

Project 2007

March 10, 2019



# Contents

<b>I</b>	<b>Generalities</b>	<b>5</b>
<b>1</b>	<b>My first command : set editing</b>	<b>7</b>
1.1	Introduction . . . . .	7
1.2	Algorithms/Photogrammetry . . . . .	7
1.3	User's side(1) . . . . .	7
1.3.1	Basic notion . . . . .	7
1.3.2	Getting help . . . . .	8
1.3.3	basic usage . . . . .	8
1.3.3.1	Exercices . . . . .	9
1.3.4	Optional paramaters . . . . .	9
1.3.4.1	Out paramater . . . . .	9
1.3.4.2	Show paramater . . . . .	10
1.3.5	More help . . . . .	10
1.4	User's side-2, global parameter . . . . .	11
1.4.1	Fixing project directory <code>DirProj</code> . . . . .	11
1.4.2	Filtering by interval <code>FFI0</code> , <code>FFI1</code> . . . . .	11
1.4.3	Redirecting message with <code>StdOut</code> . . . . .	12
1.4.3.1	Exercices . . . . .	12
1.4.4	Fixing MicMac version for export <code>NumVOut</code> . . . . .	12
1.4.5	Predefined semantics . . . . .	12
1.4.5.1	Generalities . . . . .	12
1.4.5.2	Main pattern image [MPI] . . . . .	13
1.4.5.3	File of Directory Project [FDP] . . . . .	13
1.5	User's side-3, most frequent error . . . . .	13
1.5.1	Generality . . . . .	13
1.5.2	Error <code>BadBool</code> . . . . .	13
1.5.3	Error <code>BadOptP</code> . . . . .	14
1.5.4	Error <code>MultOptP</code> . . . . .	14
1.5.5	Error <code>OpenFile</code> . . . . .	14
1.5.6	Error <code>InsufP</code> . . . . .	14
1.5.7	Error <code>BadEnum</code> . . . . .	14
1.5.8	Error <code>FileSetN</code> . . . . .	14
1.5.9	Error <code>IntWithoutS</code> . . . . .	14
1.6	Programmer's side, adding a new command (1) . . . . .	14
1.6.1	Heriting from <code>cMMVIIAppli</code> . . . . .	15
1.6.2	Link between name an class . . . . .	15
1.6.3	Specifying paramaters . . . . .	15
1.6.4	Standard access paramaters . . . . .	16
1.6.4.1	Reading set of name from file with <code>SetNameFromString</code> . . . . .	16
1.6.4.2	Main sets with <code>MainSetk</code> . . . . .	16

<b>2</b>	<b>Programming organisation, style ...</b>	<b>17</b>
2.1	Naming convention . . . . .	17
2.2	Never use <code>std::cout</code> , <code>printf</code> . . . . .	17
2.3	Encapsulation of boost, stl .. . . .	17
2.4	Error handling . . . . .	17
2.5	Memory check . . . . .	17
2.6	Serialization . . . . .	17
2.7	Shared pointer . . . . .	17
2.8	Random number . . . . .	17
2.9	Enum to string . . . . .	17
<b>3</b>	<b>Project management command</b>	<b>19</b>
3.1	Help command . . . . .	19
<b>II</b>	<b>Methodologies</b>	<b>21</b>
<b>4</b>	<b>The "Aime" methods for Tie Points computation</b>	<b>23</b>
4.0.1	Indexing problem . . . . .	23
<b>III</b>	<b>Reference documentation</b>	<b>25</b>
<b>IV</b>	<b>Annexes</b>	<b>27</b>
<b>A</b>	<b>Référence bibliographique</b>	<b>29</b>

Part I

**Generalities**



# Chapter 1

## My first command : set editing

### 1.1 Introduction

This chapter presents the first commands of MMVII. It uses a plan that will be almost systematic in many other chapter :

