



**::INSEL**

**Programmer's  
Guide**



## INSEL 8 :: Programmer's Guide



© 1986–2019 Jürgen Schumacher. All rights reserved.

Please visit us at [www.insel.eu](http://www.insel.eu)

Bison is free software and is available under the GNU General Public License.

Eclipse is a trademark of the Eclipse Foundation, Inc. and copyright protected by Eclipse contributors and others.

GCC is copyright protected by Free Software Foundation, Inc.

Flexx is copyright protected by The Regents of the University of California and The Flex Project.

gfortran is copyright protected by Free Software Foundation, Inc.

gnuplot Copyright 1986 – 1993, 1998, 2004 Thomas Williams, Colin Kelley

HP VEE is a registered trademark of Hewlett-Packard Co.

IISiBat is copyright protected (Centre Scientifique et Technique du Bâtiment CSTB).

INSEL is a registered trademark of doppelintegral GmbH, Stuttgart, Germany.

Java is copyright protected by Sun Microsystems.

LabVIEW is a registered trademark of National Instruments Corporation.

Linux is a registered trademark of Linus Torvalds.

Mac OS X is a registered trademark by Apple, Inc. in the United States and other countries.

MATLAB is a registered trademark of The MathWorks Inc.

Microsoft Windows and Visual Studio is a registered trademark of Microsoft Corporation in the United States and other countries.

MiKTeX-pdfTeX is copyright protected by D. E. Knuth and Han The Thanh

Ruby is copyright protected free software by Yukihiro Matsumoto

Simulink is a registered trademark of The MathWorks Inc.

Subversion is an open source version control system.

TeX is a trademark of the American Mathematical Society.

TRNSYS is copyright protected (S.A. Klein, F.L Alvarado).

VSEit is copyright protected by Kai Brassel.

Window Builder is provided under the terms and conditions of the Eclipse Foundation Software User Agreement.

## 0.1 INSEL Git Repositories

**Some preliminary remarks** The last release of INSEL was version 8.2.1.833, published in August 2014 as a product of the company doppelintegral GmbH (liquidated in December 2019). INSEL 8.2.1.833 was available for Windows (32 bit and 64 bit), macOS (64 bit), and Linux (Debian and RedHat, 32 and 64 bit).

The plan is to release INSEL 8.3 (as soon as possible, i. e., early 2020) as freeware of Concordia University, Montréal. The perspective is, to release INSEL 9+ as open-source software (timeline, maybe early 2021).

Concerning contents, INSEL 8.3 will be the last release (with long-term support) for 32 bit and 64 bit operating systems, based on VSEit (Versatile Simulation Environment for the Internet, pronounced as “use it”) as UI (User Interface).

Plans for INSEL 9 are, to replace VSEit by a diagram editor based on Eclipse technologies like EMF (Eclipse Modeling Framework) and Sirius. There will be no more support for 32 bit operating systems, and all INSEL block formal parameters will be replaced by 64 bit variables, i. e., 64 bit integers, 64 bit doubles, etc. As a consequence, INSEL 9 blocks will not be downward compatible, so that, in consequence, INSEL 8 blocks cannot be used in INSEL 9 anymore.

The second point concerning INSEL 9 will be an extension towards undirected graphs. The concepts are already in place, and—very surprising—they do not require any new INSEL syntax, but just a new block group (group 0 for undirected blocks). The challenge is to extend the INSEL Engine so that undirected graphs can be “solved”. This is work in progress.

Concerning plans for INSEL 10, the idea is to extend the block concept towards the exchange of “Containers” between blocks. So far (up to INSEL 9), only floating numbers can be exchanged between INSEL blocks as inputs and outputs. Typical examples for “Containers” could be vectors, polygons, buildings, or other very “general objects”. The favorite idea (at the moment) is, that a “Container” might be a pointer to a C++ object or instance of a C++ (or Java) class, for example.

Then there is “the idea” of INSEL 4D. First of all, INSEL 4D is not INSEL 10. INSEL 10 is a simulation environment with a diagram editor user interface (based on Sirius, most probably). INSEL 4D is supposed to be an “Urban Modeling Platform” which integrates INSEL 10, Sirius, and third-party tools—which provide APIs (application programming interfaces). The design of INSEL 4D is subject to discussions, mainly between Kai, Guillermo, and Jürgen for the time being.

Two “mainstream” ideas are discussed at the moment (March 2020): (1) Kai proposes to use Eclipse technology and Java as “kernel”, (2) Jürgen prefers INSEL 10 and Sirius as the “basis”, and use the INSEL Engine as “control center”. A decision is not yet made, any contribution of ideas and proposals is highly welcome.

