Safety Solutions Designer

Atelier B

User Guide

Version: 0.7.1

2024-12-13

Contents

1 Introduction

This guide provides instructions to use Atelier B 24.04, available from www.atelierb.eu, for several Operating Systems covering Windows, MacOS, and Linux. It covers both the Community and the Professional versions. It enables a better understanding of the use of the Atelier B to develop and prove B and Event-B formal models. It provides an overview of the features of the modelling environment. A dedicated chapter introduces advanced concepts and features. This guide is also supplemented by more detailed documents on modelling, proof and code generation.

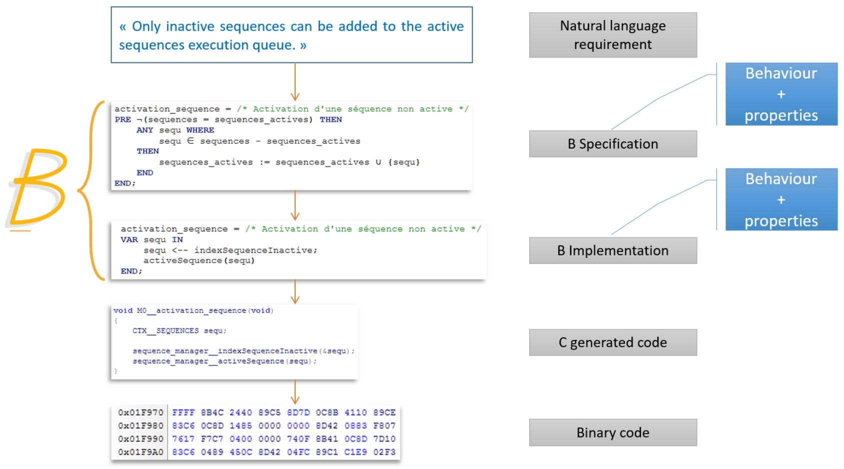
This writing of this guide is still in progress. The chapter on software design and the lexicon need to be completed, Chapter 6 needs to be written.

2 Introduction to the B Method

2.1 Principles

The B Method (Figure 1) is a model-based, formal specification approach that allows the rigorous expression of properties and the description of software behaviour in a modular and progressive way, with specifications that can be abstract. It is then possible to prove automatically that :

- these properties are consistent and not contradictory.
- the specification of the software functions verifies these properties


« Only inactive sequences can be added to the active sequences execution queue. »
B
Natural language requirement
Behaviour + properties
B Specification
Behaviour + properties
B Implementation
C generated code
Binary code

```
activation_sequence = /* Activation d'une séquence non active */
PRE ¬(sequences = sequences_actives) THEN
    ANY sequ WHERE
        sequ ∈ sequences - sequences_actives
    THEN
        sequences_actives := sequences_actives ∪ {sequ}
    END
END;
```

```
activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
    sequ <-- indexSequenceInactive;
    activeSequence(sequ)
END;
```

```
void M0__activation_sequence(void)
{
    CTX__SEQUENCES sequ;

    sequence_manager__indexSequenceInactive(&sequ);
    sequence_manager__activeSequence(sequ);
}
```

```
0x01F970  FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980  83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990  7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0  83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3
```

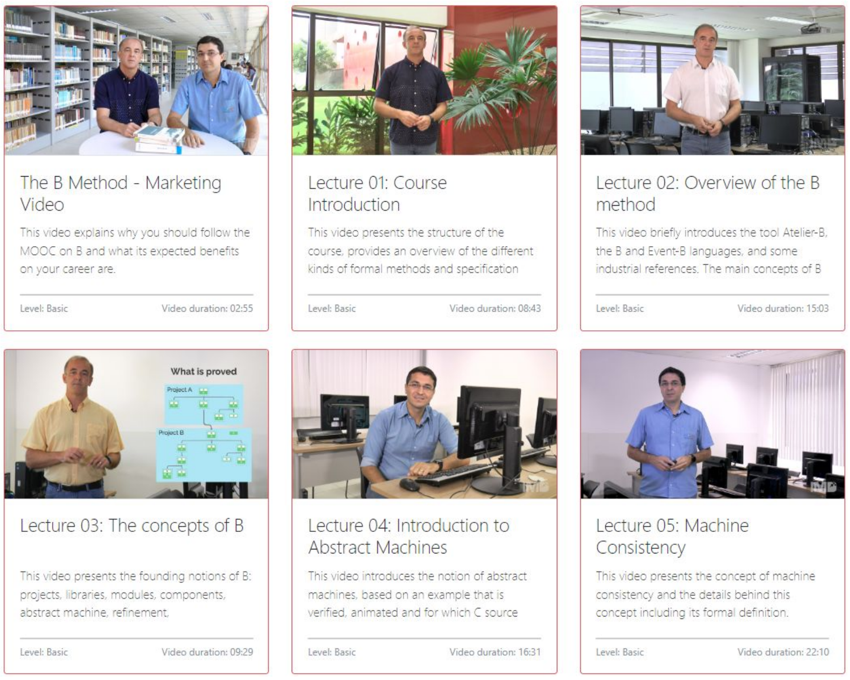
Figure 1: The B method consists of building a formal model of the software (specification and implementation) and proving it.

The mathematical proof provides a guarantee that these properties are respected throughout the design stages. The resulting software model is then considered flawless.

The B method does not cover the transformation of requirements into a specification model: an independent manned review will ensure traceability and coverage. It also does not cover the generation of source code from the implementation model: critical code review and/or diversified code generators can then be used.

2.2 Mastering the B Method

The reader should be familiar with the B Method to fully benefit from the content of this guide. For more information on the B Method, you are invited to connect to the MOOC "The B Method: from specification to code", freely accessible at mooc.imd.ufrn.br. This course introduces the B method: the basic concepts ranging from the most basic structures like the B machine to proofs using the Atelier B interactive prover. After the course, students are able to develop software from specification to source code using the B formal development approach.



The B Method - Marketing Video

This video explains why you should follow the MOOC on B and what its expected benefits on your career are.

Level: Basic Video duration: 02:55

Lecture 01: Course Introduction

This video presents the structure of the course, provides an overview of the different kinds of formal methods and specification

Level: Basic Video duration: 08:43

Lecture 02: Overview of the B method

This video briefly introduces the tool Atelier-B, the B and Event-B languages, and some industrial references. The main concepts of B

Level: Basic Video duration: 15:03

Lecture 03: The concepts of B

This video presents the founding notions of B: projects, libraries, modules, components, abstract machine, refinement,

Level: Basic Video duration: 09:29

Lecture 04: Introduction to Abstract Machines

This video introduces the notion of abstract machines, based on an example that is verified, animated and for which C source

Level: Basic Video duration: 16:31

Lecture 05: Machine Consistency

This video presents the concept of machine consistency and the details behind this concept including its formal definition.

Level: Basic Video duration: 22:10

Figure 2: The first videos of the MOOC on the B Method.

We also refer the reader to several high-quality references:

- The B-Book: Assigning programs to meanings, J.R. Abrial, Cambridge University Press (1996)
- The B-Method: an introduction, S. Schneider, Palgrave Macmillan (2001)
- Program Development by Refinement: case-studies using the B-Method, E. Sekerinski & K. Sere, Springer (1999)
- Spécification formelle avec B, H. Habrias, Lavoisier (2001)

For a presentation of system modelling with Event-B, we refer the reader to :

- Modeling in Event-B: system and software engineering, J.R. Abrial, Cambridge University Press (2010)

It should be noted that the B language supported by Atelier B differs in part from that described in the B-Book. Readers are invited to refer to the Reference Manual for the B language. Similarly, the Event-B language supported by Atelier B differs significantly from that presented in Modeling in Event-B.

3 Introduction to Atelier B

3.1 History

Atelier B was originally (in the early 90s) an internal Alstom tool used to implement the B method for the development of safety software for railways. It was used for the development of safety-related automated systems for line 14 of the Paris metro, which entered service in 1998. The key personnel in charge of the reengineering and development work of the tool created CLEARSY in 2001, which became the full owner of Atelier B and was given the responsibility to develop the tool and disseminate the method.