- a section relative to algorithmic and photogrammetric aspect of the chapter, generally this section may exist <sup>1</sup> almost totally independantly of MMVII, but it is pre-requisite as there is obviously no interest to know the command and the code if the fundamentalls are not understood;
- a user's guide section, relative to MMVII at the users level, including the syntax of the command;
- one or more programmers section, relative to **C++** code implemanting the command, it will be a presentation of general organisation <sup>2</sup>, as the detail are to be found in *doxygen* pages;

This chapter will be a bit specific as the part of user's guide and programming will be much more important than other for a single command, as many concept common to all command will be explained here, conversely the algorithmic part will be very short.

### 1.2 Algorithms/Photogrammetry

The command we learn is a command usefull for editing a set of file. Almost all command of MMVII require as parameter one or more set of file (i.e. the subset of images that we are considering for a given computation). For single case, this set of file can be simply specified by a regular expression : for example `"*.JPG"` to specify all the file with a `JPG` extension.

However for more complex case we may want to :

- create a set from a single pattern;
- add or substract an interval, a pattern ...
- memorize the result and reuse it.

This is what does the `EditSet` command, piece by piece create a `XML` file that memorize a "complex" set of file that can be used instead of a pattern.

### 1.3 User's side(1)

#### 1.3.1 Basic notion

MMVII is a command line programm. There is unique programm which name is `MMVII`. Any command, `OneCmd`, of MMVII will be called with the syntax `MMVII OneCmd Args` where `Args` are the arguments of the command. To know what are the existing command there is two way :

- a basic one just enter `MMVII`;
- a more sophisticated one , to be written, `MMVII Help t` described in 3.1;

For the basic one we get:

---

1. i.e. may be of interest for the reader, hopefully  
2. as link between concept and classes

MMVII

...

Bench => This command execute (many) self verification on MicMac-V2 behaviour

Cpp11 => This command execute some test for to check my understanding of C++11

TBS => This command execute some experiments en boost serrialization

MPDTest => This used a an entry point to all quick and dirty test by MPD ...

EditSet => This command is used to edit set of file

EditRel => This command is used to edit set of pairs of files

...

We get the list of all command and short commentary on the service given by the command.

### 1.3.2 Getting help

Very currently, user will know what the command does, but will not remember the exact syntax. The `help` key word can be used at any position for requiring this information, for example :

MMVII EditSet help

```
*****
*   Help project 2007/MMVII           *
*****
```

For command : EditSet

=> This command is used to edit set of file

=> Srce code entry in : ../MMVII/src/Appli/cMMVII\_CalcSet.cpp

== Mandatory unnamed args : ==

\* string [FDP] :: Full Name of Xml in/out

\* string :: Operator in (+= \*= -= = =0)

\* string [MPI0] :: Pattern or Xml for modifying

== Optional named args : ==

\* [Name=Show] int :: Show detail of set before/after , (def) 0->none, (1) modif, (2) all

\* [Name=Out] string :: Destination, def=Input, no save for NONE

We get three part :

- first part give the short comment, and the name of the C++ file where the entry point of the command is implemented (may be of interest to programmers);
- second part contains the description of mandatory args, we see that here we have three mandatory args; for each args is indicated the type (here all strings), and after ::, the semantic of the parameter; sometime it is inserted inside square bracket (like [FDP]) some "predefined semantics" that will be described later ( 1.4.5);
- third part contains the description of optional args, as for mandatory args, the type and a short command is given, before this is added the name the optional parameter in the form [Name=TheName];

As said before, `help` can appear at any position after `OneCmd`, this can be usefull when one has begin to edit a command, and dont want to loose it, for example with parameter of next section, the following line is perfectly valide to obtain help about `EditSet` :

MMVII EditSet File.xml = "F[0-3].txt" help

### 1.3.3 basic usage

For example, if we go in the folder `MMVII-MainFolder/MMVII-TestDir/Input/Files`, we can test :

