goGPS code

Coding standards
and
Introduction to the development

Before start: Classes and Objects

Classes: data structures + methods (functions/routines) to manipulate those data

```
doc Class_Name % display HTML documentation
```

Object: it's an instance of a Class => a specific realisation that contains values that represent the state of the object

When an object is instantiated e.g. `rec = GNSS_Station();` the creator method (`name_of_the_class()`) of the class GNSS_Station is called

When all the pointers to an object are cleared, MATLAB call the destructor method (`delete()`)

Before start: Methods

Methods are functions that can see the properties of an object

Static functions are special functions that can be used independently from an instance of a class

`Class_Name.static_function_name()` can be called from everywhere, the function is able to see all the methods and constant of a Class but no variables.

A set of static function can be written within a class making the class a sort of lib of functions

Functions that are not static need to work on an object, they are declared as:

```
function out = Class_Name.static_function_name(this, in_var)
```

Before start: Methods

Static vs NON static

Static functions can work without an instantiated object
```
e.g. Core.initGeoid(geoid)
```

declared as:
```
    methods (Static, Access = public)
        function initGeoid(geoid)
         % The first parameter is just a regular variable
```

NON Statitic functions need an object to work on
```
e.g. rec_work.remUnderCutOff(15)
```

declared as:
```
    methods (Access = public)
        function remUnderCutOff(this, cut_off)
         % The first parameter is the object itself
         % (in this case rec_work)
         % the first parameter can be an array of objects, in this case it
         % will be managed within the function with regular indexes
         % this(i), if the function support working on arrays we append _list to
         % the name of the variable e.g. rec_list
         % (but unfortunately in the past we didn't do it) -> everytime it's possible
         % I change the name
```

Before start: Naming Conventions

To code the new release of goGPS we used some naming conventions (on the new code, but legacy functions) to immediately understand how to write functions, variables, classes.

The actual convention is the following:

Variables / objects, etc: lower case, word are separated by underscores
`this_is_a_variable`

Functions / methods: title case, the first letter is lower, words area not separated
`thisIsAFunction()`

Classes: title case, the first letter is capital, words are separated by underscores
`This_Is_A_Class`

```
e.g.
this_is_a_variable = This_Is_A_Class();
this_is_a_variable.initTheObject(323);

result = thisIsAFunction(this_is_a_variable.getCopy());
```

goGPS classes: Core

Core is the main goGPS class is instantiated into the object **core** at the beginning of goGPS execution.

It contains all the variables needed by goGPS to complete its processing:

- **log (Logger)**: the logging object
- **w_bar (Go_Wait_Bar)**: wait bar object (to create and manage it)
- **state (Main_Settings)**: it mirrors the settings.ini
- **sky (Core_Sky)**: to compute orbits of satellites and celestial bodies
- **atmo (Atmosphere)**: to manage anything (MF, delays, corrections) regarding atmosphere (ionosphere / troposphere …)
- **mn (Meteo_Network)**: to store data from meteorological stations and provide (VMS) virtual meteo stations (Meteo_Data))
- **rf (Core_Reference_Frame)**: to store information about station positions
- **cmd (Command_Interpreter)**: manage the goGPS pseudo language
- **geoid:** structure to store a quasi-geoid map for undulation computation
- **gom (Parallel_Manager)**: goGPS manager for parallel execution
- **net (Network)**: object to manage the network solutions

- **rec (**array of **GNSS_Station)**: it is the most important object, it contains the list of receivers/stations needed for computations and to store **results**