### Repository structure at Concordia

INSEL 8 was developed under the version control system Subversion. The decision for the further development of INSEL 8.3+ was to use Git and to host the repositories at Concordia—and not at GitHub, or any other commercial service. The structure of the existing repositories is experimental, the aim is to gain “CERC-intern” experience with the administration of Git repositories, aiming at, finally, disclosing the repositories to the open-source community. But, for the time being, the idea is to restrict access to the INSEL repositories only to “Concordia contributors” (including Kai, and Eric, and other friends from good-old Europe).

### Access

So, if you are interested, how can you get access to the INSEL repositories during the, let’s say, test phase? As mentioned earlier, the INSEL repositories are hosted at Concordia University in Montréal, Canada, by AITS (Academic Information Technology Services). In general, AITS provides academic computing, information services and applications to support teaching, administrative and research activities of the Gina Cody School of Engineering and Computer Science.

Access rights are controlled via SSH (Secure Shell) keys. Hence, if you do not yet have a personal SSH key, the first thing to do is to generate a pair of SSH keys which uniquely identify you. SSH keys are not machine dependent, but are something comparable to your personal passport. This means, you can use your SSH keys on any computer from which you would like to access the INSEL repositories.

SSH keys have two “components” (files), i. e., a private key and a public key. Your private key should NEVER be shared with anybody, your public key—as the name says—is public. Depending on the operating system you are using, there are different locations/folders where your SSH keys should reside. Under macOS it is a hidden folder named `.ssh` in your personal home directory, Windows expects your SSH key in [please add](#), Linux uses [please add](#).

### Create SSH key

There are several ways how you can create your SSH keys. Most operating systems support the `ssh-keygen` tool. For example, on macOS you can simply type

```
ssh-keygen -t rsa
```

You will be prompted to specify a location for the key. By pressing the enter key you accept the default location (which is `~/.ssh` (on macOS), i. e., the hidden directory `.ssh` in your home directory). Next, you will be prompted for a passphrase. It is not recommended to press the enter key again as this would accept the default (no passphrase). After confirmation of your passphrase, your key pair `id_rsa` and `id_rsa.pub` will be generated.

Again, you should never share your private key `id_rsa` with anyone, it is your private identity. Your public key `id_rsa.pub` is meant to give it to those who want to identify you by SSH key exchange.

In case of insel4D and getting access to the repositories you wish to have the right to access them, send your public key to me

juergen.schumacher@concordia.ca

and i will forward it to AITS. In case you are allowed to access the INSEL repositories as a whole or in parts you may clone them from the server to any folder on your machine by, for example,

```
git clone git-cerc@login.encs.concordia.ca:inselTools
```

For the time being, these repositories are online (in alphabetical order):

```
:: inselBlocks
:: inselData
:: inselDeployment
:: inselDoc
:: inselEngine
:: inselExamples
:: inselExperiments
:: inselTools
:: inselUserBlocks
:: inselVSEitUI
```

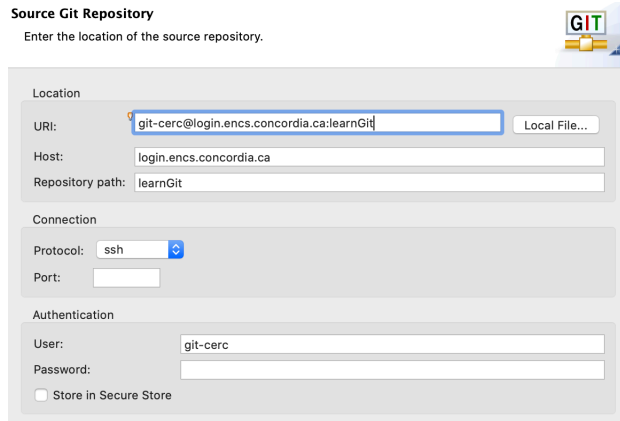
**learnGit** In addition, there is “playground repository” named learnGit which includes generic projects for all kind of programming languages. The learnGit repository is a good starting point for those who are new to Git (and Eclipse) and who want to experiment with Git and Eclipse, without the danger of destroying something important (which is almost impossible with Git, since Git traces all changes ever made in any repository and Git can revert files to earlier versions of the repository at any time).

Another option to clone a Git repository is to do it straight away in Eclipse.

