

INSEL

Programmer's Guide

INSEL 8 :: Programmer's Guide



© 1986–2019 Jürgen Schumacher. All rights reserved.

Please visit us at 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 was a registered trademark of doppelintegral GmbH, Stuttgart, Germany (liquidated in December 2019).
Current trademark/copyright holder to be confirmed.
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 history

The last release under doppelintegral GmbH (the company founded to turn INSEL into a product; liquidated in December 2019) was version 8.2.1.833, published in August 2014 for Windows (32 bit and 64 bit), macOS (64 bit), and Linux (Debian and RedHat, 32 and 64 bit).

Development then moved to Concordia University, Montréal, under the Canada Excellence Research Chair Next Generation Cities, and INSEL 8 was migrated from Subversion to Git. INSEL is now released as freeware (it may become open-source in the future), based on VSEit (Versatile Simulation Environment for the Internet, pronounced as "use it") as UI (User Interface). VSEit is still the actively maintained GUI, contrary to earlier plans to replace it.

INSEL is currently maintained by an international team (Guillermo Gutierrez Morote, Kai Brassel, and Eric Duminil). Jürgen Schumacher, INSEL's creator and lifelong developer — and author of most of this documentation — passed away in 2020. For up-to-date project news, downloads, and roadmap, see insel.eu.

Repository hosting

The INSEL repositories are hosted on a Gitea instance at Concordia University, under

```
https://ngci.encs.concordia.ca/gitea/insel/
```

Each repository can be cloned over HTTPS, e.g.

```
git clone https://ngci.encs.concordia.ca/gitea/insel/insel_documentation.git
```

Access

Access to the Gitea instance is restricted to INSEL contributors. If you are interested in contributing, contact the current maintainers (see insel.eu/development for contact details) to request an account.

Gitea also supports cloning and pushing over SSH once your public key is added to your account. 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: you can use them 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. Add it to your Gitea account under *Settings* > *SSH / GPG Keys*, then repositories can be cloned using the SSH URL shown on each repository's Gitea page instead of the HTTPS one.

Current repositories

As of this writing, the following repositories are hosted at Concordia (in alphabetical order):

- `insel_blocks`
- `insel_data`
- `insel_deployment`
- `insel_documentation`
- `insel_engine`
- `insel_examples`
- `insel_py`
- `insel_tools`
- `insel_user_blocks`
- `insel_vseit_ui`

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

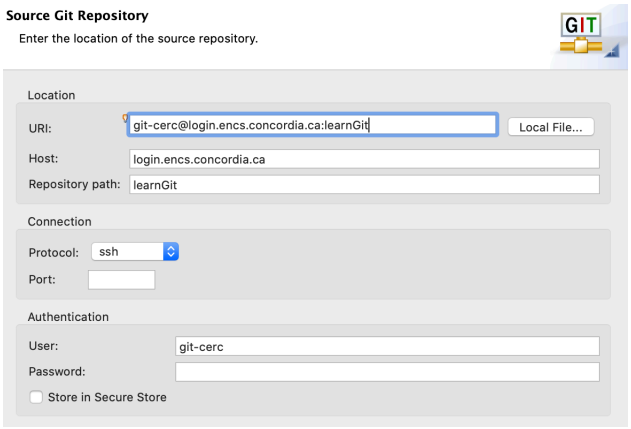
1. Start Eclipse.

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, e.g.

`https://ngci.encs.concordia.ca/gitea/insel/insel_documentation.git`

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 the repository and its projects listed 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!

What each repository contains

insel_blocks

The insel_blocks 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, grouped by topic: building simulation (BS), energy meteorology (EM), fundamental blocks (FB), power plants (PP), solar electricity (SE), and solar thermal energy (ST).