goGPS classes: GNSS_Station

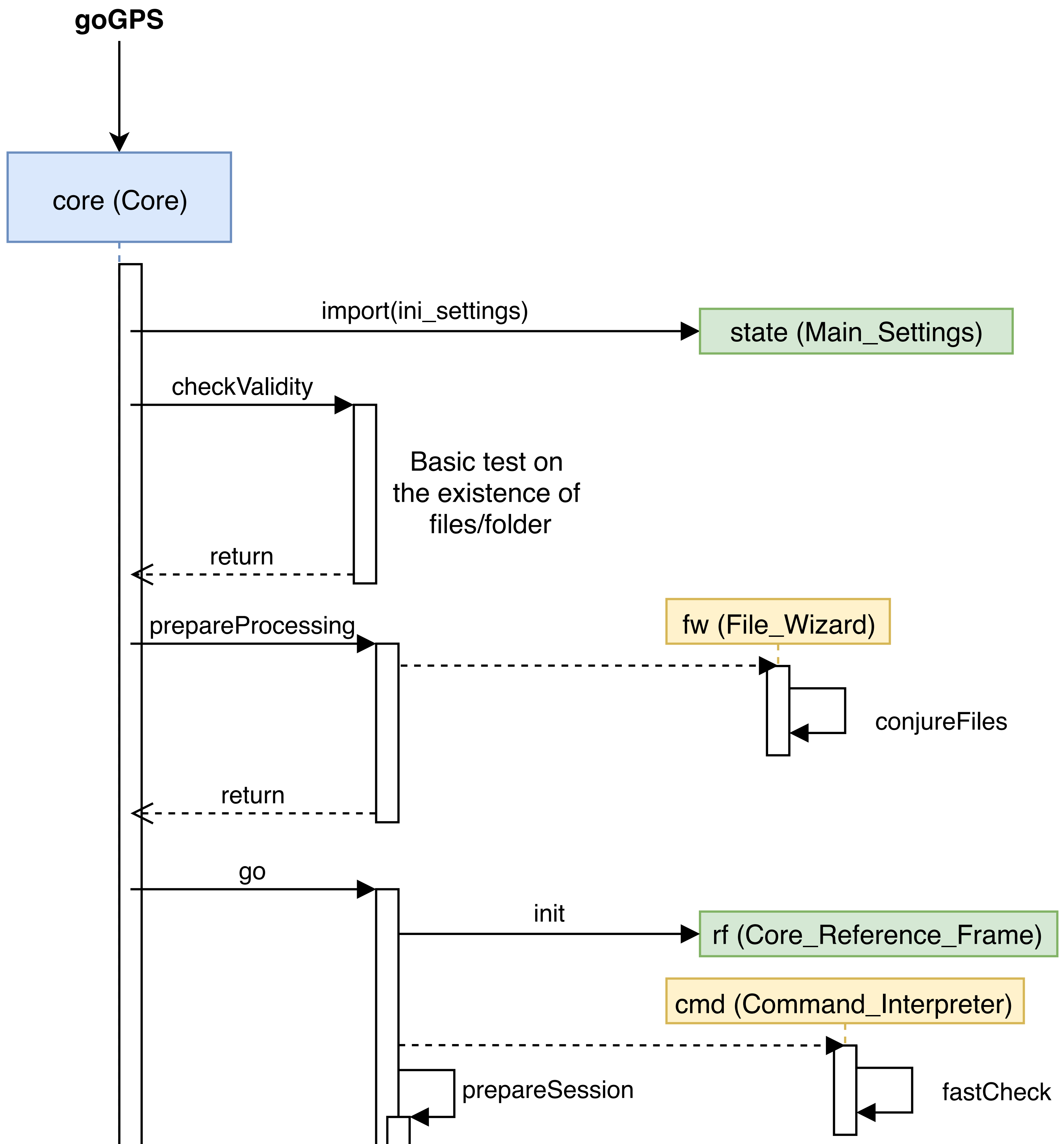
GNSS_Station it's a class to manage the data of a specific receiver (the observations that need to be used for the processing of a session and all the output)

It includes two important object:

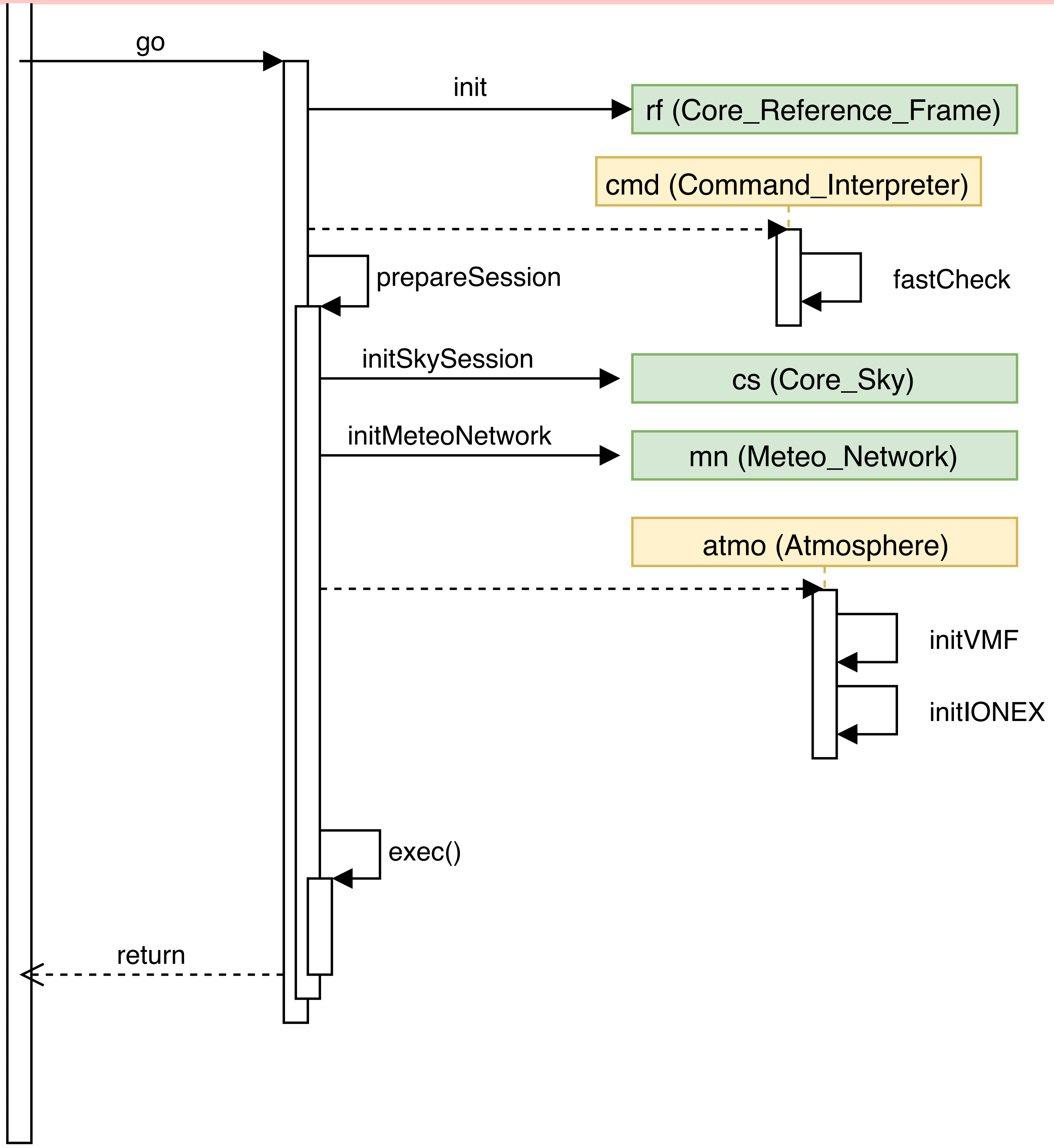
- **work** (Receiver_Workspace) that contains all the data relative to a receiver (phases pseudo_ranges, etc) and all the methods needed to pre-process and and compute stand-alone solutions. It always contains just the data loaded for the current processing session
- **out** (Receiver_Output) that contains all the cumulative results as computed session by session on the work (Receiver_Workspace) object. It has export and display functions

GNSS_Station have a set of methods to display results and export data (but they are basically calls to functions of the two main object work and out)

goGPS: working scheme



goGPS: working scheme



goGPS classes: Core.exec()

As you have seen all the real commands are executed through the core.exec command, that it’s an interface to the homonymous command of the Command_Interpreter object

```
core.cmd.exec(rec, cmd_list, level)
```

Once the core is present in the workspace it is possible to execute comments by calling this method. Multiple instructions can be executed, each command must be valid and written in a cell of strings.

```
e.g.
    core.exec({‘EMPTY T1’, ‘LOAD T1’, ‘PREPRO T1’, ‘PPP T1’});
```

The operation will be executed as if they were written in the goGPS initial interface.

Every command of the language must be coded into Command_Interface