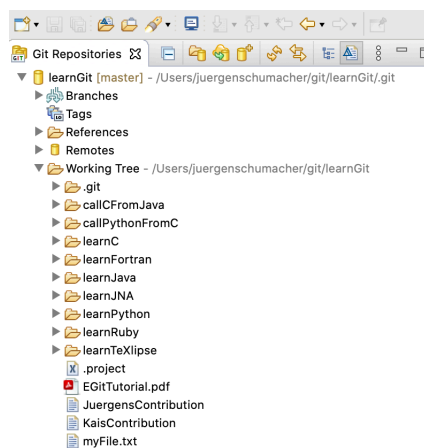
1. Start Eclipse.
2. Change to the Git Perspective: Window - Perspective - Open Perspective - Other... - Git. The Git perspective should offer a “Clone a Git repository” in the Git Repositories Explorer.
3. Eclipse displays a Source Git Repository window and asks you to enter the location of the source repository

```
git-cerc@login.encs.concordia.ca:learnGit
```

for example. The rest of the parameters should be filled in by Eclipse automatically.



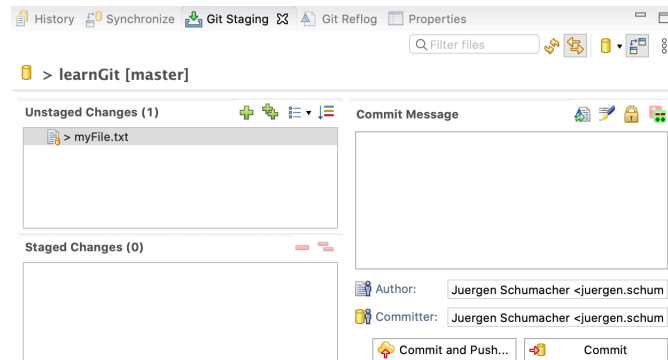
4. Clicking “Next” brings you to the “Branch Selection” window. Choose “master” and press “Next”.
5. The “Local Destination” window lets you configure the local storage location. By default, Eclipse suggest to use the git directory in your local home folder (~). It’s a matter of taste whether you follow that general habit or not, it is your personal decision, it does not affect anyone else who uses the respective repository.
6. The “Finish” button finalizes the clonig process and you should see something similar to this screenshot in Eclipse:



Once you are “in” you can edit any file in the repository and add something, like, “I have been here. My name is ...” in myFile.txt, for example.

To “commit” your changes, go to the Git staging window, the file you modified should appear in the staging list. Select it and click the plus button, write a short commit

message about your changes, and finally click on commit and push to save the changes you made in the original repository.



It's as easy as that. Just give it a try!

#### INSEL repository structure

The actual status of the INSEL repository structure is provisional and open for experience and discussions. At some point, maybe in 2021, we want to go completely open source with an appropriate repository structure. For the time being, I don't think that it's a good idea to have everything in just one repository but to split the repositories. One guideline might be, what are the "products", i. e., outputs of each repository.

#### inselBlocks

The inselBlocks repository can be considered as the "model brain" of INSEL. Everything ever programmed as INSEL blocks (from version 4 to version 8.3+) can be found in one of those libraries.

According to the current status (March 2020) of this repository there are seven products/libraries which contain topic-specific INSEL blocks. At the moment the topics are (in alphabetical order), building simulation (inselBS), energy meteorology (inselEM), fundamental blocks (inselFB), power plants (inselPP), solar electricity (inselSE), and solar thermal energy (inselST).

The output format of the libraries is operating-system dependent: .dll's for Windows (dynamic link libraries), .dylib's for macOS (dynamic libraries), and .so's for Linux (shared object libraries). A makefile is provided to build the libraries from the source folder src for all three operating systems—**still to be adapted following Guillermo's idea of having one makefile with different targets for all operating systems.**

#### inselData

The role of the inselData repository is still a bit unspecified. As the name points out, it is just a collection of INSEL-related data, like block parameter sets, everything related to the meteorological "data base" of INSEL (the mtm folder), specific weather data collections (weather folder), etc.

Whether specific 3D data files, like CityGML, geoJSON, Open Street Map, and similar should reside here is still unclear, probably the answer is no.

inselDeployment initiated by Guillermo

**inselDoc** This repository contains tons of  $\LaTeX$  files, .png graphics,  $\TeX$  picture files .pic, Ruby scripts for the partial generation of  $\TeX$  input from source code files in the inselBlocks library, etc.

The main products of this repository are the INSEL Block Reference Manual, the Simulink version of the Block Reference Manual, a generic version of the INSEL Userblock Reference Manual, the Getting Started Manual, the INSEL Programmers Guide (the one you are currently reading), and the INSEL Tutorial.

The complete documentation is in English language (so far), it would be nice to support other languages in the near future, German, French, Spanish, for example, would be “nice to have” at some point.

**inselEngine** When the inselBlocks repository is the model brain of INSEL, the inselEngine repository is the “heart” of INSEL, bringing together all the bolts and nuts, gears and wheels, and so on. The inselEngine contains a full compiler, including a scanner and a parser based on the compiler-compiler tools Flex and Bison. That means, working on the inselEngine is like open heart surgery.