Since the first commercial Atelier B (version 3.2 in 1994) and developed under Unix/X-Motif for its graphical interface, many innovations have been made:

- the use of the Qt framework for porting to Windows and MacOS;
- part of the Atelier B source code (the most recent tools) has been made open-source;
- the availability of a free, fully functional *Community*[1] version in addition to the *Professional*[2] version.

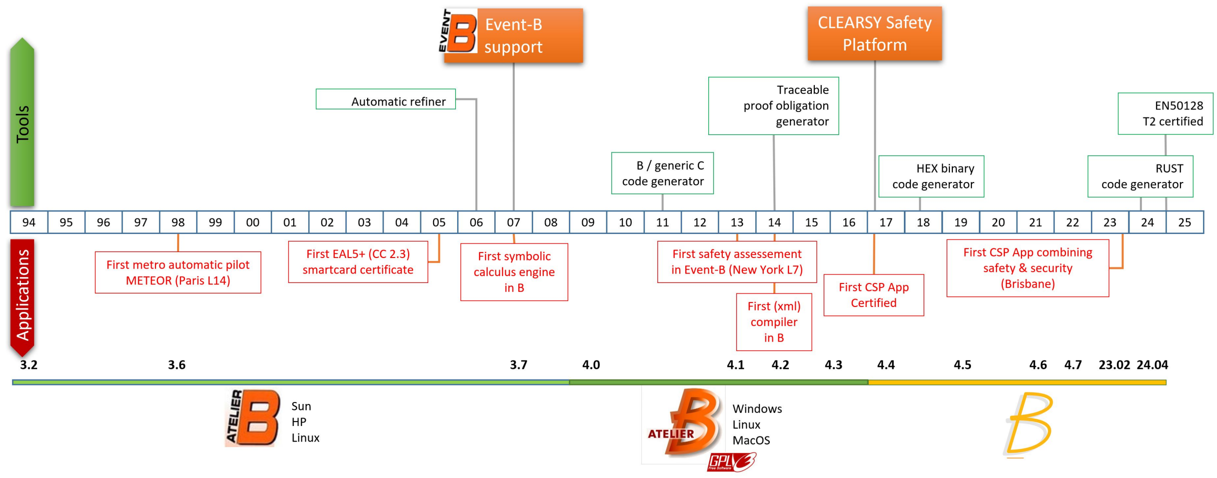

Event-B support
CLEARSY Safety Platform
Tools
Automatic refiner
Traceable proof obligation generator
EN50128 T2 certified
B / generic C code generator
HEX binary code generator
RUST code generator
94 95 96 97 98 99 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Applications
First metro automatic pilot METEOR (Paris L14)
First EAL5+ (CC 2.3) smartcard certificate
First symbolic calculus engine in B
First safety assessement in Event-B (New York L7)
First (xml) compiler in B
First CSP App Certified
First CSP App combining safety & security (Brisbane)
3.2 3.6 3.7 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 23.02 24.04
ATELIER B
Sun
HP
Linux
Windows
Linux
MacOS
GPL

Figure 3: Timeline showing the major functional evolutions of Atelier B and some remarkable applications.

A majority of the tools at the heart of the original Atelier B were developed in *theory language*, a language close to Prolog invented by Jean-Raymond Abrial and allowing easy manipulation of B models. Most of these tools have now been redeveloped in C++/Qt, with the exception of the provers, representing a total of 2Mloc written in 3 different programming languages.

[1]Supplied without support every 2 years. It includes a C code generator.
[2]Maintained version accessible to those with a maintenance contract. It is updated once or twice a year. It includes all the features of the Community version, as well as proof-of-rules tools.

Since 2001 the continuous maintenance and evolution of the tool has been ensured by CLEARSY, with academic and industrial support issued from maintenance fees and industrial contracts, as well as collaborative research projects.

Atelier B has been a resounding success, with downloads almost doubling over the last 2 years to more than 6,000 for the 2023-2024 academic year. The Windows version accounts for half of all downloads, while MacOS and Linux are on an equal footing. This interest is confirmed by the requests for CLEARSY to give presentations, courses and hands-on sessions around B, at universities and engineering schools in Western Europe and South America. Professional training with Atelier B is also continuing at a sustained level, mainly in the railway sector but also in microelectronics, when certification is based on Common Criteria and aimed at the highest security levels.

3.2 Installation

Atelier B is an integrated development environment to develop and prove models, and possibly to generate source code from the implementation models. It is available in 2 versions: **Community** and **Professional**. Several installers and an installation guide are available from the website. Supported Operating Systems are Windows (10, 11), Linux Debian (11, 12), Fedora (39), Ubuntu (22.04, 23.10, 24.04), and MacOS (up to MacOSX 11.0).

The **Community** version can be downloaded from the Downloads section. Only the latest version of the IDE is available, without support. The **Community** version is updated approximately every 2 years.

The **Professional** version is installed in the same way as the Community version, but requires access to a dedicated area for owners of a maintenance contract. The **Professional** version is updated up to 2 times a year.

To install Atelier B, you need around 200 MB of disk space. The disk space occupied by a project developed with Atelier B depends on the size of the B source files. The disk space occupied by the files generated automatically by Atelier B is approximately 25 times the size of the disk space of all the B source files.

The installation is straightforward. The only constraint/hint to consider is to install it in a directory where you have read/write access.

4 Atelier B in a Nutshell

4.1 Development Process

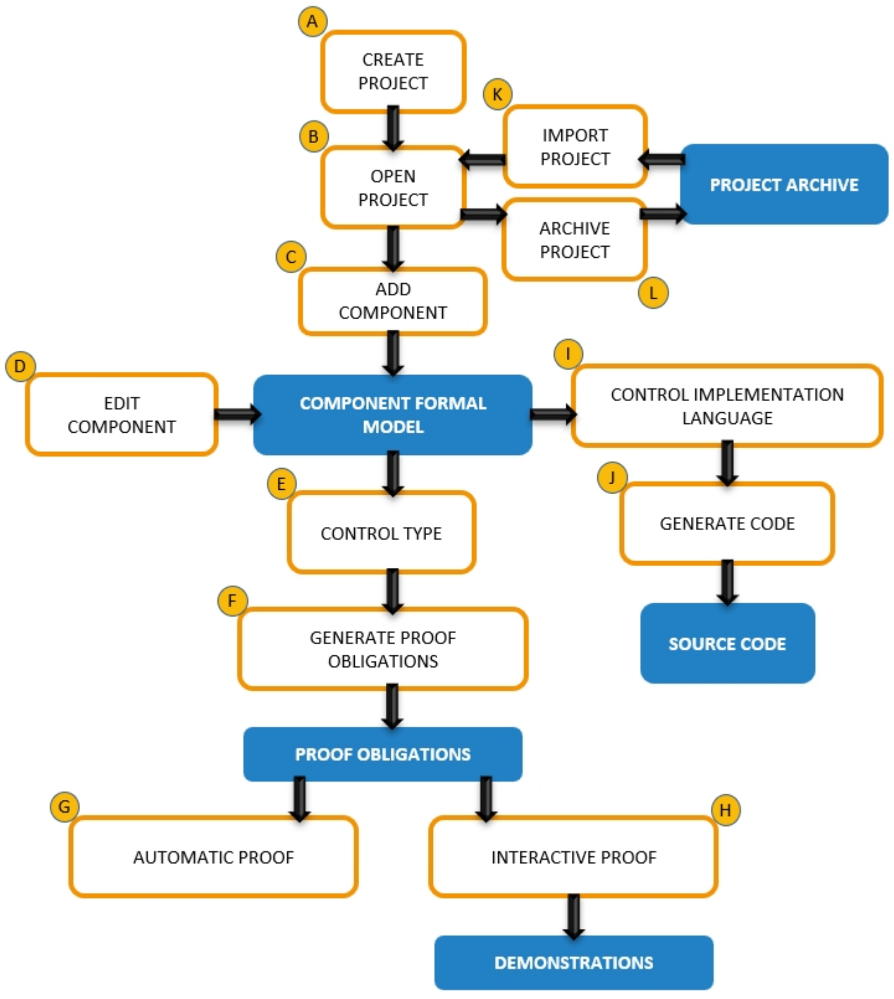


Figure 4: Using Atelier B for modelling, proving, and generating code. Orange boxes represent actions while blue boxes represent data persisted as files.