MMVII EditSet File.xml = "F[0-3].txt"



Here we have used only the mandatory paramaters. As there is no naming for these parameters, the order is used to make the correspondance between parameters and value, so here :

- `File.xml` correspond to first parameter described as "Full Name of Xml in/out";
- `=` correspond to second paramater, described as `Operator ...`;
- `"F[0-3].txt"` correspond to third paramater, described as `Pattern ...`;

Some comment on the effect of this parameter :

- `File.xml` is the name of the XML file that contain the initial list of name, it's perfectly acceptable that this file does not exist, in this case an empty list is created;
- `=` correspond to second paramater, is describe the operator that will be used to modify the file with the value `S3` of third parameter, its value must belong to an enumerated list with the following meaning

- `=` , `S3` overwrite `File.xml` ;
- `+=` , `S3` is added to `File.xml` ;
- `-=` , `S3` is substracted from `File.xml`
- `*=` , `File.xml` is the intersection of `S3` and its previous value;
- `=0` , `File.xml` is empty, whatever may be in `S3` ;

- `"F[0-3].txt"` correspond to third paramater, described as `Pattern ...`;

We can now inspect the file `File.xml` which contains the name of the files *present in the folder* and matching the regular expression `"F[0-3].txt"`:

```
cat File.xml
<?xml version="1.0" encoding="ISO8859-1" standalone="yes" ?>
<MMVII_Serialization>
  <SetOfName>
    <Nb>4</Nb>
    <el>F0.txt</el>
    <el>F1.txt</el>
    <el>F2.txt</el>
    <el>F3.txt</el>
  </SetOfName>
</MMVII_Serialization>
```

As always when a regular expression is used to specify set of file, it is understood as a filter on existing file. So if one had used `"F([0-3] | [a-z]).txt"`, given the file present in `MMVII-MainFolder/MMVII-TestDir/Input/Fi` we would have obtained exactly the same result.

### 1.3.3.1 Exercices

Try the following command and inspect the result , after each :

```
MMVII EditSet File.xml = "F[0-3].txt"
MMVII EditSet File.xml += "F[7-9].txt"
MMVII EditSet File.xml -= "F8.txt"
MMVII EditSet File.xml *= "F[02468].txt"
MMVII EditSet File.xml =0 ".*"
```

## 1.3.4 Optional paramaters

### 1.3.4.1 Out paramater

Optional parameter are given after the mandary one in a list of string `Name=Value`. For example until now we have use the file `File.xml` both as input and output, but sometime we don't want to modify the input file, we can the use the optionnal `Out` parameter. For example if we enter :

```
MMVII EditSet File.xml = "F[0-3].txt"
MMVII EditSet File.xml += "F[7-9].txt" Out=File2.xml
```

After first line `File.xml` contains t4 names. After second line, the `File.xml` is unchanged while `File2.xml` contains 7 names.

An interesting option, for this commands as each time a pattern is expected, is that if the file is XML file, created by MMVII and with main tag `<SetOfName>`, then name used will not be the pattern itself but the name contained in the file, for example :

```
MMVII EditSet File1.xml = "F[0-3].txt"
MMVII EditSet File2.xml = "F[7-9].txt"
MMVII EditSet File1.xml += File2.xml Out=File3.xml
MMVII EditSet File3.xml += File2.xml Out=File4.xml
```

After this command `File3.xml` contains the sum of `File1.xml` and `File2.xml`, here 7 name. All the operation are set operation, in the mathematical sense, so there is no duplicate, dans `File4` contain still 7 names.