The main product of this repository is the inselEngine library, plus a tiny little program named insel, which provides insel as a command-line executable. insel just calls the inselEngine with a given .insel or .vseit model, not more, not less.

**inselExamples** This repository is just a collection of examples, programmed for INSEL and VSEit. One of the most important folders is the blocks folder, which contains very simple, “generic” examples for all blocks, available in INSEL. These examples are also used in the block reference manual’s description section.

**inselExperiments** This is the place to figure out new INSEL concepts. Two current examples are the proof-of-concept for inselSirius (Kai is mainly working on that) and the re-implementation of the Building Physics and Usage Libraries adapted from SimStadt. Once the proof-of-concept is positively validated, the corresponding projects should be moved to an appropriate repository, like inselSirius (does already exist as a prototype), or inselBuildingLibraries, for example.

**inselTools** inselTools is a kind-of support repository. it contains the inselToolsLibrary project (delivering the products inselTools.dll, libInselTools.dylib, etc.) inselTools is linked to practically all libraries used by INSEL, i. e., the block libraries, the engine, etc.

Other products are (i) everything related to textual outputs, i. e., the INSEL message system, libraries for textual output to a terminal, output to the VSEit window, to files, etc. In addition inselTools provides the inselHelp executable, Python and Ruby support for INSEL, a converter named src2text from INSEL source code comments to  $\TeX$ files, Simulink support, etc.

**inselUserBlocks** The inselUserBlocks repository is comparable to the inselBlocks repository, but it is

much less “sensitive”. It is meant as a starting point to INSEL users who want to implement their block ideas in this repository for detailed inspection by “the” INSEL developer crew.

**inselVSEitUI** Last, but not least, there is the inselVSEitUI repository. VSEit has been “the” diagram editor insel INSEL 8, based on Kai’s Java framework. It is now deprecated and the last INSEL version which will use VSEit is INSEL 8.3 (including long-term support, bugfixes, and smaller updates). Most probably, Sirius will be the VSEit successor, but we are still in a proof-of-concept phase now (March 2020). We, i. e., Kai and myself (Jürgen) decided that only the two of us continue to work on VSEit for the release of INSEL 8.3, and then have “a VSEit funeral” with a couple of glasses of Champagne.

Two other projects are part of the inselVSEitUI repository. These are the inselBlockWizard, a somehow strange combination of a Netbeans-based project and an Eclipse-based part. The other project is the inselBridge, which connects the Java world of VSEit with the C++ world of the inselEngine. The technology of the inselBridge is based on JNI (the Java Native Interface).

As mentioned several times, VSEit will be used in INSEL 8.3, all new INSEL versions will use Sirius as the basis for INSEL’s versions 9+. Whether inselSirius is going to be the basis of INSEL 4D or not, with Eclipse-as-such as an alternative, is not yet decided.

# 1 :: Build INSEL 8 from the sources

There are requirements before you can actually start with this project. The code is written in several programming languages. The `inselEngine` is in C++ and makes use of the compiler-compiler tools `flex` (scanner generator) and `bison` (parser generator). The `inselTools` are mostly C++ as well, but there are also some routines in Fortran.

Almost all blocks (talking about INSEL 8) are in Fortran, only some in C++. So far, there is no support for blocks written in Java, Ruby, or Python, but this will hopefully change in the near future. The documentation is completely based on  $\text{\LaTeX}$ . The `.pdf` document generation uses Ruby scripts mostly.

The graphical user interface of INSEL 8 is based on a framework named `VSEit` (pronounced as use-it) and is mainly used for the creation of INSEL models (block diagrams). The code is written in Java, the interface to the `inselEngine`—named `inselBridge`—is written in C++ and makes use of `JNI`, the Java Native Interface.

So, for a start you need at least

- :: a C++ compiler like `gcc`
- :: a Fortran compiler like `gfortran`, and
- :: an integrated development environment like Eclipse, and, of course,
- :: access to the Git repositories with the sources

There are plenty of pre-configured Eclipse installers, many of them have Git plugins and support for C/C++ development, e.g., the C/C++ Development Tools. They can all be downloaded from <https://www.eclipse.org/downloads/>

## 1.1 A Newbies' Hello World Introduction

1. install eclipse
2. hello x examples

fortran

c/c++

java

ruby

python

texlipse

git

## 1.2 Contribution of a new User Block

To start with a simple example—and that is probably the first use case for insel4D contributors—let us assume that you have a good idea and a model for a new user block and you want to add it to the inselUserBlocks repo. The first step is obvious: clone the inselUserBlocks repository to your local machine.

