

# **Fieldtrip from a software development perspective**

Version: April 20, 2023

1.	Introduction .....	3
2.	First impressions .....	4
2.1.	First impressions as a user .....	4
2.2.	First impressions as a software developer.....	5
3.	Topics of special interest.....	7
3.1.	The tests.....	7
3.2.	Code organization .....	9
3.2.1.	Code organization in general .....	9
3.2.2.	Private directories versus packages.....	10
3.2.3.	Self-contained modules .....	11
3.2.4.	Externals.....	12
3.3.	Preamble / postamble .....	13
4.	Code analysis tools.....	15
4.1.	Code duplication .....	15
4.2.	Mlint and Code Analyzer.....	15
4.3.	Code Compatibility Analyzer.....	16
4.4.	Checkcode function .....	16
4.5.	Dependency analysis.....	16
4.6.	Memory profiling .....	17
5.	Appendix: Specific code pieces .....	18
6.	Appendix: Questions and selected answers from first impressions as a software developer .....	20
7.	Appendix: A single matlab.unittest.TestCase wrapper.....	23
8.	Appendix: Considerations for the naming convention and subdivision of packages .....	24
9.	Appendix: Code examples for reimplementaion of preamble / postamble.....	26
9.1.	Current analysis function .....	26
9.2.	Analysis function with preamble and postamble v2.....	26
9.3.	Preamble v2 .....	27
10.	Appendix: Code Analyzer app results .....	29
11.	Appendix: Memory profiling examples.....	30
11.1.	Example of single call memory report .....	30
11.2.	Example of multi call memory report .....	30
12.	Appendix: Results of checkCode complexity report .....	33

## 1. Introduction

This document contains recommendations for improving the quality of the software in the Fieldtrip toolbox, with a focus on non-functional quality aspects like robustness and maintainability (as opposed to the functional quality: does the software give the correct answer).

It is part of a MathWorks Community Toolbox Project (CTP). The goals of these projects are to improve quality and performance of open source Matlab toolboxes. The work for the current CTP is done by VORtech, in association with Fieldtrip toolbox developers from Radboud University / Donders Institute for Brain, Cognition and Behaviour.

Improving the non-functional quality of the Fieldtrip toolbox based on the current recommendations, will make it easier to apply performance optimization. Already during this CTP, performance optimizations were briefly looked at. However, while looking for the best places to do performance optimization, various impediments to an overall performance analysis were found. Thus this document first focusses on those impediments.

This document has three main chapters, each further subdivided in sections. Each section ends with one or more recommendations. The main chapters are followed by several appendices, that give additional background information for some of the sections. Additionally, there is an “Appendix: Specific code pieces” with observations on some details in the code, that are not part of any of the sections.

## 2. First impressions

Since VORtech is a relatively new participant in the Fieldtrip toolbox, this gives the opportunity to record how a fresh set of eyes experiences working with and on Fieldtrip. Also, it brings a different perspective, looking at the code mostly from a software development perspective, instead of that of a scientific user. The first impression is an important one, since it should encourage a person to want to work with / on Fieldtrip. This chapter tries to convey the impressions of a first encounter with fieldtrip, and gives recommendations on improving those first impressions.

### 2.1. First impressions as a user

*To be inviting to new users, there should be a way to quickly get an answer to the following questions:*

- *What is Fieldtrip, what does it do, can it do the thing that I want to do?*
- *How do I start, to get the results that I am looking for?*

*What if I come to Fieldtrip as a new user, through the github repository. Is it clear what Fieldtrip is, and what it does?*

Before I even get to the README, at the top-level I first encounter a lot of directories, and an overwhelming number of files.

The README itself is very short and directs me to the website front page for the actual info on what Fieldtrip is and how to use it. The front page of the website then redirects me to the Getting started page. That Getting started page starts with two paragraphs that appear to try and warn me off from trying out Fieldtrip.

If I persist and do want to try out what Fieldtrip is about, the Getting started page has links to 10 different locations that I might want to visit, some of which are again a collection of links.

In those 10 links, the Walkthrough is the first page that directly speaks to me, as a prospective user, and starts to gently tell me something about how I could be using Fieldtrip. However, it is very quick in going to the data structures and missing a bit more introduction on the toolbox itself.

