Geosci. Model Dev. Discuss., 1, S133–S137, 2009
www.geosci-model-dev-discuss.net/1/S133/2009/
© Author(s) 2009. This work is distributed under
the Creative Commons Attribute 3.0 License.

Geoscientific
Model Development
Discussions

Interactive comment on "Spud 1.0: generalising and automating the user interfaces of scientific computer models" *by* D. A. Ham et al.

D. A. Ham et al.

Received and published: 19 January 2009

We would like to thank the reviewer for his constructive comments on our paper which have enabled us to produced a revised version which we feel is significantly improved.

General Comments

> What I would like to have explained in a clearer way is how the system – including all files and program components starting from definition of the grammar file over Diamond up to the simulation code interacting with libspud – is built up. Preferably this could be done in a graphical way.

We have added a graphical flowchart showing the data dependencies in a model using the Spud system to the revised paper. Unfortunately, the limitations on interactive

Full Screen / Esc
Printer-friendly Version
Interactive Discussion
Discussion Paper

comments do not allow it to be reproduced here.

> Especially the part of the interaction with the (most likely already existing) scientific code for me in my opinion is not discussed in a sufficient way.

We have significantly expanded this section, in particular including the following text:

> At this stage it is important to re-emphasise the distinction between options and data which was raised in Sect. 2.2. The role of the libspud in-memory tree is to provide access to parameters specified in the options file such as the time step, choice of numerical scheme and the location files containing bulk input data. The storage of bulk data (such as solution fields and matrices) itself is not handled by Spud but is left to the existing mechanisms in the model. Indeed, it will generally be necessary to locally store options from the libspud tree in the local routines in which they are used. The advantage in this respect of the Spud system is that it removes the need to modify the options reading code and to pass the option in question through the call tree to the point at which it is used. Figure (omitted) illustrates the difference in the code changes which are needed to introduce a new model option. From this it can be seen that the addition of a new option is accomplished by adding its specification to the schema file and then accessing the option directly from the routine which implements the new feature. No additional values need to be propagated through the code either by function calls (as illustrated) or alternatively by some form of global, module or common variable arrangement.

Once again we note that it is not possible to reproduce in an interactive comment, the figure to which we refer above.

GMDD

1, S133–S137, 2009

The reviewer is correct in understanding that it is necessary to link against libspud and call libspud API functions in order to access the options from the file. However, this is distinct from the question of how the model stores its data. So, for example, the user may specify the numerical scheme to be employed in the options file, and libspud makes this information available for the program to access. However, implementing the numerical scheme and storing the resulting data is not the problem which Spud exists to solve.

With respect to the ease of adding options from the model development end, there are a number of steps in adding a new option to a model: defining it in the file format, reading the option in to the model, passing the option through the model to the point at which it will be used and using the option in the code. Spud makes the first of these easy and the second and third automatic, which is the sense in which it decreases the development cost of adding new options. The use of the option is a matter for the model itself. The missing link here, of course, is how the option is transfered from the libspud in-memory representation to a local variable at the point at which the model wishes to access the value of the variable. To address this question, we have added the following section:

> From the developer's perspective, options are accessed as needed by referring to their location in the options tree. For example, suppose that the schema fragment shown in Fig. 1 occurs at the top level of the options tree. Then the model developer can retrieve the value of the model time step with the Fortran call:
>
> ```
> call get_option('/timestepping/timestep',dt)
> ```
>
> where *dt* is a double precision real variable. In the schema used to drive the diamond interface in Fig. 2 there is an optional parameter which enables adaptive time stepping. Clearly the relevant routine in the model

GMDD

1, S133–S137, 2009

Interactive Comment

will need to test for the presence of this parameter. The following function call returns true if the parameter is present and false otherwise:

```
have_option('/timestepping/adaptive_timestep')
```

A more comprehensive example of the use of libspud in model code is presented in *ballistics.F90* in the *examples* directory in the accompanying source code while a full description of the entire libspud interface in Fortran, C and C++ is to be found in the manual (*doc/manual.pdf* in the source directory).

Copyright We acknowledge that the copyright information was in a very un-obvious place (it was in the Debian directory and is copied to the correct locations by Debian and Ubuntu package installs). The Copyright information is now included in the revised paper as well as in COPYING files in the main spud and diamond directories. For the record, Diamond is distributed under the GPL V3.0 and the remainder of Spud is distributed under the LGPL V2.1.

Abstract The *"generic option reading module"* is indeed libspud and the revised abstract makes this explicit.

Terminology for options files The existing mix of terminology is confusing. We have standardised on "options file" throughout the revised paper.

Comments and documentation We agree that schema annotations presented in a GUI are not a complete replacement for other forms of documentation and have therefore added the following qualification:

Relatively short comments associated with individual input parameters are clearly not adequate as a sole source of documentation, however their presence significantly adds to the information readily available to the user and therefore contributes materially to the usability of the model in question.

GMDD

1, S133–S137, 2009

Interactive Comment

Applicability to other models

We hope that the additional paragraphs added above, as well as the flow chart, make the context of this statement clear. Spud is an options handling mechanism and the only part of the options handling mechanism which is model-dependent is the schema. Of course the code has to do something with those options and that requires work by the model developer but that is not really a part of the options handling system.

Figures

Figure 3 was indeed essentially a zoom in on figure 2 which was included to attempt to overcome the technological issues of screenshots. We have now produced a much higher resolution screen shot for figure 2 and removed figure 3. The new figure is very readable in print and upon zooming in to the pdf. It is improved but unfortunately still less than perfect when viewed in the pdf at one page per screen size but this is difficult to avoid.

Supplement

As requested, we have added a small example called “ballistics” to the examples directory of the source distribution. This includes a short Fortran program and schemas in compact and XML formats. Even though it’s very simple, many of the features of Spud are utilised, including a number of different data types and an element (projectiles) of which an unlimited quantity are permitted.

Interactive comment on Geosci. Model Dev. Discuss., 1, 125, 2008.