The full processus implemented by Atelier B is described (Figure 4) by actions named with uppercase letters. These actions are groups logically grouped in 3 sections:

- modelling: actions {A, B, C, D, K, L} modify the model files (specification, refinement, implementation) and project files (project descriptor, project archive)
- proving: actions {E, F, G, H} modify the proof files (proof obligations, saved demonstrations, mathematical rules, proof status)

- generating source code: actions {I, J} modify the generated source code (headers, body, makefile)

If modelling is mandatory, proving and generating source code are independent as it is possible to:

- prove a model without generating source code, if only a proof of coherence of the model is seeked;
- generate source code of an unproven model, but using this source code is risky as it comes without any guaranty of correction.

Atelier B comes with three main views for managing projects, for modelling/editing models, and for proving. Source code generation is triggered through menu items without much iteraction.

4.2 Project Types

Atelier B enables you to create 2 types of project: software development and system modelling.

4.2.1 Software development.

A software development project is used to build a model of sequential software. The model contains the specification and design of the functions performed by the software, specification and design being distinct. The design can then show the sequencing of functions from imported components, specified and designed in the same way. The dependency graph of a software project is a tree whose root is the highest-level component (see Figure 5).

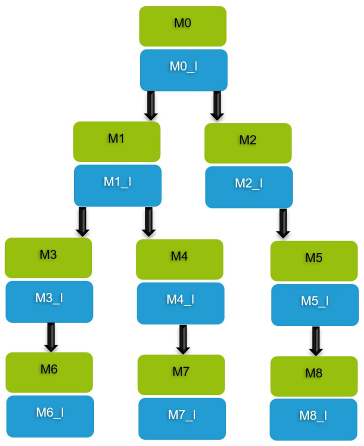


Figure 5: Example of a B project tree. The green components are the specifications (MACHINE), the blue components represent the implementations (IMPLEMENTATION), and the arrows show the import links (IMPORTS).

The software is obtained by translating each implementation component into source code. The development cycle of a software project is detailed in Figure xxxx. It is made up of two branches: one for generating code, the other for proving the model. It is possible to generate code without proving the model, but there is no guarantee that the software will be faultless. It is also possible to use dumb specifications (an operation specified by skip): in this case, the scope of the proof is considerably reduced, and the use of this technique must be justified and supervised.

4.2.2 System modelling.

A system modelling project is used to build a model of a system, in the broadest sense of the term (e.g. an aircraft, a level crossing, a processor, an administrative procedure). The model represents the system's state variables and the atomic events that modify its state, one after the other. This model implies an asynchronous and non-deterministic "execution" of its events. Any system can then be modelled, its specification can be equipped with properties and then proven to correspond to these properties. The dependency graph of a system modelling project is a refinement column that contains an abstract machine at the root and possibly a succession of refinements. A system model does not contain any implementation.

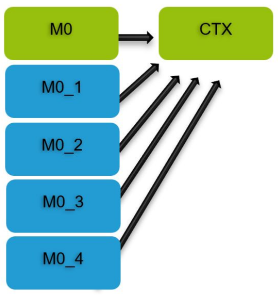


Figure 6: Example of an Event-B project tree. The green components are the specifications (SYSTEM), the blue components represent the refinements (REFINEMENT), and the arrows show the visibility links (SEES).

4.3 Graphical Interface

Atelier B can be used in :

- Interactive mode through a graphical user interface (GUI) designed to enable the B method to be applied effectively, i.e. to specify, refine, prove and generate code.
- Batch mode through a command line interface (CLI) based on a command language. This mode allows scripted use and is described in the chapter on bbatch).

Interactive mode provides the functionality required for developing formal models:

- Project management, through the component view ;
- Model editing, via the editor view;
- Mathematical demonstration of models, through the proof view.

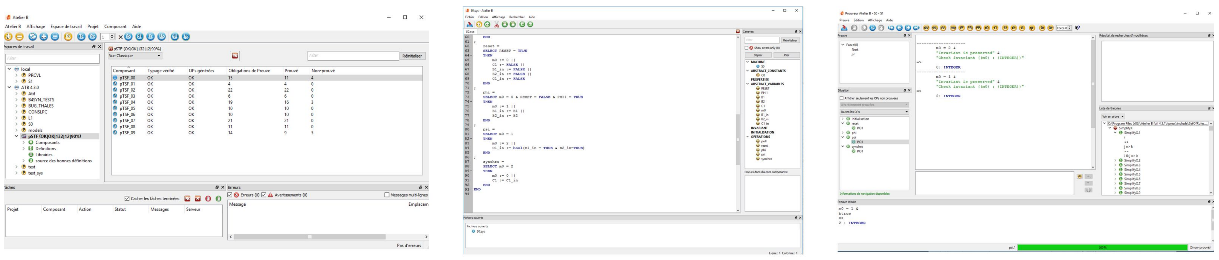


Figure 7: The three main views of Atelier B (from left to right): the component view, the editor view and the proof view.

There is no code generation view as this activity is limited to activate one menu item and to get the result in a dedicated folder.

There is also another view, the Interactive Refinement view, to apply automatic refinement to manual modelling, but it is not described in this document.

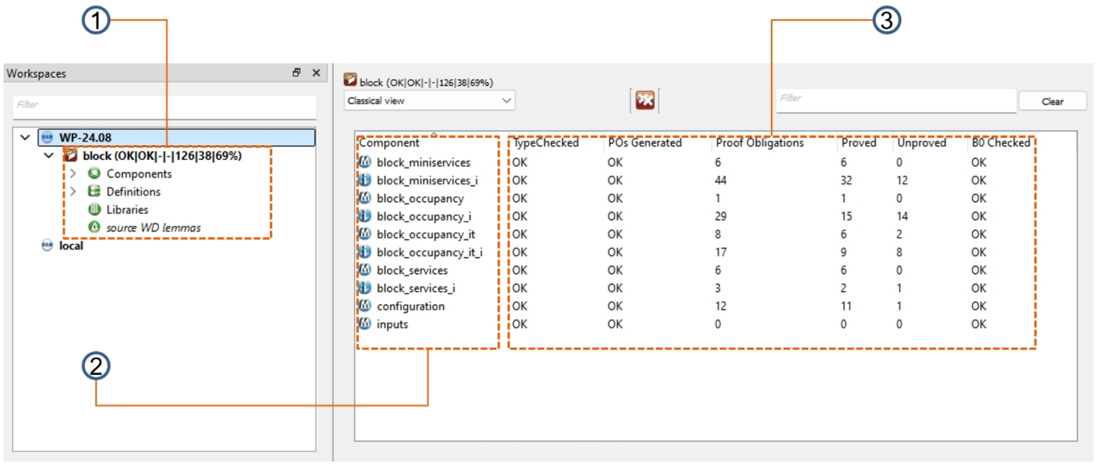
4.3.1 Component View

The Atelier B component view lists the various components of a project, their status (typing verification, proof obligations generated, ability to be translated into source code) and the number of proof obligations proven and to be proven. This view is central and provides an overview of the project and the tasks currently being carried out.

In Figure 8,

1. the current workspace (named here WP-24.08) contains one open project (named block). The two OK following the name ondicate that the project is fully typechecked and all proof obligations have been generated. 126 proof obligations have been generated and 38 are unproved - hence the proof ration of the project is 69 percent.
2. the project contains 10 components listed as either machines or implementations.
3. the status of the components is shown: typechecked completed, proof obligations generated, number of proof obligations, number of proved and unproved obligations, compliance with B0 conditions.