One option: General Project

In order to separate the Eclipse workspace from the project folders, deselect Use default location, and browse to the folder with your project files. If do not yet have a folder you want to use, you can create a new folder and click Open. Next you can name the project—it is recommended to use the folder name as project name, but keep in mind that they are different. One is the name of the folder—as you can see in Finder, the other is the project name, hidden in the .project directory in the project folder you just created. The project name is then visible in Eclipse's Project Explorer, for instance.

Now you can add some content to the new project. For INSEL block libraries that is typically a folder named src where usually Fortran and/or C/C++ files are located plus a makefile like this

```
all: inselST
sourcesF := $(wildcard ./src/*.f)
sourcesC := $(wildcard ./src/*.cpp)
objectsF := $(patsubst %.f,%.o,$(sourcesF))
objectsC := $(patsubst %.cpp,%.o,$(sourcesC))
objects := $(patsubst ./src%,.%, $(objectsF)) \
            $(patsubst ./src%,.%, $(objectsC))

inselST:
    @echo 'Building target $@'
    /usr/local/bin/gfortran -c -O0 -Wall -fno-automatic \
        -fno-underscoring -fmessage-length=0 $(sourcesF)
    g++ -O0 -Wall -c -fmessage-length=0 $(sourcesC)
    /usr/local/bin/gfortran -linseSTools -L. -dynamiclib \
        -olibInselST.dylib $(objects)
    mv libInselST.dylib ../../bin
    rm *.o

clean:
```

The make utility is mostly used for the creation of INSEL libraries, one reason being that it gives users the maximum flexibility to define their properties.

...

File - New - Convert to a C/C++ Project (Adds C/C++ Nature)

build the project output: Project - Build

### 1.3 Understanding the INSEL Message System

Text output from software is a complex field. A quick-and-dirty solution of a text message would be something like

```
print*, "I am a text message in Fortran"
```

or

```
printf("I am a text message in C/C++");
```

or

```
System.out.println("I am a text message in Java");
```

Basically, there are—at least—three problematic issues about text output.

(i) Every programming language has its own syntax for text output, but okay, that's not a big issue.

(ii) When text is used as string literals in the source code, what if you want a—let's say—German version of your software? This implies to go through the whole set of source code files, replace all the string literals by German text, recompile everything, resulting in two versions of your software.

Java, for example, offers a method called internationalization, or short `i18n` (very funny shortcut, because `internationalization` is just 18 characters) where the text is implemented as a placeholder like

```
println(i18n("PlaceholderText"));
```

for example. The final text to be displayed is stored in a `.properties` file and—depending on the language settings—the appropriate `.properties` file is used. Hence, no need to make any changes to the source codes.

The Java `i18n` concept goes much further than just text output. It can be used with graphical user interfaces, menu items, practically everything related to text as such. INSEL, per se, does not have a graphical user interface, so that the `i18n` concept of INSEL is much slimer.

(iii) The third question is, “where should the text message go to?” All the above mentioned methods in Fortran, C/C++, and Java send the message to the standard output, `stdio`, which is usually a terminal from where “the program” is executed. But what, if you want to send the message to a specific GUI window, a website, an email adress, or a log file, for instance?

INSEL offers these options via a function named `os0txt` (where `os` stands for operating system, `0` for any operating system, and `txt` for text, obviously)<sup>1</sup>

So, the INSEL solution is, to place `os0txt` into different libraries, and, depending on the environment, decide, which library to use and call `os0txt`. For the time being (July 2019) there are three different libraries implementing `os0txt`, i.e., `insetTextToTerminal`, `insetTextToFile`, and `insetTextToWindow`. The three libraries are available in a Git repository named `insetTools` at Concordia University, Montréal.

There is another `os0txt` implementation available in the `insetBridge` library, which connects INSEL with the VSEit GUI.

### 1.3.1 Implementation Details

```
insetEngine(Fenster, &iRunBit, argc, argv);

inset.msg
```

## 1.4 Understanding the INSEL Bridge

Include path to `jni.h` and `jni_md.h`

```
all: insetBridge
insetBridge:
    @echo 'Building target $@'
    g++ -O0 -Wall -m64 -c -fmessage-length=0 \
        -I"/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/include" \
        -I"/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/include/darwin" \
        -I"/src/insetBridge.cpp"
    g++ -shared -Wall -olibInsetBridge.jnilib -linsetTools -L. insetBridge.o
    mv libInsetBridge.jnilib ../../bin
    rm *.o
clean:
```

### 1.4.1 How does INSEL find binaries and libraries?

macOS answer: symbolic links in `/usr/local/bin` and `/usr/local/lib`

<sup>1</sup> INSEL has been developed in FORTRAN in the 1980's, and the FORTRAN standard allowed only six characters for variable- and function names—I know, last century (-)

## 2 :: Creation of the INSEL manuals

The INSEL documentation is completely based on  $\text{\LaTeX}$ , a typesetting system, developed by Donald E. Knuth. All documents are in English language, but the naming convention (`_en` for English, `_fr` for French, `_de` for German, etc.) allows for other language translations later.