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).

insel_data — The insel_data repository is a collection of INSEL-related data, like block parameter sets, the meteorological "data base" of INSEL (the mtm folder), specific weather data collections (weather folder), etc.

insel_deployment — The insel_deployment repository contains the build orchestration: it clones/updates all sibling repositories, compiles the engine, tools, bridge, blocks and GUI, generates the documentation, and packages everything into Linux (.deb) and Windows (.exe) installers, using Docker containers so that neither a local LaTeX nor a local compiler toolchain is strictly required.

insel_documentation — This repository (the one you are currently reading) contains all LaTeX files, .png graphics, TeX picture files .pic, and the Python-based src2tex tooling that extracts block documentation from the source code comments in the insel_blocks and insel_user_blocks libraries.

The main products of this repository are the INSEL Block Reference Manual, the INSEL Userblock Reference Manual, the Getting Started Manual, the INSEL Programmer's Guide, and the INSEL Tutorial.

insel_engine — The insel_engine repository is the "heart" of INSEL, bringing together all the bolts and nuts, gears and wheels, and so on. It contains a full compiler, including a scanner and a parser based on the compiler-compiler tools Flex and Bison. That means, working on the insel_engine 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.

insel_examples — 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.

insel_py — The insel_py repository is the source of `inselpy`, published on PyPI (pypi.org/project/insel), which lets Python programs run INSEL models and templates directly, without going through the GUI.

insel_tools — insel_tools is a kind-of support repository. It contains the inselToolsLibrary project (delivering the products inselTools.dll, libInselTools.dylib, etc.), which 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., and (ii) src2tex, the Python package that converts INSEL source code comments into TeX files for this documentation.

insel_user_blocks — The insel_user_blocks repository is comparable to the insel_blocks repository, but it is

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

insel_vseit_ui The insel_vseit_ui repository contains the Java/JavaFX GUI of INSEL 8.3, based on Kai’s VSEit framework. It is made of three sub-projects: inselVSEitUI (the main application), inselBlockWizard (a block parameter dialog generator), and inselBridge (the C++/JNI library connecting the Java GUI to the native INSEL engine).

1 :: Build INSEL 8 from the sources

There are requirements before you can actually start with this project. The code is written in several progamming 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 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

The INSEL repositories are cloned side by side, e. g., by `make git_update` in `insel_deployment`. All build commands run from `insel_deployment`: `make all` compiles the tools, the engine, the bridge, the block libraries and the `insel` executable, `make help` lists all targets, e. g., `make linux_package` for a complete Docker-based build with packaging and tests.

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 new INSEL 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 `insel_user_blocks` repo. The first step is obvious: clone the `insel_user_blocks` 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.

The `insel_user_blocks` repository contains a folder `src` with the two templates `ubfor.f` (Fortran) and `ubcpp.cpp` (C++), and a `makefile` which compiles all files in `src` into the library `libInselUB`. New block files must start with `ub` in lower case, and each one defines the block function of the same name, e. g., `SUBROUTINE UBFOR` in `ubfor.f`. The makefile lists these functions in the block table which `insel_library()` returns to the inselEngine. Every block uses the block API version 2, i. e., the interface `(CTRL,INS,OUTS,INTS,REALS,PARAMS,STRINGS)` declared in `insel_block.inc` and `insel_block.h`; the engine ignores libraries built for an older API with warning W04095. From the repository folder, the library is built with

```
mkdir -p build && make -C build -f ../makefile
```

The same makefile is installed as `makefile` in the INSEL installation directory, and the GUI runs it to compile the user blocks in `insel.work/inselUB`. Fortran files are compiled with

```
-fdefault-real-8 -fdefault-double-8 -fno-underscoring
-Werror=line-truncation -finit-real=snan -finit-integer=-999999999 -fPIC
```

like all INSEL blocks, see `insel_deployment/makefile.var`. All reals are double precision. Local variables are not static and uninitialised locals are NaN or -999999999, so that their use is easy to detect: any state of a block instance which must survive between two calls belongs in `INTS` and `REALS`.

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 nternationalizatio 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)[1]

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 differnt libraries implementing `os0txt`, i.e., `inselTextToTerminal`, `inselTextToFile`, and `inselTextToWindow`. The three libraries are available in a Git repository named `inselTools` at Concordia University, Montréal.

