Instruction For The Use Of (.v)Files

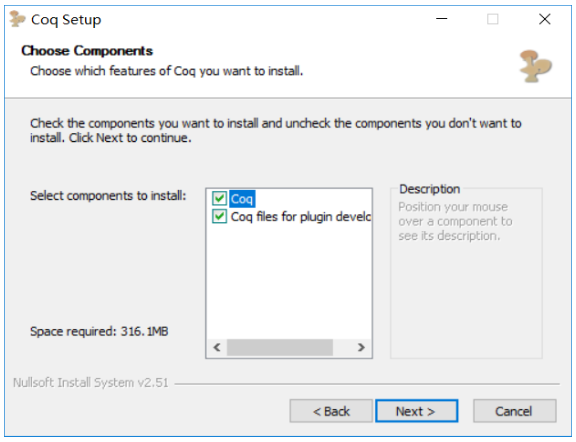
Coq is an open-source formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. [cite]

This instruction aims to introduce the most basic usage for running code in CoqIDE, helping reviewers and readers to verify the integrity and correctness of our Coq code. For other more details of the usage of Coq, see https://coq.inria.fr/documentation.

1. Install the CoqIDE (version 8.13.2).

The formalization is implemented in the CoqIDE (version 8.13.2) for 64 bit Windows. Here is the download link.

Open the downloaded installer and proceed with the installation.



Coq Setup

Choose Components

Choose which features of Coq you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install: [✓ Coq] [✓ Coq files for plugin develc]

Description: Position your mouse over a component to see its description.

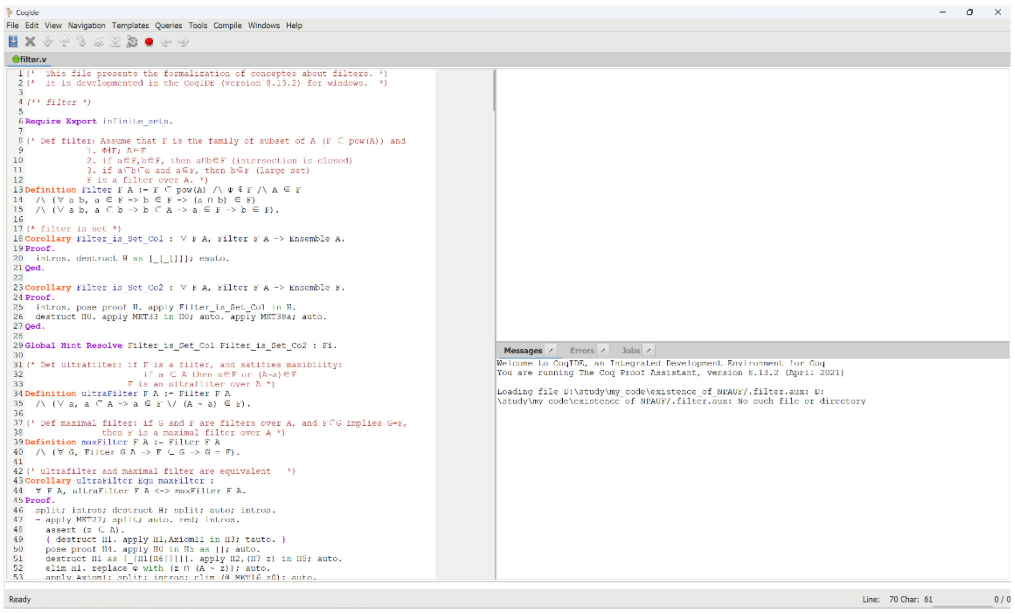
Space required: 316.1MB

Nullsoft Install System v2.51

Try to ensure that the installation path is in English. After installation, (.v)files can be opened and edited by CoqIDE.