The component view is available in several types of representation: tabular (classic) or graphical (to



Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved	B0 Checked
block_miniservices	OK	OK	6	6	0	OK
block_miniservices_i	OK	OK	44	32	12	OK
block_occupancy	OK	OK	1	1	0	OK
block_occupancy_i	OK	OK	29	15	14	OK
block_occupancy_it	OK	OK	8	6	2	OK
block_occupancy_it_i	OK	OK	17	9	8	OK
block_services	OK	OK	6	6	0	OK
block_services_i	OK	OK	3	2	1	OK
configuration	OK	OK	12	11	1	OK
inputs	OK	OK	0	0	0	OK

Figure 8: Component view (classic) of a software project. The components are listed in alphabetical order. Specifications (Machine) are preceded by the M icon, implementations by the I icon.

be able to see certain values at a glance). The choice of view displayed is determined by the menu below. Note that the graphical view can be set to display from "top to bottom" or "left to right".

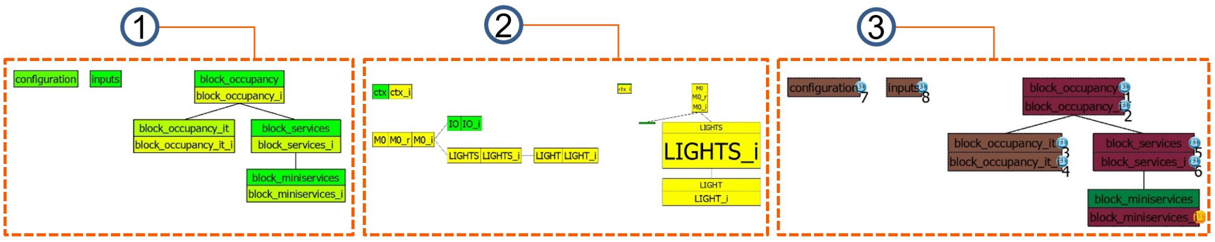


Figure 9: The graphical view allows to show several aspects of the project status at a glance.

In Figure 9,

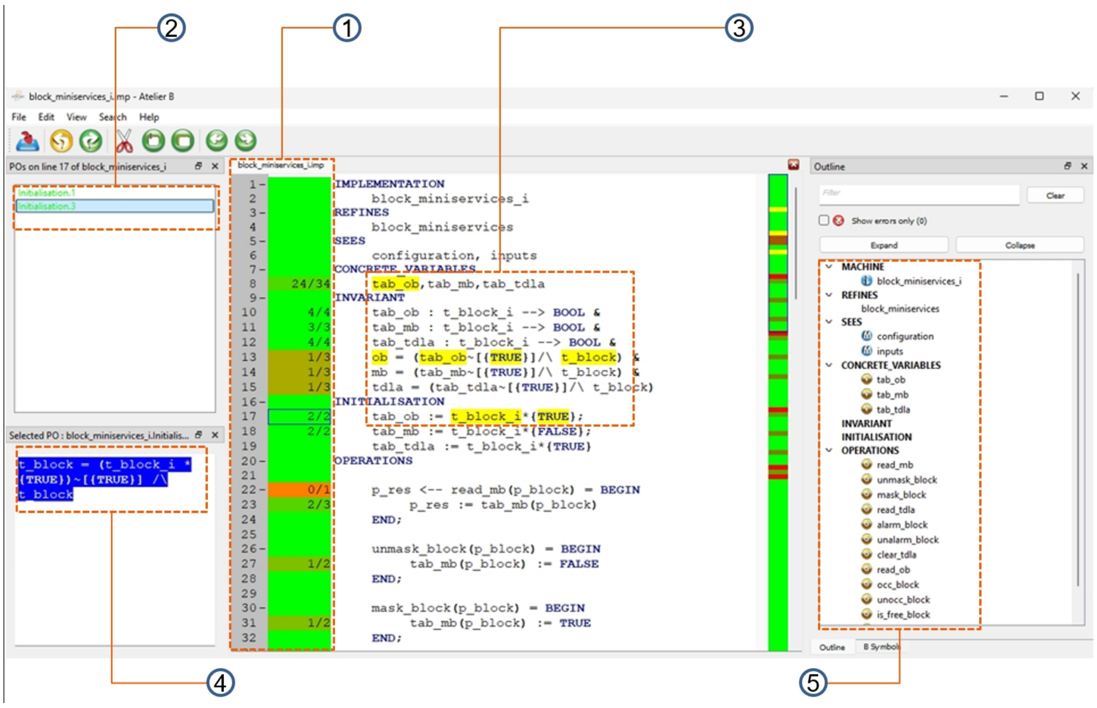
1. simply displays the architecture of the project. Single boxes are specification component (MACHINE). The sets of two or more boxes represent a complete component: the upper box is the specification (MACHINE), the lower box is the implementation (IMPLEMENTATION). The lines represent import links (IMPORTS). The colour of the boxes represents the rate of proof: from bright green indicating a rate of 100 percent, to bright red indicating a rate of 0 percent.
2. The display is similar to number 1, except that the size of the boxes is proportional to the number of proof obligations. In this way, you can see where the greatest number of unproven proof obligations are located.
3. The display is similar to number 1. Current tasks are displayed on top of the boxes. Here the components are being proven in force 1 (F1 picture). The numbers displayed indicate the

order in which the components will be proven.

4.3.2 Editing View

This view is used to enter :

- formal models (B components),
- mathematical rules and demonstrations for the proof phase (pmm files),
- refinement rules for the automatic refinement phase (rmf files).



```
t_block = (t_block_i *
{TRUE})~[{TRUE}] /\
t_block
```

```
IMPLEMENTATION
    block_miniservices_i
REFINES
    block_miniservices
SEES
    configuration, inputs
CONCRETE_VARIABLES
    tab_ob,tab_mb,tab_tdla
INVARIANT
    tab_ob : t_block_i --> BOOL &
    tab_mb : t_block_i --> BOOL &
    tab_tdla : t_block_i --> BOOL &
    ob = (tab_ob~[{TRUE}]/\ t_block) &
    mb = (tab_mb~[{TRUE}]/\ t_block) &
    tdla = (tab_tdla~[{TRUE}]/\ t_block)
INITIALISATION
    tab_ob := t_block_i*{TRUE};
    tab_mb := t_block_i*{FALSE};
    tab_tdla := t_block_i*{TRUE}
OPERATIONS

    p_res <-- read_mb(p_block) = BEGIN
        p_res := tab_mb(p_block)
    END;

    unmask_block(p_block) = BEGIN
        tab_mb(p_block) := FALSE
    END;

    mask_block(p_block) = BEGIN
        tab_mb(p_block) := TRUE
    END;
```

Figure 10: Editing a B model.

The editing of B components is accompanied by services such as the display of the model structure (outline), navigation, completion, display and localisation of errors, automatic indentation, spelling correction of comments, etc.