There are six INSEL documentation manuals for INSEL 8.3

- 1 Getting Started
- 2 Tutorial
- 3 Block Reference Manual
- 4 Block Reference Manual for Simulink
- 5 User Block Reference Manual
- 6 INSEL Programmer's Guide

The `.tex` files, some support scripts are located in a Git repository named `insel_documentation`, which is hosted at Concordia. `insel_documentation` can be cloned with the command

```
git clone git-cerc@login.encs.concordia.ca:insel\documentation
```

The complete repository structure (as of February 28, 2020) is the following:

```
-rw-r--r--      1 juergenschumacher staff  8808 28 Feb 20:10 inseldoc_header.tex
-rw-r--r--      1 juergenschumacher staff  1961 28 Feb 20:10 disclaimer.tex
drwxr-xr-x    400 juergenschumacher staff 12800 28 Feb 20:10 icons
drwxr-xr-x     19 juergenschumacher staff   608 28 Feb 20:10 input
drwxr-xr-x     30 juergenschumacher staff   960 28 Feb 20:10 inselBlockReference
drwxr-xr-x     30 juergenschumacher staff   960 28 Feb 20:10 inselBlockReferenceSimulink
drwxr-xr-x     15 juergenschumacher staff   480 28 Feb 20:10 inselBlockReferenceUB
drwxr-xr-x      9 juergenschumacher staff   288 28 Feb 20:10 inselGettingStarted
drwxr-xr-x     14 juergenschumacher staff   448 28 Feb 20:10 inselProgrammersGuide
-rw-r--r--      1 juergenschumacher staff  8294 28 Feb 20:10 inselRepositoryStructure
drwxr-xr-x     36 juergenschumacher staff  1152 28 Feb 20:10 inselTutorial
drwxr-xr-x     10 juergenschumacher staff   320 28 Feb 20:10 out
drwxr-xr-x      5 juergenschumacher staff   160 28 Feb 20:10 pdf
drwxr-xr-x    376 juergenschumacher staff 12032 28 Feb 20:10 pic
drwxr-xr-x    378 juergenschumacher staff 12096 28 Feb 20:10 png
drwxr-xr-x     15 juergenschumacher staff   480 28 Feb 20:10 ruby
```

- ad 1 Getting Started is just an “ordinary”  $\text{\LaTeX}$  project, manually written and handtailored. The main `.tex` file is `inselGettingStarted_en.tex`. This file inputs `quickstart.tex`, which includes the bulk text. The reason behind this split is, that `quickstart`, and the Getting Started Module is also part of the Tutorial. But Getting Started is a stand-alone document in `.pdf` format.

There are several paths how to reach out to the .pdf file from the .tex sources. The most direct way is probably to use a terminal, cd to the directory which contains inselGettingStarted\_en.tex, any type

```
xelatex inselGettingStarted_en
```

In most cases, pdflatex would also do the job. But since the input file `\input{../inseldoc_header}` includes some XE specific packages and Xe<sub>La</sub>TeX is a superset of  $\text{\LaTeX}$ , the INSEL documentation makes use of this.

A second option is to use Eclipse as Integrated Development Environment with, for instance, TeXlipse and PDF4Eclipse as plug-ins.

Both options—needless to say—require a full  $\text{\TeX}$  installation. Everything else is just a matter of taste.

ad 2 The INSEL Tutorial is also a manually written and handtailored  $\text{\LaTeX}$  project. It consists of thirteen Modules, Module 1 being a copy of the Getting Started manual. With INSEL 8.3 a new type of Modules, named Workshops, was introduced.

Everything which was said to the Getting Started manual creation applies to the Tutorial as well. However, since the complete Tutorial has roundabout 400 pages, the main .tex file inselTutorial\_en.tex makes use of the “include” option in  $\text{\LaTeX}$ . For example,

```
\pagenumbering{arabic}
\include{PartOneFrontmatter}
\include{module1}
\include{module2}
..
```

shows, how the first two modules are “included” (as opposed to on input). This has the advantage that some parts of the document can be excluded from the development process of a certain module. For example, when from a list of includeonly’s just one module is uncommented, i.e.,

```
%\includeonly{module1}
\includeonly{module2}
%\includeonly{module3}
..
```

then pdflatex and xelatex ignores all the other modules during compilation.

ad 3, 4, and 5 From a production point of view, there is no principal difference between the three reference manuals. In contrast to the Getting Started manual and the Tutorial, they are not one-hundred percent handtailored. There are some scripts involved, the basic mechanisms are the following.