2. Interface of CoqIDE.

The CoqIDE interface consists mainly of three windows:


CoqIde
File Edit View Navigation Templates Queries Tools Compile Windows Help
filter.v
Messages
Errors
Jobs
Ready
Line: 70 Char: 61
0 / 0

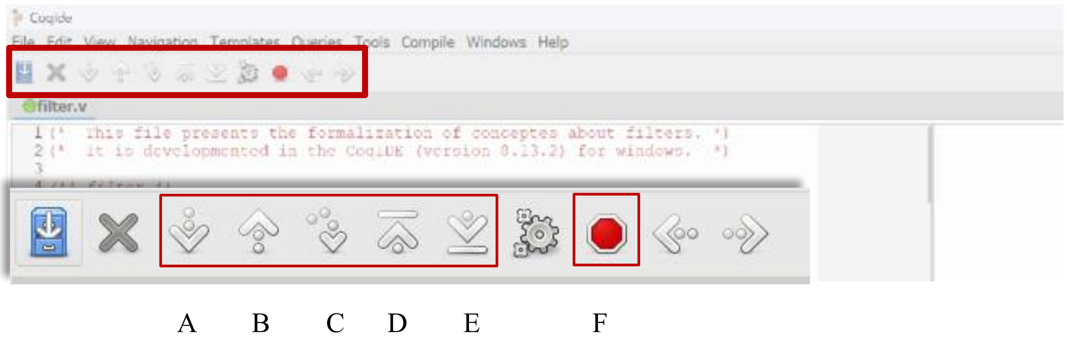
```
(*  This file presents the formalization of conceptes about filters. *)
(*  It is developmented in the CoqIDE (version 8.13.2) for windows.  *)

(** filter *)

Require Export infinite_sets.

(* Def filter: Assume that F is the family of subset of A (F ⊂ pow(A)) and
              1. Φ∉F; A∈F
              2. if a∈F,b∈F, then a∩b∈F (intersection is closed)
              3. if a⊂b⊂ω and a∈F, then b∈F (large set)
              F is a filter over A. *)
Definition Filter F A := F ⊂ pow(A) /\ Φ ∉ F /\ A ∈ F
  /\ (∀ a b, a ∈ F -> b ∈ F -> (a ∩ b) ∈ F)
  /\ (∀ a b, a ⊂ b -> b ⊂ A -> a ∈ F -> b ∈ F).

(* filter is set *)
Corollary Filter_is_Set_Col : ∀ F A, Filter F A -> Ensemble A.
Proof.
  intros. destruct H as [_[_[]]]; eauto.
Qed.

Corollary Filter_is_Set_Co2 : ∀ F A, Filter F A -> Ensemble F.
Proof.
  intros. pose proof H. apply Filter_is_Set_Col in H.
  destruct H0. apply MKT33 in H0; auto. apply MKT38a; auto.
Qed.

Global Hint Resolve Filter_is_Set_Col Filter_is_Set_Co2 : Fi.

(* Def ultrafilter: if F is a filter, and satifies maxibility:
                             if a ⊂ A then a∈F or (A~a)∈F
                         F is an ultrafilter over A *)
Definition ultraFilter F A := Filter F A
  /\ (∀ a, a ⊂ A -> a ∈ F \/ (A ~ a) ∈ F).

(* Def maximal filter: if G and F are filters over A, and F⊂G implies G=F,
                then F is a maximal filter over A *)
Definition maxFilter F A := Filter F A
  /\ (∀ G, Filter G A -> F ⊂ G -> G = F).

(* ultrafilter and maximal filter are equivalent   *)
Corollary ultraFilter_Equ_maxFilter :
  ∀ F A, ultraFilter F A <-> maxFilter F A.
Proof.
  split; intros; destruct H; split; auto; intros.
  - apply MKT27; split; auto. red; intros.
    assert (z ⊂ A).
    { destruct H1. apply H1,AxiomII in H3; tauto. }
    pose proof H4. apply H0 in H5 as []; auto.
    destruct H1 as [_[H1[H6[]]]]. apply H2,(H7 z) in H5; auto.
    elim H1. replace Φ with (z ∩ (A ~ z)); auto.
    apply AxiomI; split; intros; elim (@ MKT16 z0); auto.
```

```
Welcome to CoqIDE, an Integrated Development Environment for Coq
You are running The Coq Proof Assistant, version 8.13.2 (April 2021)

Loading file D:\study\my_code\existence_of_NPAUF/.filter.aux: D:
\study\my code\existence of NPAUF/.filter.aux: No such file or directory
```