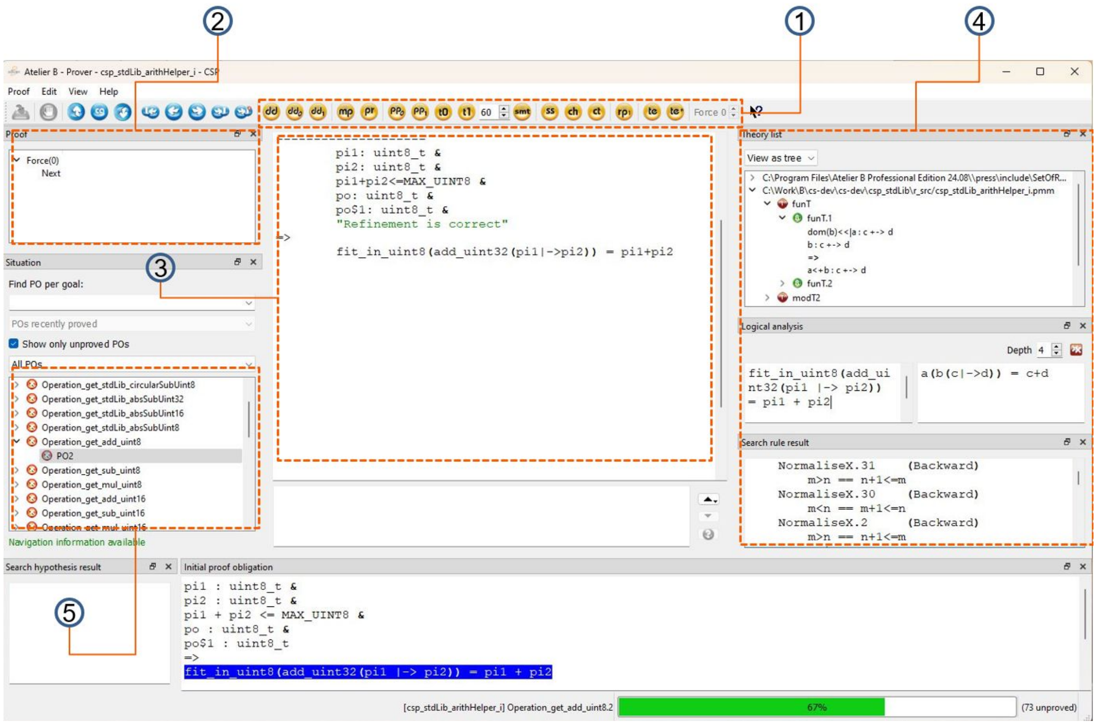
In Figure 10,

1. displays for each ling several information: the number of proven proof obligations, the total number of proof obligations, a color code representing the proof ration (from red (0 percent) to green (100 percent)).
2. lists the proof oligation associated to the last line you selected. The color of the elements indicate the proof status: red if unproven, green if proven.

3. is the B model you typed in. If you select identifiers in (4), the modelling elements used to construct the proof obligation predicate are highlighted, that way the relationship between the model and the proof obligation then becomes clearer and makes it easier to correct the model.

4.3.3 Proof View

This view is related to the interactive proof of proof obligations. This graphical interface is aimed at easing the completion of mathematical demonstration, by providing means to navigate the hypotheses, to search for expression and predicate for both hypotheses and mathematical rules, etc.



```
pi1: uint8_t &
pi2: uint8_t &
pi1+pi2<=MAX_UINT8 &
po: uint8_t &
po$1: uint8_t &
"Refinement is correct"
=>
fit_in_uint8(add_uint32(pi1|->pi2)) = pi1+pi2
```

```
dom(b)<<|a : c +-> d
b : c +-> d
=>
a<+b : c +-> d
```

```
fit_in_uint8(add_ui
nt32(pi1 |-> pi2))
= pi1 + pi2
```

```
a(b(c|->d)) = c+d
```

```
    NormaliseX.31     (Backward)
        m>n == n+1<=m
    NormaliseX.30     (Backward)
        m<n == m+1<=n
    NormaliseX.2      (Backward)
        m>n == n+1<=m
```

```
pi1 : uint8_t &
pi2 : uint8_t &
pi1 + pi2 <= MAX_UINT8 &
po : uint8_t &
po$1 : uint8_t
=>
fit_in_uint8(add_uint32(pi1 |-> pi2)) = pi1 + pi2
```

Figure 11: Proving a B model.

In Figure 11, are displayed

1. the toolbar to activate most common proof commands,
2. the proof tree, i.e. the list of commands used for the proof until now, and the remaining goals to demonstrate,
3. the proof obligation to demonstrate,
4. several tools used to ease the proof,
5. the list of the proof obligations of the current component.

5 The B Method for Modelling Software

This chapter presents the several steps that make up the B method.

5.1 Create Project

In Atelier B, developments are organized in so-called **B projects**.

5.1.1 Files Structure

Each B project has a data base directory (bdp) and a language directory (lang).

- The data base directory (bdp) contains all the internal artifacts necessary for Atelier B and bbatch to work.
- The language directory (lang) contains the files in a programming language, such as C, that are produced from the B components of the project.

To display the properties of a project, select it then right-click and select "properties". This opens the project properties dialog (Figure 12) where are displayed:

1. the project type, database, and translation directories,
2. the configuration details (click on it to get a window showing the complete configuration, including resources, of your Atelier B),
3. the user list that can be modified,
4. the libraries used and usable for the project,
5. the directories containing the definition files `def`.

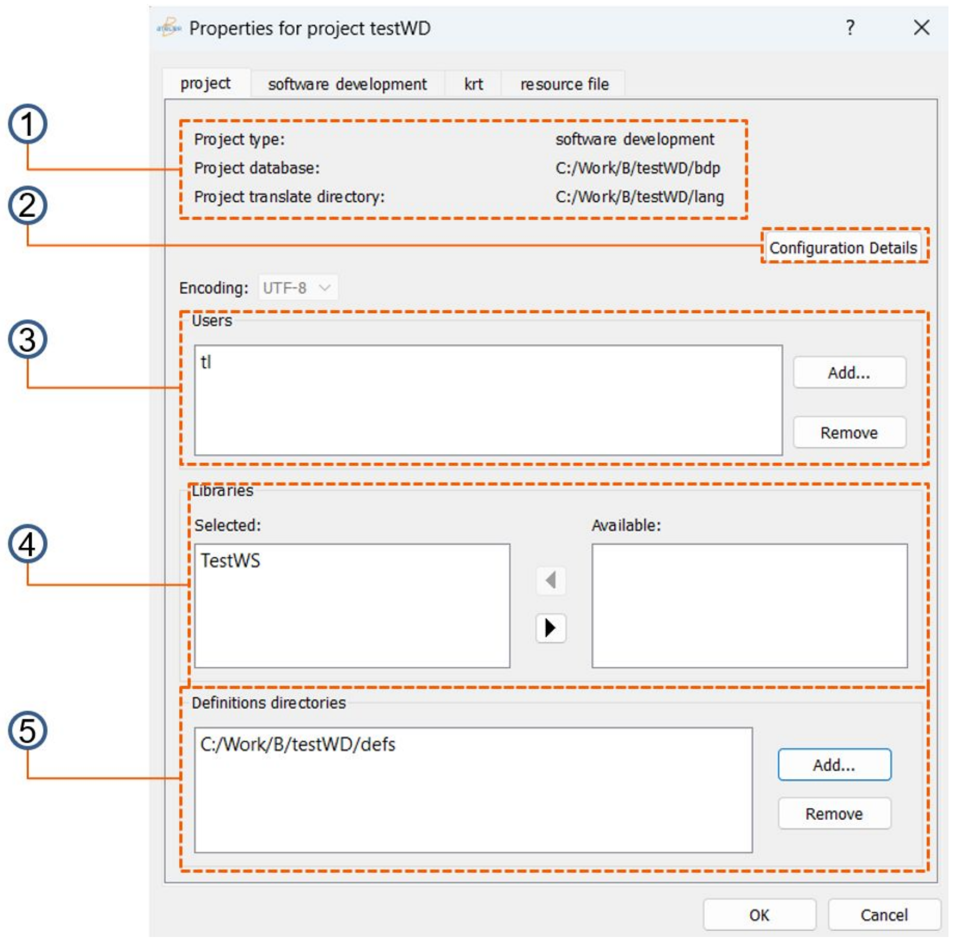
The B project is made up of a number of source files. Source files are either B or Event-B components. The list of the source files of a B project is stored in the bdp in an XML file with the extension `db`.

For each B component, there is a corresponding PMM file with the same name and the extension `pmm`, containing proof rules and proof tactics for this component. The PMM file shall be stored in the same directory as the corresponding source file.

It is good practice to store the source files of a B project in a single directory.