#### 1.3.4.2 Show paramater

The Show allow to visualize the result of the operation.

```
MMVII EditSet File.xml =0 "."
MMVII EditSet File.xml = "F[0-4].txt"
MMVII EditSet File.xml += "F[0-6].txt" Show=1
-+ F5.txt
-+ F6.txt
MMVII EditSet File.xml *= "F[02468].txt" Show=2
++ F0.txt
++ F2.txt
++ F4.txt
++ F6.txt
+- F1.txt
+- F3.txt
+- F5.txt
```

The third command use the parameter `Show`, as the value is 1, only the modification are shown :

— `-+ F5.txt` means that the file was inially absent (−) and is present at end (+)

It is also possible to show all the result, including the names that present before and after the modification :

— `++ F0.txt` : means that the file is present before and after the operation;

— `+- F1.txt` : means that the file is present before and absent after the operation;

#### 1.3.5 More help

In MMVIIthere exists many optional parameter. There are not shown by default in the help mode, but it is possible to show :

— the standard common parameter by setting `Help` instead of `help`

— all the common parameter, including the *internal* common parameter by setting `HELP` instead of `help`; the internal parameter are used by MMVIIto communicate information to sub-process when MMVIIcalls itself; for example the parameter `LevCall` allow MMVIIto know if it was called by the user or by MMVIIand to which level of imbrication; obviously it is generally a bad idea to fix yourself the internal parameter;

Here is an example with `EditSet` :

```
MMVII EditSet File.xml *= "F[02468].txt" Show=2 HELP
...
* [Name=Out] string :: Destination, def=Input, no save for NONE
* [Name=FFI0] string [FFI0] :: File Filter Interval, Main Set   ### COMMON
* [Name=FFI1] string [FFI1] :: File Filter Interval, Second Set  ### COMMON
* [Name=NumVOut] int :: Num version for output format (1 or 2)  ### COMMON
```

```

* [Name=DirProj] string [DP] :: Project Directory    ### COMMON
* [Name=StdOut] string :: Redirection of Ouput (+File for add,NONEfor no out)    ### COMMON
...
* [Name=LevCall] int :: Internal : Don't Use !!    ### INTERNAL
* [Name=ShowAll] bool :: Internal : Don't Use !!    ### INTERNAL
...

```

As some command have many option, it possible to filter the optionnal parameter using a regular expression , with a syntax `help=expr` (or `Help` or `HELP`), for example :

```

MMVII EditSet File.xml *= "F[02468].txt" Show=2 HELP=F.*
...
== Optional named args : ==
* [Name=FFI0] string :: File Filter Interval, Main Set    ### COMMON
* [Name=FFI1] string :: File Filter Interval, Second Set    ### COMMON

```

## 1.4 User's side-2, global parameter

### 1.4.1 Fixing project directory DirProj

In MMVII the notion of project is closely related to the folder where are stored a given set of data, basically one can consider for universall the rule "one project/one folder". MMVII uses the following rule to determine the directory of project :

- many command have a parameter that fix the project folder, for example with `EditSet` the first parameter fix the project directory, they are indicated by `[FDP]` (see 1.4.5);
- when there is no command to fix the folder, by default MMVII fix the project folder to `"/"`;
- it is also possible to fix this directory with the optionnal parameter `DirProj`.

For example, if we go in the folder `MMVII-MainFolder/MMVII-TestDir/Input/`, we can test :

```

MMVII EditSet Files/FileX.xml = "F[0].txt"
MMVII EditSet Files/FileX.xml = "F[0].txt" Show=2
++ F0.txt
MMVII EditSet FileX.xml = "F[0].txt" Show=2 DirProj=Files/
++ F0.txt

```

In the first two command, the project folder is computed from `Files/FileX.xml`. In the last command, it is computed from `DirProj=Files/`.