The left area is for code writing; the top right window is for interaction, where you can observe the validation process of proofs while running code, and then continue writing and modifying proof code based on this feedback; the bottom right window is for information display, which can show error messages and print definitions and theorem contents that have been formalized.

3. The execution of code.

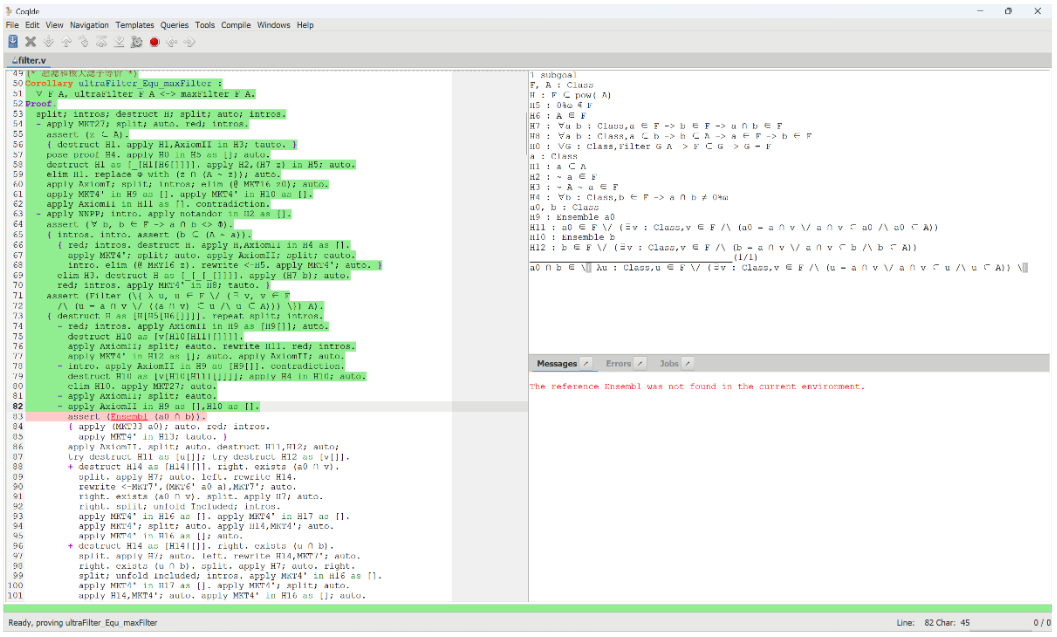
Coq code consists of various internal commands, which can be executed by clicking on the buttons indicated in the figure below.


Coqide
File Edit View Navigation Templates Queries Tools Compile Windows Help
filter.v
A
B
C
D
E
F

```
(* This file presents the formalization of conceptes about filters. *)
(* It is developmentcted in the CoqIDE (version 8.13.2) for windows.  *)
```

A: forward one command (ctrl + down)
B: backward one command (ctrl + up)
C: run to cursor (ctrl + right)
D: restart Coq
E: run to the end
F: interrupt the execution

The code executed successfully will be displayed in green, and the proof process will be presented in the interactive area. If some command cannot be executed, it will be displayed in red and the error massage be displayed in the bottom right window.