5.1.2 Workspaces

Atelier B lets the user have several **workspaces**, each workspace being a collection of projects. In practice, a workspace is associated to a project data base directory (bdb), where each file ending



Project type: software development
Project database: C:/Work/B/testWD/bdp
Project translate directory: C:/Work/B/testWD/lang
Encoding: UTF-8
Users: tl
Libraries Selected: TestWS
Definitions directories: C:/Work/B/testWD/defs

Figure 12: Project properties.

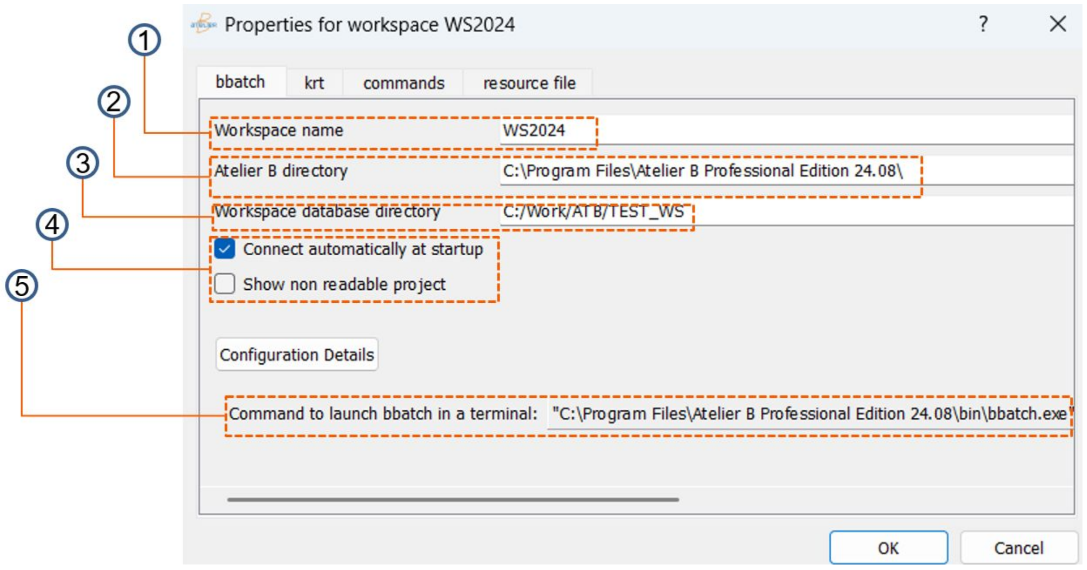
with the desc extension corresponds to a project. Actually, the contents of the desc file is only the path to the bdp and lang directories of the project.

A fresh install creates an empty workspace named "local" and associates it with a directory (bdb) located in the user's home directory.

To know what is the bdb of a workspace, right-click on the workspace in Atelier B and select "Properties". This opens the workspace properties dialog (Figure 13) where are displayed:

1. the name of the workspace,
2. the path of the installation of Atelier B,
3. the path to the bdb of the workspace (namely, the workspace database directory),
4. a checkbox to indicate whether the workspace is the default workspace.

It is currently not possible to change the path to the bdb of a workspace from the GUI. You may change the path to the bdb of a workspace by editing directly the settings of the tool.



Properties for workspace WS2024

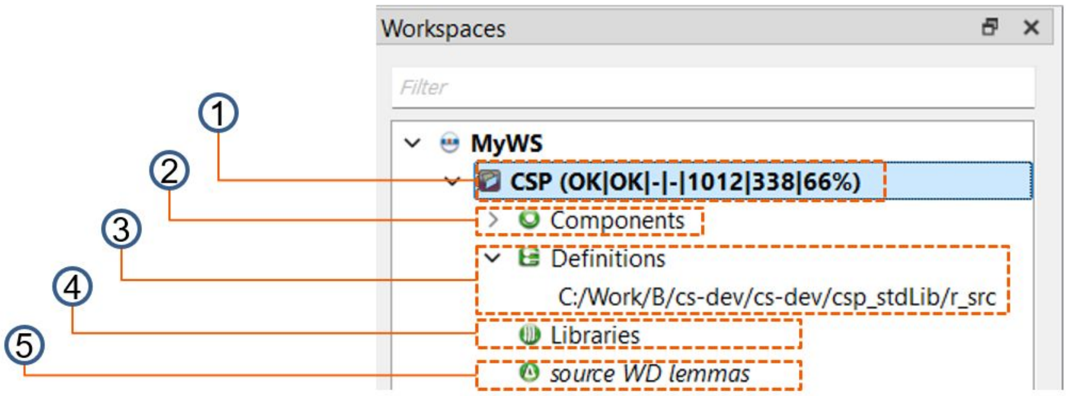
Workspace name: WS2024
Atelier B directory: C:\Program Files\Atelier B Professional Edition 24.08\
Workspace database directory: C:/Work/ATB/TEST_WS
[✓] Connect automatically at startup
[ ] Show non readable project
Command to launch bbatch in a terminal: "C:\Program Files\Atelier B Professional Edition 24.08\bin\bbatch.exe"

Figure 13: Workspace properties.

If you don't have write access to the "local" workspace, create another one and specify a path to the bdb where you can create and modify files and directories.

5.2 Open Project

Before accessing the components of project, it needs to be open first by doucle-clicking on it.



MyWS
CSP (OK|OK|-|-|1012|338|66%)
Components
Definitions
C:/Work/B/cs-dev/cs-dev/csp_stdLib/r_src
Libraries
source WD lemmas

Figure 14: Open a project.

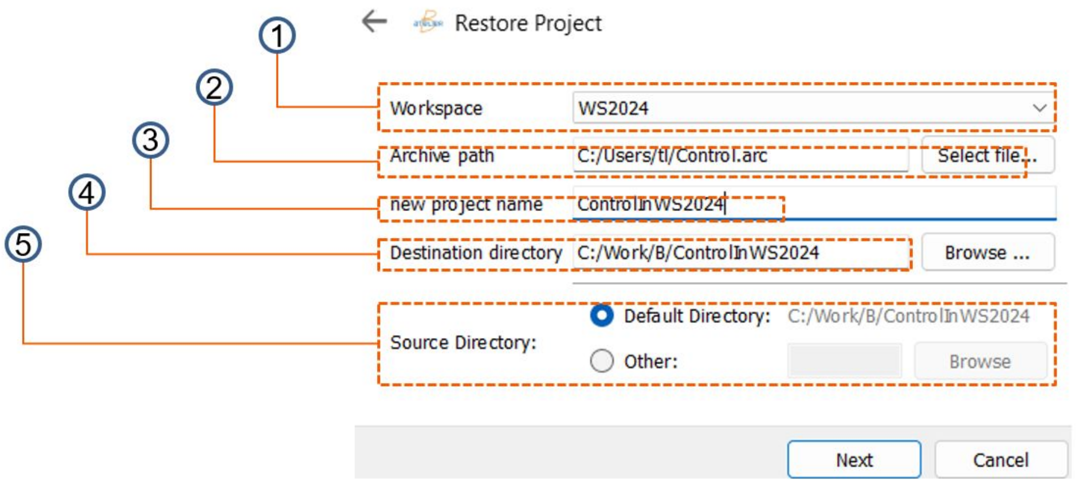
In Figure 14, are displayed

1. the open project (bold) with its current status (typecheck, proof obligation (PO) generation, total number of PO, number of proved PO, proof percentage),

2. the components,
3. the directories containing definitions,
4. the B librairies used by the project,
5. the well-definedness lemmas of the components (deprecated).

5.3 Restore Project

Restoring a project allows to transfer a project to another computer or another workspace. To restore a project archived preferably with the same version of Atelier B and Operating System.



Workspace: WS2024
Archive path: C:/Users/tl/Control.arc
new project name: ControlInWS2024
Destination directory: C:/Work/B/ControlInWS2024
Source Directory: [✓ Default Directory: C:/Work/B/ControlInWS2024] Other:

Figure 15: Restore a project.

