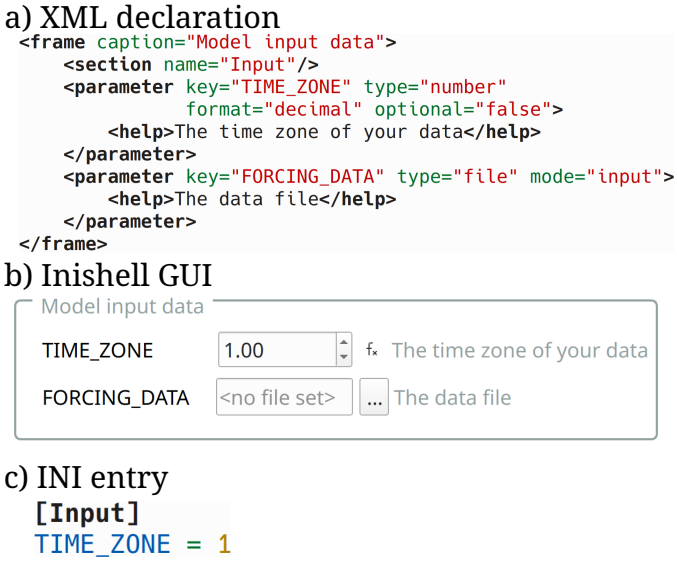

Figure 5. Visually grouping elements together

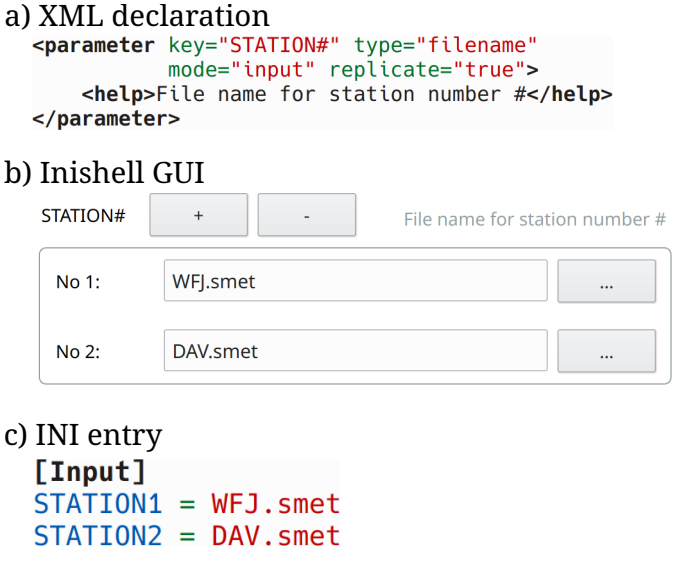

Figure 6. Simple example of templates

to generate "STATION1", "STATION2" ...). This lets the end user provide as many variants as necessary without having to hard-code the configuration keys for each variant. In the XML file, it is handled with a system of templates where the iterators are defined first (for example, as integral numbers or as a fixed list of strings) followed by the group of configuration keys containing a wildcard character. Inishell will then dynamically generate as many entry widgets (or groups) as asked by the end user and dynamically generate the resulting INI keys.

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions
Open Access
EGU

3.6 Nested widgets

a) XML declaration

```
<parameter key="METEO" label="Select METEO data plugin:"
           type="alternative" optional="false">
    <section name="Input"/>
    <option value="SMET">
        <parameter key="METEOPATH" type="path" optional="false" />
        <parameter key="METEOPATH_RECURSIVE" type="alternative">
            <option value="TRUE"/>
            <option value="FALSE"/>
        </parameter>
        <parameter key="STATION#" type="filename"
                   mode="input" replicate="true" />
        <parameter key="SNOWPACK_SLOPES" type="alternative">
            <option value="TRUE"/>
            <option value="FALSE"/>
        </parameter>
    </option>
</parameter>
```

b) Inishell GUI before combobox selection

c) Inishell GUI after combobox selection

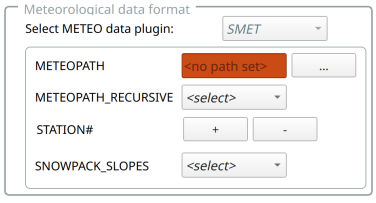

Figure 7. Example of nested widgets (for clarity, the help texts have been removed). Please note that the METEO key uses an alternate label and is defined as mandatory. Once it is selected as SMET, more widgets appear including the METEOPATH that is then also mandatory.

Some dedicated widgets offer the possibility to include more configuration options that will be shown only when a certain choice is selected by the user. This allows offering more configuration options related to a given submodule if the said submodule has been enabled (for example, ticking a checkbox could show further options of the same INI section and so could the selection of specific list entries). This is a recursive process and allows for indefinite nesting.

3.7 Workflows

In order to allow the end user to run the numerical model from within Inishell, it is possible to declare the necessary workflow in the XML file. This includes command line programs as well as their command line options (based on the data types that are provided by the end user), directory views (for example to open the model results directory) or opening URLs (for example to open an online viewer). The terminal outputs of the applications started by Inishell are captured and shown in Inishell's main window with some basic syntax highlighting in order to highlight error messages or warnings.