CoqIde
File Edit View Navigation Templates Queries Tools Compile Windows Help
filter.v
Messages
Errors
Jobs

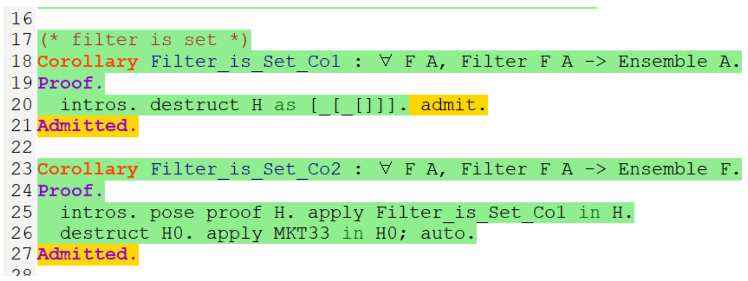
```
(* 超滤和极大滤子等价 *)
Corollary ultraFilter_Equ_maxFilter :
  ∀ F A, ultraFilter F A <-> maxFilter F A.
Proof.
  split; intros; destruct H; split; auto; intros.
  - apply MKT27; split; auto. red; intros.
    assert (z ⊂ A).
    { destruct H1. apply H1,AxiomII in H3; tauto. }
    pose proof H4. apply H0 in H5 as []; auto.
    destruct H1 as [_[H1[H6[]]]]. apply H2,(H7 z) in H5; auto.
    elim H1. replace Φ with (z ∩ (A ~ z)); auto.
    apply AxiomI; split; intros; elim (@ MKT16 z0); auto.
    apply MKT4' in H9 as []. apply MKT4' in H10 as [].
    apply AxiomII in H11 as []. contradiction.
  - apply NNPP; intro. apply notandor in H2 as [].
    assert (∀ b, b ∈ F -> a ∩ b <> Φ).
    { intros. intro. assert (b ⊂ (A ~ a)).
      { red; intros. destruct H. apply H,AxiomII in H4 as [].
        apply MKT4'; split; auto. apply AxiomII; split; eauto.
        intro. elim (@ MKT16 z). rewrite <-H5. apply MKT4'; auto. }
      elim H3. destruct H as [_[_[_[]]]]. apply (H7 b); auto.
      red; intros. apply MKT4' in H8; tauto. }
    assert (Filter (\{ λ u, u ∈ F \/ (∃ v, v ∈ F
      /\ (u = a ∩ v \/ ((a ∩ v) ⊂ u /\ u ⊂ A))) \}) A).
    { destruct H as [H[H5[H6[]]]]. repeat split; intros.
      - red; intros. apply AxiomII in H9 as [H9[]]; auto.
        destruct H10 as [v[H10[H11|[]]]].
        apply AxiomII; split; eauto. rewrite H11. red; intros.
        apply MKT4' in H12 as []; auto. apply AxiomII; auto.
      - intro. apply AxiomII in H9 as [H9[]]. contradiction.
        destruct H10 as [v[H10[H11|[]]]]; apply H4 in H10; auto.
        elim H10. apply MKT27; auto.
      - apply AxiomII; split; eauto.
      - apply AxiomII in H9 as [],H10 as [].
        assert (Ensembl (a0 ∩ b)).
        { apply (MKT33 a0); auto. red; intros.
          apply MKT4' in H13; tauto. }
        apply AxiomII. split; auto. destruct H11,H12; auto;
        try destruct H11 as [u[]]; try destruct H12 as [v[]].
        + destruct H14 as [H14|[]]. right. exists (a0 ∩ v).
          split. apply H7; auto. left. rewrite H14.
          rewrite <-MKT7',(MKT6' a0 a),MKT7'; auto.
          right. exists (a0 ∩ v). split. apply H7; auto.
          right. split; unfold Included; intros.
          apply MKT4' in H16 as []. apply MKT4' in H17 as [].
          apply MKT4'; split; auto. apply H14,MKT4'; auto.
          apply MKT4' in H16 as []; auto.
        + destruct H14 as [H14|[]]. right. exists (u ∩ b).
          split. apply H7; auto. left. rewrite H14,MKT7'; auto.
          right. exists (u ∩ b). split. apply H7; auto. right.
          split; unfold Included; intros. apply MKT4' in H16 as [].
          apply MKT4' in H17 as []. apply MKT4'; split; auto.
          apply H14,MKT4'; auto. apply MKT4' in H16 as []; auto.
```

```
1 subgoal
F, A : Class
H : F ⊂ pow( A)
H5 : 0%ω ∉ F
H6 : A ∈ F
H7 : ∀a b : Class,a ∈ F -> b ∈ F -> a ∩ b ∈ F
H8 : ∀a b : Class,a ⊂ b -> b ⊂ A -> a ∈ F -> b ∈ F
H0 : ∀G : Class,Filter G A -> F ⊂ G -> G = F
a : Class
H1 : a ⊂ A
H2 : ~ a ∈ F
H3 : ~ A ~ a ∈ F
H4 : ∀b : Class,b ∈ F -> a ∩ b ≠ 0%ω
a0, b : Class
H9 : Ensemble a0
H11 : a0 ∈ F \/ (∃v : Class,v ∈ F /\ (a0 = a ∩ v \/ a ∩ v ⊂ a0 /\ a0 ⊂ A))
H10 : Ensemble b
H12 : b ∈ F \/ (∃v : Class,v ∈ F /\ (b = a ∩ v \/ a ∩ v ⊂ b /\ b ⊂ A))
______________________________________(1/1)
a0 ∩ b ∈ \{ λu : Class,u ∈ F \/ (∃v : Class,v ∈ F /\ (u = a ∩ v \/ a ∩ v ⊂ u /\ u ⊂ A)) \}
```

```
The reference Ensembl was not found in the current environment.
```