Right-click any workspace and select "restore project". A windows shows up (Figure 15), where are displayed:

1. the workspace used to restore the project,
2. the full path to the archive `arc` file,
3. the new name of the project,
4. the destination directory for the project,
5. the source directory of the archived project.

5.4 Archive Project

Archiving a project allows to save a project, to keep track of a particular configuration or to prepare for the transfer to another computer or another workspace. To archive a project, select it then right-click "archive".

Three different archives `arc` can be generated:

1. sources files: only the B components,
2. whole project: the current architecture of the project, including the lang directory and the MANIFEST file,
3. sources and proof files: the B components and the related proof files `pmm`.

5.5 Add Component

To add a component to an open project, you can either:

- click on the + blue button
- type Ctrl+N
- select Atelier B / New / Component menu

You are asked for a component name and for a type (Machine, Refinement, Implementation). If it is a refinement or an implementation, it is better to first select the component you want to refine and then type Ctrl+N. The naming is then automatic and you are asked for the CLAUSES you want to replicate into the refined component. You are also asked for the directory you want to save the component in. By default, it is in the project directory.

If the file (Machine, Refinement, Implementation) already exists, you can drag/drop the component for the filesystem explorer to the Atelier B component view, only if this one is in Classical view (tabular). If it is in Graphical view, it is not going to work.

5.6 Edit Component

5.7 Control Type

5.8 Generate Proof Obligations

5.9 Prove Automatically

5.10 Prove Interactively

5.11 Control Implementation

5.12 Translate Target Language

6 Modelling Software Systems with Event-B

7 Advanced Use

7.1 BBatch

Atelier B batch mode allows to use it from a console, as well as semi-automatically using command files. The command interpreter, bbatch, has most of the same functions as the Atelier B graphical interface. To run the command interpreter, open a terminal, move to the directory containing the Atelier binaries and type the command

```
startBB
```

You can use this interface either interactively, or with a command file.

A command file can contain:

- Comments: lines beginning with the # character,
- Atelier B commands: long or abbreviated form.

The command

```
help
```

displays the commands usable at the current level.

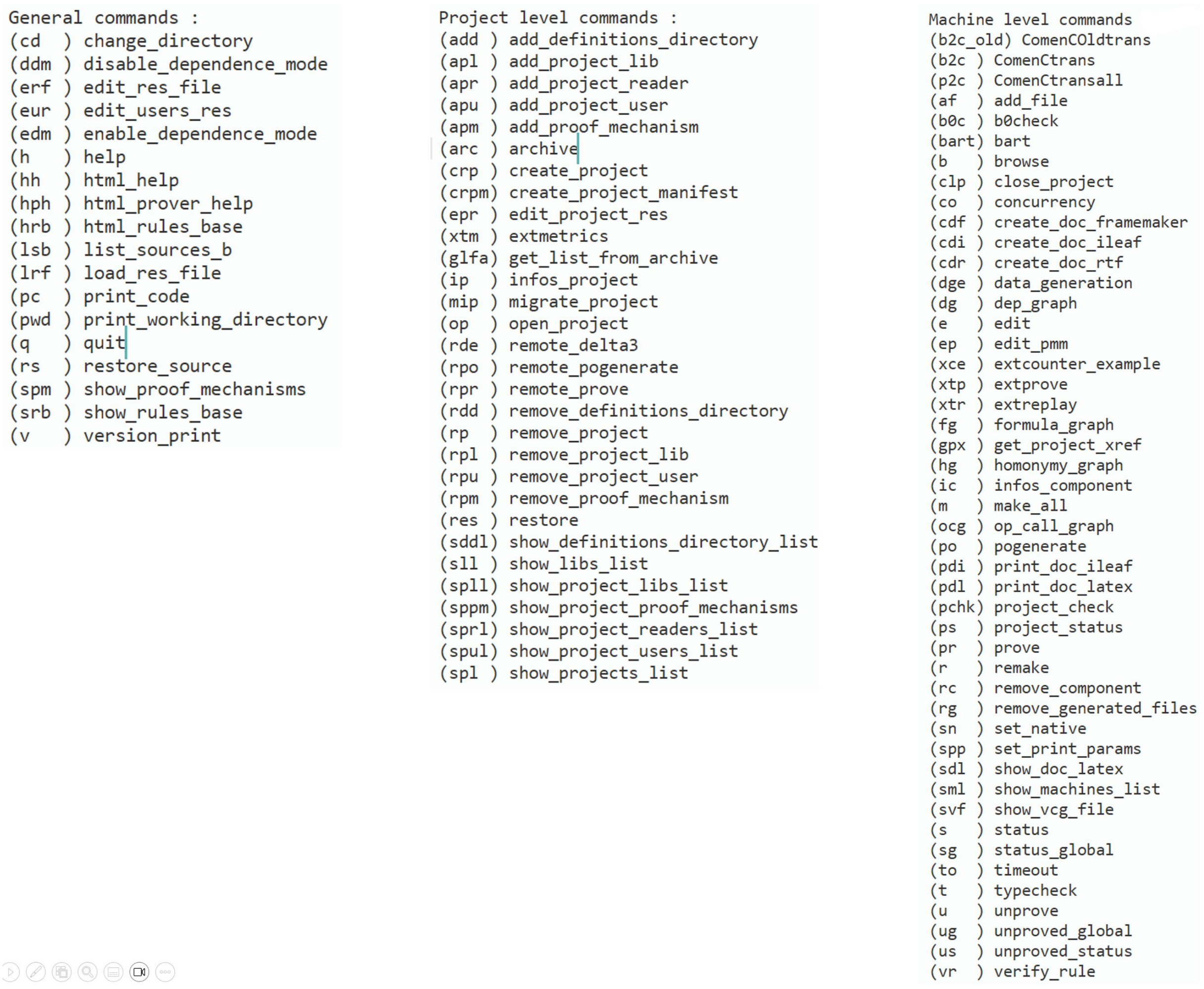
To get more information about a bbatch command, type:

```
help <command>
```

To execute the bbatch commands in a command file, type the following command:

```
startBB -i=<filename>
```

Some commands have default settings. For project management commands the interpreter remembers the last project name typed. Similarly, the interpreter remembers the last component name typed. To use this default parameter, just type <return>.



```
General commands :
(cd  ) change_directory
(ddm ) disable_dependence_mode
(erf ) edit_res_file
(eur ) edit_users_res
(edm ) enable_dependence_mode
(h   ) help
(hh  ) html_help
(hph ) html_prover_help
(hrb ) html_rules_base
(lsb ) list_sources_b
(lrf ) load_res_file
(pc  ) print_code
(pwd ) print_working_directory
(q   ) quit
(rs  ) restore_source
(spm ) show_proof_mechanisms
(srb ) show_rules_base
(v   ) version_print
```

```
Project level commands :
(add ) add_definitions_directory
(apl ) add_project_lib
(apr ) add_project_reader
(apu ) add_project_user
(apm ) add_proof_mechanism
(arc ) archive
(crp ) create_project
(crpm) create_project_manifest
(epr ) edit_project_res
(xtm ) extmetrics
(glfa) get_list_from_archive
(ip  ) infos_project
(mip ) migrate_project
(op  ) open_project
(rde ) remote_delta3
(rpo ) remote_pogenerate
(rpr ) remote_prove
(rdd ) remove_definitions_directory
(rp  ) remove_project
(rpl ) remove_project_lib
(rpu ) remove_project_user
(rpm ) remove_proof_mechanism
(res ) restore
(sddl) show_definitions_directory_list
(sll ) show_libs_list
(spll) show_project_libs_list
(sppm) show_project_proof_mechanisms
(sprl) show_project_readers_list
(spul) show_project_users_list
(spl ) show_projects_list
```

```
Machine level commands
(b2c_old) ComenCOldtrans
(b2c ) ComenCtrans
(p2c ) ComenCtransall
(af  ) add_file
(b0c ) b0check
(bart) bart
(b   ) browse
(clp ) close_project
(co  ) concurrency
(cdf ) create_doc_framemaker
(cdi ) create_doc_ileaf
(cdr ) create_doc_rtf
(dge ) data_generation
(dg  ) dep_graph
(e   ) edit
(ep  ) edit_pmm
(xce ) extcounter_example
(xtp ) extprove
(xtr ) extreplay
(fg  ) formula_graph
(gpx ) get_project_xref
(hg  ) homonymy_graph
(ic  ) infos_component
(m   ) make_all
(ocg ) op_call_graph
(po  ) pogenerate
(pdi ) print_doc_ileaf
(pdl ) print_doc_latex
(pchk) project_check
(ps  ) project_status
(pr  ) prove
(r   ) remake
(rc  ) remove_component
(rg  ) remove_generated_files
(sn  ) set_native
(spp ) set_print_params
(sdl ) show_doc_latex
(sml ) show_machines_list
(svf ) show_vcg_file
(s   ) status
(sg  ) status_global
(to  ) timeout
(t   ) typecheck
(u   ) unprove
(ug  ) unproved_global
(us  ) unproved_status
(vr  ) verify_rule
```

Figure 16: Available bbatch commands.