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions

a) XML declaration

```
<workflow>
    <section caption="DEMO">
        <element type="label" caption="Start date:"/>
        <element id="start_date" type="datetime"/>
        <element type="label" caption="INI file:"/>
        <element id="ini" type="text" default="${inifile}"/>
        <element caption="Run DEMO" type="button">
            <command>my_demo -c %ini -b %start_date</command>
        </element>
        <element type="label" caption="Visualize results:"/>
        <element id="visualize" caption="Open niViz" type="button">
            <command>setpath(%outpath, ${key:OUTPUT::PATH})</command>
            <command>openurl(https://run.niviz.org)</command>
        </element>
        <element type="label" caption="Then drag your desired
                        output file into niViz from below:"/>
        <element type="path" id="smetpath"/>
    </section>
</workflow>
```

b) Inishell Workflow

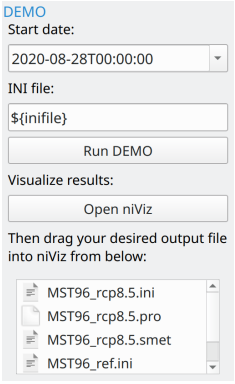

Figure 8. Example of a workflow: a few command line parameters must be provided by the user who can then run the numerical model and open a visualization application.

3.8 Applications

Since multiple numerical models can be loaded into Inishell by opening their respective XML files, it is necessary to visually show which choices of models are available and easily change between them. This is achieved by providing an XML file that defines the application properties as well as the previously described XML elements for the configuration widgets and potentially a workflow. For added modularity it is highly recommended to rely on the include mechanism (with the *include* XML element) so XML files are dynamically included and only enable specific parameter groups as required. An application will therefore consist of a name and an icon in the applications panel, several tabs with configuration options in the main panel and often a workflow to run the application. The title of the Inishell windows is also adjusted to reflect which application is currently loaded.

Combining the features listed here (choosing an application from a list, optionally auto-loading an INI file and a coupled workflow), developers can set up a list of all their models' workflows and simulations. Inishell can then handle everything from configuring the model, running it and performing maintenance work (by executing user defined system commands) – all with the click of a button within one uniform GUI and without any programming necessary.

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020


Geoscientific Model Development Discussions

4 Conclusions

The approach outlined in this work has worked very well for many years with the original version of Inishell written in the Java programming language in 2011, allowing multiple numerical models to evolve freely without worrying about tedious redesigns of the GUI: most of the additionally introduced configuration keys could be declared in Inishell in a matter of minutes. The possibilities offered by the XML elements recognized by Inishell have been mostly adequate. Support requests by end users have dramatically dropped since Inishell was launched.

The new Inishell has successfully expanded the descriptive capabilities of the XML elements to fully cover the needs that arose in the last nine years and to offer a fully self sufficient environment for configuring and running the numerical models that rely on it. This means that end users don't need to work through a combination of tools that tended to encourage them to manually tweak the configuration files (and therefore introduce errors) but find everything they need in one integrated package.

The new version has since been used in complex operational simulation toolchains with completely different numerical models than it was originally developed for. Merely by adhering to the INI syntax it was possible to adequately set up the models' parameters through Inishell, document them and to offer an easy to use and familiar GUI to the people running the models.

Code availability. The current version of Inishell is available from the project forge https://models.slf.ch/p/inishell-ng/ under the GNU General Public License v3.0 (GPL v3) licence. The exact version of Inishell presented in this paper is archived on envidat.ch (Bavay et al., 2020b).

Author contributions. M. Bavay lead the project from the beginning, contributed maintenance and development on all versions. He also wrote the bulk of the paper. M. Reisecker provided the bulk of the development of the new Inishell as well as maintenance since then and also contributed to the paper. T. Egger assisted with the implementation of the first version, helped maintain it over many years and contributed to the paper. D. Korhammer co-designed and implemented most of the original Inishell.

Competing interests. The authors declare that they have no conflict of interest

Acknowledgements. The authors are very thankful for the continued support of Charles Fierz and Michael Lehning who trusted us with our vision.

https://doi.org/10.5194/gmd-2020-339
Preprint. Discussion started: 11 December 2020

Geoscientific Model Development Discussions
Open Access
EGU

References

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E.: UIML: an appliance-independent XML user interface language, Computer networks, 31, 1695–1708, 1999.

Bavay, M. and Egger, T.: MeteoIO 2.4. 2: a preprocessing library for meteorological data, Geoscientific Model Development, 7, 3135–3151, 2014.

Bavay, M., Fiddes, J., and Østein Godøy: Automatic Data Standardization for the Global Cryosphere Watch Data Portal, Data Science Journal, 19, https://doi.org/10.5334/dsj-2020-006, 2020a.

Bavay, M., Reisecker, M., Egger, T., and Korhammer, D.: Inishell-2.0.4, https://doi.org/http://dx.doi.org/10.16904/envidat.194, https://www.envidat.ch/dataset/inishell-2-0-4, 2020b.

Ceaparu, I., Lazar, J., Bessiere, K., Robinson, J., and Shneiderman, B.: Determining causes and severity of end-user frustration, International journal of human-computer interaction, 17, 333–356, 2004.

Frost, B.: Atomic design, Brad Frost Pittsburgh, https://atomicdesign.bradfrost.com/, 2016.

Goodger, B., Hickson, I., Hyatt, D., and Waterson, C.: XML User Interface Language (XUL) 1.0, Tech. rep., Mozilla.org, https://www-archive.mozilla.org/projects/xul/xul, 2001.

Lehning, M., Bartelt, P., Brown, B., Fierz, C., and Satyawali, P.: A physical SNOWPACK model for the Swiss avalanche warning: Part II. Snow microstructure, Cold regions science and technology, 35, 147–167, 2002.

Mendoza, V. and Novick, D. G.: Usability over time, in: Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information, pp. 151–158, 2005.

Spreitzhofer, G., Fierz, C., and Lehning, M.: SN_GUI: a graphical user interface for snowpack modeling, Computers & geosciences, 30, 809–816, 2004.