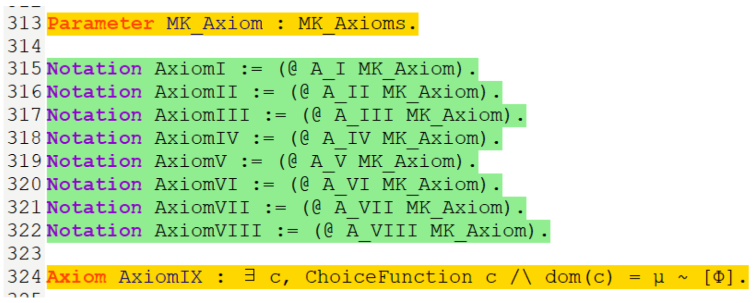
```
Ready, proving ultraFilter_Equ_maxFilter          Line:  82 Char:  45          0 / 0
```

Especially, some code can be executed successfully but will be displayed in yellow. This is divided into two cases: one is that the command “admit” or “Admitted” is used:



```
(* filter is set *)
Corollary Filter_is_Set_Co1 : ∀ F A, Filter F A -> Ensemble A.
Proof.
  intros. destruct H as [_[_[]]]. admit.
Admitted.

Corollary Filter_is_Set_Co2 : ∀ F A, Filter F A -> Ensemble F.
Proof.
  intros. pose proof H. apply Filter_is_Set_Co1 in H.
  destruct H0. apply MKT33 in H0; auto.
Admitted.
```

the other is that the code is declared by the command “Parameter” or “Axiom”:



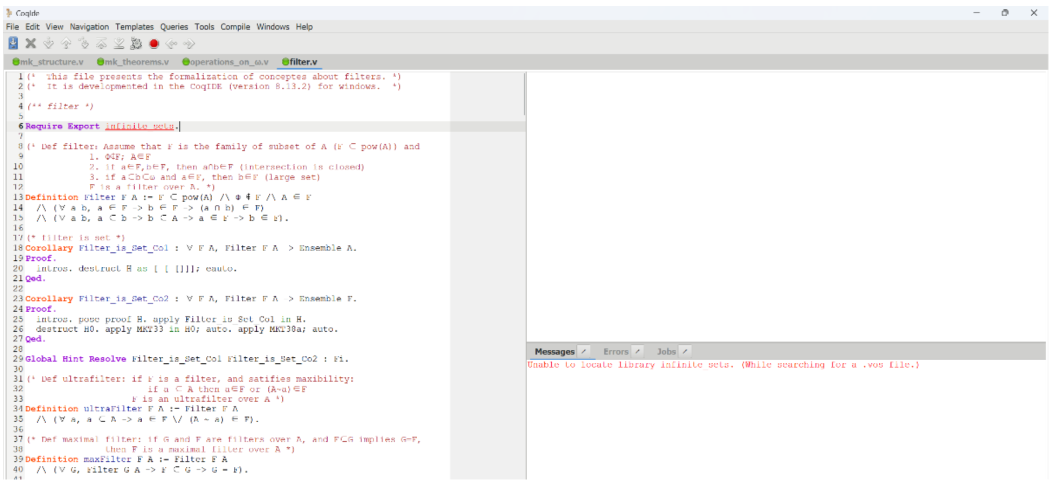
```
Parameter MK_Axiom : MK_Axioms.

Notation AxiomI := (@ A_I MK_Axiom).
Notation AxiomII := (@ A_II MK_Axiom).
Notation AxiomIII := (@ A_III MK_Axiom).
Notation AxiomIV := (@ A_IV MK_Axiom).
Notation AxiomV := (@ A_V MK_Axiom).
Notation AxiomVI := (@ A_VI MK_Axiom).
Notation AxiomVII := (@ A_VII MK_Axiom).
Notation AxiomVIII := (@ A_VIII MK_Axiom).

Axiom AxiomIX : ∃ c, ChoiceFunction c /\ dom(c) = μ ~ [Φ].
```

For the first case, it is not permitted because the command “admit” and “Admitted” indicate that the proof is not finished, and the display in yellow is to remind developers of completing it.

The second case is conditionally permitted. The command “Parameter” and “Axiom” are utilized to describe propositions that are temporarily unprovable or objects that are temporarily unconstructible, such as the description of axioms, which is common in mathematics. Therefore, these two commands are not to be used casually; they are typically reserved for describing widely recognized or rigorously proven consistent axioms and other objects.