(i) The source code of all INSEL blocks (should) start with a comment section, which follows a certain syntax. All keywords start with a # symbol, examples are #Begin, #Block,

#Layout, etc. A complete description of the keywords and their use can be found in Module 12 :: Programming INSEL Blocks of the Tutorial.

(ii) The code is converted into .tex code by the src2tex executable, located in the inselTools repository. In addition, src2tex checks whether there is a description file with extension .des and the same name as the current block, and, if yes, includes the manually written and handtailored T<sub>E</sub>X code into the blocks .tex file.

(iii) The use of src2tex is straight forward. For example,

```
src2tex fb0004.f
```

creates div.tex, given that fb0004.f (which contains nothing but the DIV block) is located in the current directory where src2tex is executed.

(iv) The final .tex files for the reference manuals are generated by Ruby scripts. The main structure of the manual's chapters is

```
\cleardoublepage
\mchapter{Fundamental Blocks}
\cleardoublepage
\input{includeFB}
```

The main components are the includeXX files. In the current version, there are five chapters and XX stands for: (a) fundamental blocks FB, (b) energy meteorology EM, (c) solar electricity SE, (d) solar thermal ST, and (e) building simulation BS. The corresponding Ruby scripts which generate the five includeXX.tex files are named inselXX.rb, i.e., inselFB.rb for the fundamental blocks, for example.

The ruby scripts are located in a folder named ruby, which is directly located in the root directory of the insel\_documentation repository. In order to execute the scripts, cd to this directory and type

```
ruby inselFB.rb
```

and so on. After this step, inselBlockReference\_en.tex should be prepared for compilation and the result should be inselBlockReference\_en.pdf.

A final remark: the Ruby scripts rely on a very specific folder structure of the INSEL installation and make extensive use of relative paths. Should the directory structure of the insel development change, the Ruby scripts need to be adapted accordingly.

(v) In earlier versions of INSEL 8 the Help button of the Entity Editors in VSEit used hyperlinks to the corresponding blocks within the complete block reference manual. Since Adobe does not support this feature anymore, the Help button is now based on individual .pdf files, one for each block. These individual .pdf files are stored in the doc/BLOCKS directory of the insel installation.

Die Frage ist, wer generiert die ganzen .pdf's?

## 3 :: Ruby

INSEL comes with support for the Ruby language.

- :: ruby 1.9.3p194 (Ubuntu 13.04 (64 Bit)
- :: (Ubuntu 13.04 (32 Bit)
- :: ruby 2.0.0p353 (Fedora 20 (64 Bit)
- :: ruby 2.0.0p353 (Fedora 20 (32 Bit)

### 3.1 Ubuntu 13.04

64 Bit ruby can simply be installed by executing  
`sudo apt-get install gfortran`  
and  
`sudo apt-get install g++`  
in a Terminal.

32 Bit The same rules apply as for the 64 bit version of Ubuntu 13.04.

### 3.2 Fedora 20

64 Bit Ruby can simply be installed by executing  
`su -c "yum install ruby"`  
in a Terminal.

If Fedora nerves with "Another app is currently holding the yum lock" press  
`cmd C` and repeat a second attempt.

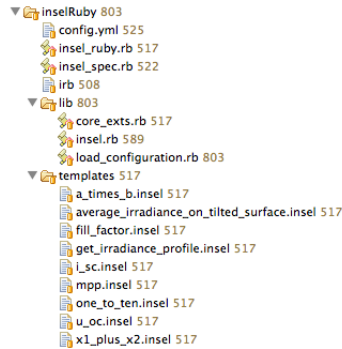
32 Bit The same rules apply as for the 64 bit version of Fedora 20.

### 3.3 Installation procedure

When Ruby support is wanted from INSEL it can be addressed from the Ruby button  in the toolbar.

If Ruby is not installed VSEit will display a corresponding message. Otherwise (only for the first time) the user is asked to confirm that the directory `resources/inseIRuby` (on Windows and Linux) or `Contents/inseIRuby` (on macOS), respectively, can be copied to the working directory—usually `inseI.work/inseIRuby`.

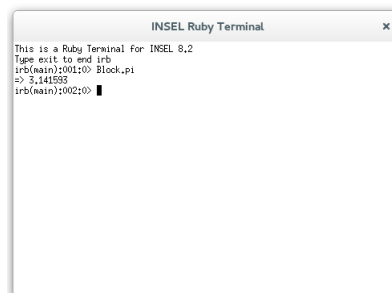
The `inseIRuby` directory has this structure:



The path to the INSEL executables must be adapted in `load_configuration.rb`:

```
module Insel
  ..version.....= 8
  ..default_configs => {
    ..linux'.....=> {'insel_path'=> '/usr/local/INSEL/resources'
    ..windows'.....=> {'insel_path'=> File.join(ENV['ProgramFiles'] || '', 'insel 8/resources')
    ..mac'.....=> {'insel_path'=> '/Users/Shared'
    ..wine'.....=> {'insel_path'=> '/usr/local/INSEL/resources'
  }
end
```

When everything is okay, the following result should be reached:



`irb`: interactive ruby

on start: `irb` executes `.irbc` (i) in user's home directory, (ii) in current directory. Both files are generated/updated by the `InselVseit.java` class.