### 1.4.2 Filtering by interval FFI0, FFI1

Intervall can be used for different ordered type, for string the order is the standard lexicographic order. Interval are describe on command line usign square barcket, "[" and "]", separated by a comma ", ". Rather than a formal definition, explain by example :

- one can use closed interval : `[a100.jpg,a150.jpg]` filter the string  $S$  such that  $a100.jpg \leq S$  and  $S \leq a150.jpg$
- one can use open interval : `]a100.jpg,a150.jpg[` filter the string  $S$  such that  $a100.jpg < S$  and  $S < a150.jpg$
- interval can be semi open as `[a100.jpg,a150.jpg[` with obvious interpretation;
- interval can be semi finite : `[a100.jpg,[` filter the string  $a100.jpg \leq S$ , and no upper bound;
- finally one can create union of intervall by simply concateneng the string: `] ,a110jpg[ [a140.jpg,[` filter the string such that  $S < a110.jpg$  or  $a140.jpg \leq S$

The common optional parameter `FFI0` (and `FFI1`) can be used to do this filtering, for example , if we go in the folder `MMVII-MainFolder/MMVII-TestDir/Input/Files`, we can test :

```

MMVII EditSet File.xml = "F.*txt" FFI0="[F1.txt,F3.txt[]F7.txt,["
...
<el>F1.txt</el>

```

```
<el>F2.txt</el>
<el>F8.txt</el>
<el>F9.txt</el>
```

In this example, the parameter `FFI0` has been used to filter `"F.*txt"`, and gives the result described with the 4 names. Of course, the question is "How the user can know that the filter `FFI0` will apply to this parameter ?". This is where comes the "predefined semantics" [`MPI0`] that is shown in the help (see 1.4.5).

### 1.4.3 Redirecting message with StdOut

By default `MMVII` print several messages on the console. When user want to print the messages in a file `File.txt`, it is possible to :

- just append the messages at the end to the possibly existing file by `StdOut=File.txt`;
- just append the messages to the possibly existing file and still print the messages on the console by `StdOut+=File.txt`;
- print the messages in this file, and reset if it exist, `StdOut=0File.txt`;
- print the messages in this file, and reset if it exist, and still print the messages on the console by `StdOut=0+File.txt`;
- print nothing by `StdOut=NONE`.

This has for consequences that the name of the file of redirection cannot begin by `+` or `0`.

#### 1.4.3.1 Exercices

Try the following command and inspect the result , after each :

```
# File Mes.txt grows
MMVII EditSet File.xml = "F[0-3].txt" StdOut=Mes.txt Show=2
MMVII EditSet File.xml = "F[0-3].txt" StdOut+=Mes.txt Show=2
MMVII EditSet File.xml = "F[0-3].txt" StdOut=Mes.txt Show=2
# File Mes.txt reinialized
MMVII EditSet File.xml = "F[0-3].txt" StdOut=0Mes.txt Show=2
MMVII EditSet File.xml = "F[0-3].txt" StdOut=0+Mes.txt Show=2
MMVII EditSet File.xml = "F[0-3].txt" StdOut+=0Mes.txt Show=2
# No output
MMVII EditSet File.xml = "F[0-3].txt" StdOut=NONE Show=2
```

### 1.4.4 Fixing MicMac version for export NumVOut

As versions 1 and 2 of MicMac will coexist for several (many ?) years, it is usefull that new tools are able to import/export. For import, the solution is easy, `MMVII`, recognize by analyzing the first tag which version is it (if any). For export the rule are more complicated but quite logical, they use the common optionnal parameter `NumVOut` :

- if `NumVOut` is set (to 1 or 2) this fix the num version for export;
- else if there at least one file of `V2` was imported, the export will be in `V2`;
- else if there at least one file of `V1` was imported, the export will be in `V1`;
- else the export will be in `V2`;

### 1.4.5 Predefined semantics

#### 1.4.5.1 Generalities

Many parameters of many command of `MMVII` correspond to the same meaning/semantic, this is the case for "main set of images", "main orientation", ... These predefined semantic are indicated in square bracket after the types, for example with command `EditSet` we can see [`FDP`] , [`MPI0`] , [`FFI0`] , [`FFI1`] [`DP`] :

MMVII EditSet HELP

```
...
* string [FDP] :: Full Name of Xml in/out
* string [MPI0] :: Pattern or Xml for modifying
...
* [Name=FFI0] string [FFI0] :: File Filter Interval, Main Set   ### COMMON
* [Name=FFI1] string [FFI1] :: File Filter Interval, Second Set ### COMMON
* [Name=DirProj] string [DP] :: Project Directory   ### COMMON
..
```

We describe after this semantic (but not for common parameter, as they have already been described).