4. The compilation of code.

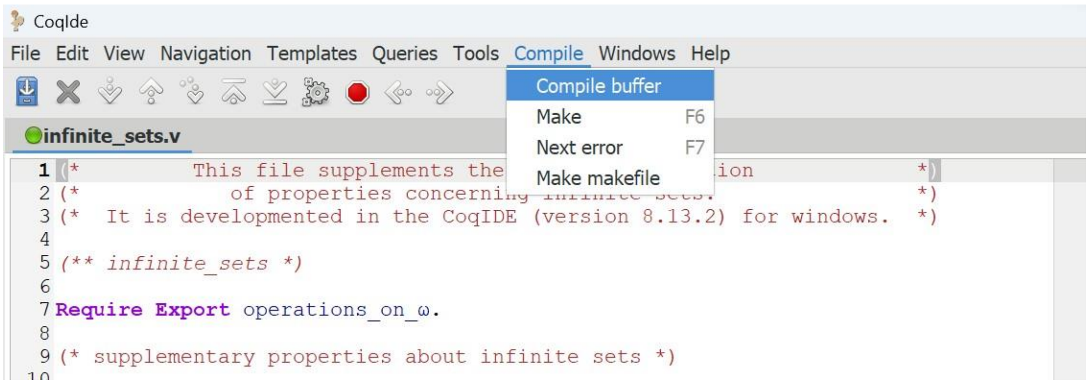
In general, a (.v)file may depend on other (.v)files (usually read with the “Export” or “Import” command). In this case, the dependent files need to be compiled first, otherwise, it cannot run. For example, in the figure below, the dependent file is not compiled, errors occur when running.



```
(*  This file presents the formalization of conceptes about filters. *)
(*  It is developmented in the CoqIDE (version 8.13.2) for windows.  *)

(** filter *)

Require Export infinite_sets.

(* Def filter: Assume that F is the family of subset of A (F ⊂ pow(A)) and
              1. Φ∉F; A∈F
              2. if a∈F,b∈F, then a∩b∈F (intersection is closed)
              3. if a⊂b⊂ω and a∈F, then b∈F (large set)
              F is a filter over A. *)
Definition Filter F A := F ⊂ pow(A) /\ Φ ∉ F /\ A ∈ F
  /\ (∀ a b, a ∈ F -> b ∈ F -> (a ∩ b) ∈ F)
  /\ (∀ a b, a ⊂ b -> b ⊂ A -> a ∈ F -> b ∈ F).

(* filter is set *)
Corollary Filter_is_Set_Co1 : ∀ F A, Filter F A -> Ensemble A.
Proof.
  intros. destruct H as [_[_[]]]; eauto.
Qed.

Corollary Filter_is_Set_Co2 : ∀ F A, Filter F A -> Ensemble F.
Proof.
  intros. pose proof H. apply Filter_is_Set_Co1 in H.
  destruct H0. apply MKT33 in H0; auto. apply MKT38a; auto.
Qed.

Global Hint Resolve Filter_is_Set_Co1 Filter_is_Set_Co2 : Fi.

(* Def ultrafilter: if F is a filter, and satifies maxibility:
                      if a ⊂ A then a∈F or (A~a)∈F
                    F is an ultrafilter over A *)
Definition ultraFilter F A := Filter F A
  /\ (∀ a, a ⊂ A -> a ∈ F \/ (A ~ a) ∈ F).

(* Def maximal filter: if G and F are filters over A, and F⊆G implies G=F,
                  then F is a maximal filter over A *)
Definition maxFilter F A := Filter F A
  /\ (∀ G, Filter G A -> F ⊂ G -> G = F).
```

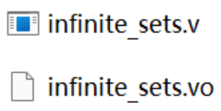
```
Unable to locate library infinite_sets. (While searching for a .vos file.)
```

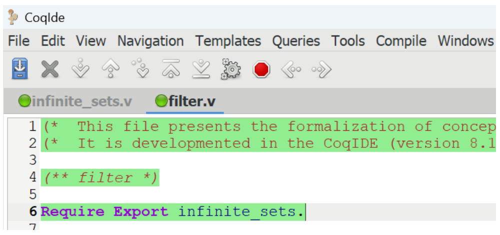
To compile the dependent file, click on “Compile” and “Compile buffer” in the interface of the file to be compiled as shown in the figure below.


CoqIde
File Edit View Navigation Templates Queries Tools Compile Windows Help
Compile buffer
Make F6
Next error F7
Make makefile
infinite_sets.v

```
(*          This file supplements the                   ion        *)
(*             of properties concerning                             *)
(*  It is developmented in the CoqIDE (version 8.13.2) for windows.  *)

(** infinite_sets *)

Require Export operations_on_ω.

(* supplementary properties about infinite sets *)
```

Make sure that the code to be compiled can be successfully executed in green or yellow, otherwise it cannot be compiled. If compilation succeeds, a corresponding (.vo)file will be generated of the same name and path as the compiled (.v)file. After this, the dependent file can be read by other files:




CoqIde
File Edit View Navigation Templates Queries Tools Compile Windows
infinite_sets.v
filter.v

```
(*  This file presents the formalization of concep
(*  It is developmented in the CoqIDE (version 8.1

(** filter *)

Require Export infinite_sets.
```

Note that the dependent file and the other files that read it should be in the same path.

Run all files according to the instructions above, until all the code runs successfully and turns green (or yellow, which should be carefully checked). This indicates that all formal definitions and theorems are correctly described and verified.