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

1.3.1 Implementation Details

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

```
insel.msg
```

1.4 Understanding the INSEL Bridge

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

```
all: inselBridge
inselBridge:
   @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" \
      "./src/inselBridge.cpp"
   g++ -shared -Wall -olibInselBridge.jnilib -linselTools -L. inselBridge.o
   mv libInselBridge.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`

[1] 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 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 five INSEL documentation manuals for INSEL 8.3

1. Getting Started
2. Tutorial
3. Block Reference Manual
4. User Block Reference Manual
5. INSEL Programmer's Guide

The .tex files and support scripts are located in the Git repository insel_documentation, hosted on Gitea at Concordia (see the "INSEL Git Repositories" chapter). It can be cloned with the command

```
git clone https://ngci.encs.concordia.ca/gitea/insel/insel_documentation.git
```

ad 1 Getting Started is just an "ordinary" 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 XeLaTeX is a superset of LaTeX, the INSEL documentation makes use of this.

Any LaTeX editor works equally well; a full TeX installation is required either way.

ad 2 The INSEL Tutorial is also a manually written and handtailored 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 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 sytax. All keyords 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 src2tex, a Python package located in the insel_tools repository (`insel_tools/inselSrc2Tex`), imported directly by the extraction script described below — there is no separate install or PATH entry needed. 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 TEX code into the block’s .tex file.

(iii) The extraction is driven by `inselBlockReference/01_extract_insel_src_tex.py`, which walks the source folders of the insel_blocks and insel_user_blocks repositories (grouped by BS, EM, FB, PP, SE, ST, UB, and Community) and calls `src2tex.extract()` on every `.f` and `.cpp` file it finds, writing one .tex file per block into the corresponding group subdirectory.

(iv) The final .tex files for the reference manuals are assembled by the `makefile` at the root of the insel_documentation repository, either locally (requiring LATEX, Python, and the sibling insel_tools and insel_blocks/insel_user_blocks repositories) or, more conveniently, inside the Docker image built from `docker/Dockerfile` via `make release`, which has every required dependency preinstalled. See `README.md` at the root of the repository for the exact requirements and targets.

(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 generated by `inselBlockReference/02_compile_every_block_pdf.py` and stored in the `doc/BLOCKS` directory of the insel installation.

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.

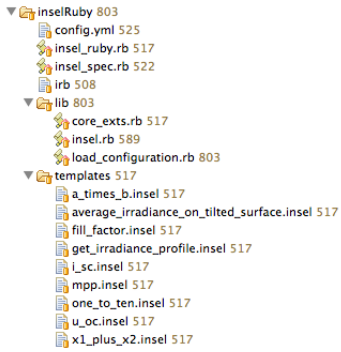
32 Bit The same rules apply as for the 64 bit version of Fedora 20.

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/inselRuby` (on Windows and Linux) or `Contents/inselRuby` (on macOS), respectively, can be copied to the working directory—usually `insel.work/inselRuby`.

The inselRuby directory has this structure:

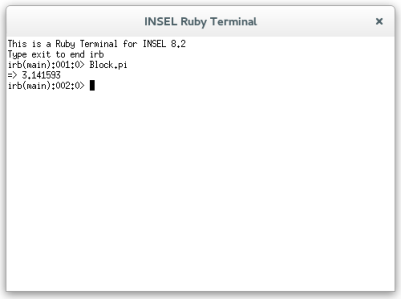


The path to the INSEL executables must be adapted in `load_configuration.rb`:

```
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'
  }
```

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_irbrc = File.expand_path '.irbrc'
  if File.exist? local_irbrc
    load local_irbrc
  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

INSEL can be called from Python using `inselpy`, published on PyPI at

`https://pypi.org/project/insel/`

It lets Python code run INSEL models (`.insel` files), templates, and individual blocks directly, without going through the GUI, and is the recommended way to script or test INSEL from Python. See the *Development* page of insel.eu for usage examples ('.insel' files, templates, GUI templates, and writing tests for INSEL models).