#### 1.4.5.2 Main pattern image [MPI]

Many command have a parameter which is the main set/pattern of files (generally images). This parameter is described by the predefined semantic [MPI0]. If this parameter exists, then it is possible to use [FFI0] to filter the set (if there is no [MPI0] then use of [FFI0] is forbidden).

Some command have several main sets, in this case one of their parameter will have the predefined semantic [MPI1], which can be filtered by [FFI1]. See the command `EditRel`.

#### 1.4.5.3 File of Directory Project [FDP]

The notion of project directory was introduced in 1.4.1. Generally there is no need to specify it, as there is one "main" file parameter that fix this directory. This parameter can be recognized by the predefined semantic [FDP], in `EditSet` command, this is first parameter that corresponds to this.

## 1.5 User's side-3, most frequent error

### 1.5.1 Generality

When a command fails, it generates an error message and generally wait for the user to press "return key". The first part of the message contains the type of error, it can be :

- `Level=[Internal Error]` : this mean that some incoherence in MMVII was encountered, probably in this case user cannot do many thing but report to forum or developping team mentioning the complete message;
- `Level=[UserEr:XXXX]` : this means that the error is probably due to a bad manipulation of the user, where XXX is the reference of the error;

For example :

```
MMVII EditSet File.xml = "F.*txt" ShowAll=tru
```

```
Level=[UserEr:BadBool]
```

```
Mes=[Bad value for boolean :[tru]]
```

Let comment the message :

- `Mes=[Bad value for boolean :[tru]]` : this as short message, which will be generally sufficient to analyse the error, here the error occurred because `ShowAll` is of type boolean and `tru` is not a valide string to create a boolean;
- `Level=[UserEr:BadBool]` : this line indicate the reference of the error, this reference can be used , if the short message is unsufficient, as an entry in this documentation to get more information on the error;

### 1.5.2 Error BadBool

This error occurs when a parameter of boolean type is initialized with an invalid string. Valide string for boolean are : `{0,1,false,true}` (case unsensitive). Example, parameter `ShowAll` being boolean :

```
MMVII EditSet File.xml = "F.*txt" ShowAll=tru
```

### 1.5.3 Error BadOptP

This error occurs when an optionnal parameter name do not match any the expected paramater name. Example, typing `AllShow` instead of `ShowAll`.

```
MMVII EditSet File.xml = "F.*txt" AllShow=true
```

### 1.5.4 Error MultOptP

This error occurs when the same optional parameter was used several time. Example doubling the `NumVOut` :

```
MMVII EditSet File.xml = ".*txt" NumVOut=1 NumVOut=1
```

### 1.5.5 Error OpenFile

This error occurs when MMVII cannot open a file, in read or write mode, several reason can exist : hard disk full, rights on the file system, directory do not exist. Example :

```
MMVII EditSet File.xml = ".*txt" Out=o/o.xml
```

### 1.5.6 Error InsufP

This error occurs when the number of parameter is inferior to the number of mandatory parameters. Example, omiting the operator in `EditSet`:

```
MMVII EditSet File.xml "F.*txt"
```

### 1.5.7 Error BadEnum

This error occurs when a string cannot create a specific enum. Example, typing `eq` instead of `=` in `EditSet`.

```
MMVII EditSet File.xml eq "F.*txt"
```

### 1.5.8 Error FileSetN

This error occurs when a File a file was expected to be a set of name and : the file exist (else it would be just an empty set) but is not a correct xml file in V1 or V2 format. Exemple under `MMVII-MainFolder/MMVII-TestDir/Input/Files`, using the file `BadFile.xml` :

```
MMVII EditSet BadFile.xml = .*txt
```

### 1.5.9 Error IntWithoutS

This error occurs when a file filter image (`FFI0`, `FFI1`) were used but the corresponding main pattern is not member of the command, for example :

```
MMVII EditSet BadFile.xml = .*txt FFI1=[,]
```

## 1.6 Programmer's side, adding a new command (1)

In this chapter we will see the main roadmap to follow for adding a new command in MMVII.