(i) contains

```
# INSEL #
puts "This is /Users/juergenschumacher/.irbc"
if Dir.pwd != File.expand_path("~/insel.work/inselRuby")
  local_irbc = File.expand_path '.irbc'
  if File.exist? local_irbc
    load local_irbc
  end
end
```

end

(ii) contains

```
# INSEL #
puts "This is /Users/juergenschumacher/insel4D/inselTools/inselRuby.irbc"
require File.expand_path("lib/insel", File.dirname(__FILE__))
include Insel
puts "This is a Ruby Terminal for INSEL 8.3"
puts "Type exit to end irb"
```

As a result, the Insel module is loaded (located in `insel.rb`).

The Insel module defines several paths and includes the core extensions (`Array`, `Tempfile`, `Object`, and `Hash` classes) implemented in `core_exts.rb`.

## 4 :: Python

Python is a script language, i.e., an interpreted programming language, and not a compiled one, like Fortran or C/C++, for instance. What is puzzling sometimes is, that Python “compiles” .py source code files to .pyc **bytecode** files, which are then executed by Python’s virtual machine.

### 4.1 Installation

Today (April 2020) there are two Python “series”, i.e., Python 2.7 and Python 3.7. Both of them can be downloaded from the Python website

<https://www.python.org/downloads/>

The current bugfix version of the 2.7 series is 2.7.17, the latest 3.7 version is 3.7.7 (March 10, 2019). The latest release seems to be 3.8.2 (February 24, 2020).

Python 2.7 is the last major release in the 2.x series, as the Python maintainers have shifted the focus of their new feature development efforts to the Python 3.x series. This means that while Python 2 continues to receive bug fixes, and to be updated to build correctly on new hardware and versions of supported operated systems, there will be no new full feature releases for the language or standard library.

Once you have downloaded and installed Python, use

```
python -v
```

to check that you have a working Python environment on your computer. For example

```
Juergens-MacBook-Pro:inselPython juergenschumacher$ python --version
Python 2.7.10
```

### 4.2 Packages

Additional Python packages can be installed using pip (pip installs packages),

```
sudo python get-pip.py
```

Check out the version installed:

```
python -v
```

This will install the setuptools and wheel utilities as well. While pip alone is sufficient to install from pre-built binary archives, up to date copies of the setuptools and wheel projects are useful to ensure you can also install from source archives.

## 4.3 Interfacing INSEL 8 blocks with Python

The “interface” ie the formal parameters of INSEL han+ve not been changed since INSEL version 4.0.

In Fortran syntax the formal parameters of all INSEL blocks is

```
SUBROUTINE XXYYYY(IN,OUT,IP,RP,DP,BP,SP)
```

where XX represents a two-character code for the library, the block is implemented in (for examle, FB for Fundamental Blocks, EM for Energy Meteorology, UB for User Blocks, etc.) and YYYY is a unique, four-digit identifier for the subroutine. It is important to know, that this identifier is NOT identical with the INSEL block name, one subroutine can contain a set of different INSEL blocks. The INSEL block names are specified—in Fortran syntax—as a CHARACTER\*1024 variable named BNAMES. The number of INSEL blocks implemented in a subroutine is given through the OPM parameter (operation mode), represented by IP(3). For further deatls, please refer to the INSEL Tutorial.

The C/C++ syntax of INSEL blocks is very similar:

```
extern "C" void XXYYYY(float *in, float *out, int *IP, float *RP,
    double *DP, float *BP, char SP[][1024])
```

The C/C++ syntax shows, that all formal parameters are transferred as “pointers”, i. e., the Fortran and C/C++ codes have read/write access to all formal parameters. This can be considered as a “security hole”, because, for example, INSEL blocks should not be allowed to manipulate the input vector, because the input vector “belongs” to othe INSEL blocks, defined as their outputs. The same applies to the (numerical) block and string parameters.

Talking about Python interfacing the INSEL formal parameters is a completely different story. The main reason for that is, that Python has a VERY different concept of variables. While in Fortran and C/C++ every variable needs to be “decalred” with a specific assigned data type, Python variables do not have a declaration “per se”, they are variable “on-the-fly”, i. e., a Python variable named Name, for example, can be a string in one part of Python code, but it can also be used as something completely different, a numerical identifications number, for example, or whatever.