8 Troubleshooting

Below are listed the most common errors encountered and the associated workarounds.

8.1 The bbatch process could not be started

It happens when the directory containing the description files of the projects cannot be accessed, often because it as installed by the administrator in a directory only accessible to him. This directory in appears in the directory where Atelier B is installed. Serach for etc/AtelierB file. You should find at the end of this file a line similar to this `ATB*ATB*Atelier_Database_Directory: C:\Users\<user>\AtelierB_Data\AtelierB_free_24.04\press\bdb`

8.2 GDP is missing

The graph of the project cannot be calculated because your B components contains many errors preventing a successful parsing and the determination of a correct project architecture with the SEES and IMPORTS links.

9 Filesystem

Atelier B comes with an architecture of files with given locations and formats, that the user needs to know and understand before making any significant modification. There are three locations to consider:

- the installation directory
- the projects database directory
- the workspaces

9.1 The Installation Directory

It contains several sub-directories with binaries, librairies, dictionnaries and several other content. Two important resources need to be mentioned:

- etc/AtelierB: this file contains the declaration of several resources used by Atelier B. It defines in particular the location of the Projects Database Directory, which has to be reacheable (read/write access) all the time by the current user (unless no project can be created).
- share/plugins: this directory contains the plugins (if any) used to extends the Atelier B. Plugins interface is defined with .etool files.

9.2 The Projects Database Directory

Coressponding to the workspace "local", it contains the description of the projects created by the current user if no other workspace is selected. Each project is defined with a .desc file, created and managed by Atelier B. A .desc file contains the location of bdp and lang directories (respectively the directories containing the intermediate B files and the code generated), the name of the owner, and the B libraries used by the project.

When installing Atelier B for the first time, you are asked to enter the path to this directory. Ensure that you have continuous read/write access to this directory.

9.3 The Workspaces

A workspace is created by the user. For each project created in it, it contains a .desc file and a home directory containing bdp and lang directories.

10 AtelierB Resources

AtelierB file contains resources that are defined at installation and project level. This section defines all the resources that can be valued in an AtelierB file. Resources are grouped according to the tool they are related to. They are listed with their default value if any.

10.1 Typechecker

- `ATB*TC*Generate_B0_Conditions`: generation of B0 preconditions (Boolean, default: TRUE)
- `ATB*TC*Enable_Extended_Sees`: enables the use of extended SEES (Boolean, default: FALSE)
- `ATB*TC*Check_B0_Arrays`: B0 verification of arrays (Boolean, default: FALSE)
- `ATB*TC*Enable_Local_Operations`: Enables LOCAL_OPERATIONS (Boolean, default: TRUE)
- `ATB*TC*Allow_Abstract_Constants_In_Values`: Enables constants in VALUES (Boolean, default: FALSE)
- `ATB*TC*EventB_DeadlockFreeness`: generation of Event-B deadlockfreeness proof obligations (Boolean, default: FALSE)

10.2 Proof Obligation Generator

- `ATB*ATB*Proof_Obligations_Generator_NG`: use the non-legacy proof obligation generator (Boolean, default: TRUE)
- `ATB*POG*Generate_EventB_Non_Divergence_PO`: generation of Event-B variant proof obligations. The clause VARIANT is mandatory (Boolean, default: FALSE)
- `ATB*POG*Generate_EventB_Coverage_PO`: generation of Event-B coverage proof obligations (Boolean, default: FALSE)
- `ATB*POG*Generate_EventB_Feasibility_PO`: generation of Event-B feasibility proof obligations (Boolean, default: FALSE)
- `ATB*POG*Generate_EventB_Exclusivity_PO`: generation of Event-B exclusivity proof obligations (Boolean, default: FALSE)
- `ATB*POG*Enable_Structural_Refinement`: enables structural refinement (Boolean, default: FALSE)
- `ATB*POG*Generate_Obvious_PO`: generation of obvious proof obligations (Boolean, default: FALSE)
- `ATB*POG*Generate_PO_When_Extended_Sees_Is_Enabled`: generation of extended SEES proof obligations (Boolean, default: TRUE)
- `ATB*POG*Generate_EventB_Non_Divergence_PO`: generation of Event-N non divergence proof obligations (Boolean, default: FALSE)

- `ATB*POG*Generate_Correctly_ASSERT_Refinement`: generation of ASSERT refinement proof obligations (Boolean, default: FALSE)
- `ATB*POG*Strict_WD_PO`: generation of welldefinedness proof obligations (Boolean, default: FALSE)
- `ATB*POG*Generate_WD_PO`: generation of welldefinedness proof obligations (Boolean, default: FALSE)

10.3 Prover

- `ATB*PR*Time_Out_Auto`: prover time out in seconds (Integer)
- `ATB*PR*Time_Out`: prover time out in seconds(Integer)
- `ATB*PR*Keep_Non_Simplified_Hypothesis`: the prover keeps non simplified hypothesis (Boolean)
- `ATB*PR*Enable_WD_Command`: enables welldefinedness checks for interactive prover commands (Boolean)
- `ATB*PR*Enable_TC_Command`: enables type check for interactive prover commands (Boolean)
- `ATB*PR*Use_Rule_Package`: additional rule package(s) to use (List of identifiers, Default: ?)
- `ATB*PR*BRPR_Path`: reusable proof rule base path (String)
- `ATB*PR*Max_Number_Of_Universal_Hypothesis_Instantiation`: maximum number of universal Hhpothesis instantiation (Integer)
- `ATB*PR*Restrained_Forward_Equalities`: retrains the forward equalities mechanism (Boolean, default: FALSE)
- `ATB*PR*Pr_3_6_compatibility`: disables prover 3.7 extensions (Boolean, default: FALSE)
- `ATB*PR*Trace_User_Rules`: disables trace of user rules (Boolean, default: FALSE)
- `ATB*PR*Pr_3_7_compatibility`: disables prover 4.0 extensions (Boolean, default: FALSE)
- `ATB*PR*Mono_Character_Identifier_Is_A_Joker_In_Proof_Command`: a single character identifier is a joker in pmi and user pass (Boolean, default: TRUE)
- `ATB*PR*ProB_Path`: path to ProB model checker (String)
- `ATB*PR*ProB_Time_Out`: ProB timetout in seconds (Integer)

11 Lexicon

bdb See Workspace.

bdp The **bdp** of a project is a directory that stores all the internal files necessary for the management of the project.

lang The **bdp** of a project is a directory that stores all the files resulting from the translation to a programming language of B components.

pmm A file containing proof rules and proof tactics for a B component.

project A **project** is a collection of B components and PMM files. It is referenced from a workspace thanks to a `desc` file. It contains a data base directory (bdp) and a language directory (lang).

workspace A **workspace** is a collection of projects. It is a directory containing a collection of `desc` files. Atelier B stores the workspaces in its settings (in registry on Windows, under the $HOME/.config directory on Linux and macOS).