### 1.6.1 Heriting from `cMMVII_Appli`

A first and easy principle is "One command/One class" and "this class must inherit from `cMMVII_Appli`". In our case the class corresponding to `EditSet` is the class `cAppli_EditSet` defined in file :

— `MMVII-MainFolder//src/cMMVII_CalcSet.cpp`

As `cMMVII_Appli` is pure virtual class, the concrete class must override 3 methods :

- `int Exe();` : this method execute the action of the command;
- `cCollecSpecArg2007 & ArgObl(cCollecSpecArg2007 &)` this method communicate the specification of mandatory parameters;
- `cCollecSpecArg2007 & ArgOpt(cCollecSpecArg2007 &)` this method communicate the specification of optional parameters;

### 1.6.2 Link between name and class

The first thing `MMVII` has to do is to create the object heriting from `cMMVII_Appli` from the command name (here create a `cAppli_EditSet` from `EditSet`). To avoid huge compilation this creation is done without declaration of all the class in header; the philosophy is to have "hidden" derived class definition in ".cpp" files and just export an allocator. More precisely this is done via the class `cSpecMMVII_Appli` which is the specification of an application, it contains :

- an allocator function able to create a `cMMVII_Appli` from command line (see type `tMMVII_AppliAllocator`), here this is the function `Alloc_EditSet`;
- the name of the command, here `EditSet`;
- the comment of the command;
- three vector of specification : features (what main group the command belongs to), type of input, type of output; these specification are used in the `help` command to look for a given command (satisfying a request of the user such that "which command deal with orientation and produce ply data?");
- the file where the spec is created (using macro `__FILE__`) to help recovering file from command name.

Once the spec is created, it must be added to vector containing all the specification, this is done simply by adding a line as

— `TheRes.push_back(&TheSpecEditSet);`  
in file `src/Appli/cSpecMMVII_Appli.cpp`.

### 1.6.3 Specifying parameters

The specification of parameters is done by the methods `ArgObl` and `ArgOpt`. They both return a `cCollecSpecArg2007` with are an aggregation of `cSpecOneArg2007`. A `cSpecOneArg2007` contains the specification of one parameters, it is a virtual class that contains :

- the variable that will be initialized, this variable which can be of different type as it is contained in the derived classes;
- a vector of predefined semantics, a predefined semantic is create from one enum `eTA2007` and an optional additional string;
- the comment associated to the parameter;
- the name of the parameter (always empty string "" for mandatory parameters);

Let's make a brief comment with class `EditSet`, first mandatory parameters :

```
cCollecSpecArg2007 & cAppli_EditSet::ArgObl(cCollecSpecArg2007 & anArgObl)
{
    return
        anArgObl
            << Arg2007(mXmlIn,"Full Name of Xml in/out",{eTA2007::FileDirProj})
            << Arg2007(mOp,"Operator in (" + StrAllVall<eOpAff>()+")" )
            << Arg2007(mPat,"Pattern or Xml for modifying",{eTA2007::MPatIm,"0"});
}
```

Let's comment :

- The function `Arg2007` is template and adapt to the type of the l-value , here all the parameter are string , but could be `int ...` or any type pre-instatiated in `cReadOneArgCL.cpp` using macrp `MACRO_INSTANTIATE_ARG2007`;
- the `StrAllVall<eOpAff>()` function is used to generate the string of all valid operators;
- the first parameter will fix the project directory, we indicate this by a having the semantic `{eTA2007::FileDirProj}`
- the third parameter will be the first main set of file, with indicate it by the semantic `eTA2007::MPatIm,"0"`;

```
cCollecSpecArg2007 & cAppli_EditSet::ArgOpt(cCollecSpecArg2007 & anArgOpt)
{
    return
        anArgOpt
            << AOpt2007(mShow,"Show","Show detail of set before/after , (def) 0->none, (1) modif, (2) all",{ })
            << AOpt2007(mXmlOut,"Out","Destination, def=Input, no save for " + MMVII_NONE,{ });
}
```

Here we use the template function `AOpt2007`, it is used with a `bool` (parameter `mShow`) and a `std::string` (parameter `mXmlOut`). We see also that there is one more parameter : the name of the parameter that MicMac user will see (here `Show` or `Out`).

## 1.6.4 Standard access paramaters

Many command have paramaters that have more or less the same meaning. It is possible to avoid code redundancy via standard acces function and/or use of predefined semantic.

### 1.6.4.1 Reading set of name from file with `SetNameFromString`

### 1.6.4.2 Main sets with `MainSetk`



## Chapter 2

# Programming organisation, style ...

### 2.1 Naming convention

### 2.2 Never use `std::cout`, `printf` ...

### 2.3 Encapsulation of boost, stl ..

### 2.4 Error handling

### 2.5 Memory check

### 2.6 Serialization

### 2.7 Shared pointer

### 2.8 Random number

MMVII use some pseudo random generator. As every such generator they must be initialized with a seed. By default, the seed is always the same to facilitate debugging. When user wants initialization from time this must be specified with a negative value of parameter `SeedRand`.

### 2.9 Enum to string

The enum/string conversion is a recurrent problem of `C++` which as far as I know is still an issue. A possible solution would be to use some code generation which from easy to read and write text file would generate it. But I tried to limitate the code generation in MMVII.

In file `Serial/uti_e2string.cpp` is implemented the used solution, it consist to create data for each enum for which we want to do the conversion `Serial/uti_e2string.cpp`.



## Chapter 3

# Project management command

### 3.1 Help command



Part II

**Methodologies**



## Chapter 4

# The "Aime" methods for Tie Points computation

### 4.0.1 Indexing problem

For each image, we have computed tie points. A tie points is made of a vector  $V \in \mathbb{R}^n$  . Typically  $V$  is invariant to the main geometric deformation .

We have two sets of tie point corresponding to 2 image. Each tie points is characterized by a vector of





## Part III

# Reference documentation



# Part IV

## Annexes



## Appendix A

### Référence bibliographique



# Bibliography

- [Tomasi Kanabe 98] S. Roy, I.J. Cox , 1998, "Shape and Motion from Image Streams under Orthography: a Factorization Method", *International Journal of Computer Vision*, 9:2, 137-154 (1992)
- [Cox-Roy 98] S. Roy, I.J. Cox , 1998, "A Maximum-Flow formulation of the N-camera Stereo Correspondence Problem", *Proc. IEEE International Conference on Computer Vision*, pp 492–499, Bombay.
- [Fraser C. 97] C. Fraser, 1997, "Digital camera self-calibration", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 52, issue 4, pp. 149-159,
- [Penard L. 2006 ] L. Pnard, N. Paparoditis, M. Pierrot-Deseilligny. "Reconstruction 3D automatique de faades de btiments en multi-vues.", RFIA (Reconnaissance des Formes et Intelligence Artificielle), Tours, France, January 2006.

# Index

BadBool, 13  
BadEnum, 14  
BadOptP, 14  
  
EditSet, 7  
  
FileSetN, 14  
  
Help, 19  
  
InsufP, 14  
IntWithoutS, 14  
  
MultOptP, 14  
  
OpenFile, 14