If I do get to the tutorial "Introduction to the FieldTrip toolbox", it also has useful introductory information, some of which is not in the walkthrough. And this tutorial gives me an idea of how the toolbox expects me to approach its functions.

Together, the Walkthrough and the Introduction tutorial do have the information to give me an impression of both the what and the how of Fieldtrip. However, getting there takes many steps, and has a lot of distractions on the way.

**Recommendation:** Have a single entry-point page on the website for new users (linked directly from the README), that gently introduces them to Fieldtrip. Approximately the first parts of the Introduction tutorial, with some of the first parts of the Walkthrough page. Then the Walkthrough can follow that with more information on what Fieldtrip does, the Introduction tutorial with how you are expected to do that yourself, and the Getting started page as the reference to all the other information that is available.

#### **Additional recommendations:**

- Reorganize the top-level directory, so the README can more easily be found, and the number of files and directories being encountered is less overwhelming. One clear entry point is more inviting than a hundred possible entry points. See also section Code organization.
- Have a directory examples within the repository, with example scripts with clear names, that are directly runnable, and produce a graphical output that speaks to the potential user.

Preferably this directory is added after reorganizing the top-level directory, otherwise this examples directory can easily go unnoticed.

## 2.2. First impressions as a software developer

To be inviting to prospective contributors, they should be able to quickly answer the following questions:

- Where to put my addition, and its dependencies?
- Does Fieldtrip already have a function that can be used instead of introducing a new dependency?
- What should my addition / edit look like (naming, code structure, etc.)?
- How do I check whether my edits to existing functions break other parts of the code?

Here a summary is given of the first impressions, when looking at the code base as a software developer, with the above questions in mind. This summary is based on various questions that come up, which are written down in Appendix: Questions and selected answers from first impressions as a software developer.

The repository content itself does not guide a developer to answers for most of these questions. There is not enough organization in the directory structure: directories with different status / purpose are all next to each other, and there are many files and directories at each level. The one question the repository does answer is the one on where to put additions and dependencies, however that answer isn't very satisfying, as it seems to be: all additions go into a high level directory, possibly the top level, next to a large amount of other files, and all dependencies go into the private directory of that specific high level directory.

The website does have pages that are related to these questions.

- The architecture page can be used as a reference for where to put additions. However, it gives the same answer as the repository itself, i.e. in a directory that already has a large number of files.
- The functions listing (<https://www.fieldtriptoolbox.org/reference/>) could be seen as a first answer to whether fieldtrip has certain functionality. However, the web page only gives a partial list of high level functions. And given the size of the toolbox and the lack of automated technical documentation generation, it cannot be expected from the website to be complete. Also, for utility functions, code organization is very helpful to guide developers in addition to / in lieu of documentation.
- The contributing page (<https://www.fieldtriptoolbox.org/development/contribute/>) does have guidelines for what additions should look like, although not all questions are answered yet (e.g. function naming).
- The page on testing (<https://www.fieldtriptoolbox.org/development/testing/>) answers how edits can be tested. However, the main answer on whether an edit affects other code, appears to be to wait for an email from someone at the Donders Institute on whether your pull request has passed the tests.

### Recommendations:

- To have a more satisfying answer to the question where to put additions and dependencies, reorganize the directory structure to collect related functions into smaller groups.
- To make it easier to find out what utility is already available, reorganize the directory structure to group utilities with related purposes.
- To make it easier to find out what is available, expand the reference documentation to include all public API functions. This would be separate from the existing reference page, as

that page is dedicated to listing only the most important functions. The expanded documentation would need to be automatically generated, various tools are available to help with that.

- To have an answer directly in the repository on what is expected of contributions, and to be in line with the common practice for GitHub, add a CONTRIBUTING.md file. It can mostly link to various pages on the website, e.g. look quite like the <https://www.fieldtriptoolbox.org/development/> page. Do add a part on naming conventions for files to that page.
- To make it easy to run at least a relevant subset of tests, update the tests organization

The next chapter has additional background and information related to recommendations on the code organization and testing.

### 3. Topics of special interest

From the above observations, and discussions with the main Fieldtrip developers, a number of topics have been identified for further analysis. This chapter has additional details on the tests organization, the code organization, and a specific code construct that is widely used in Fieldtrip.

#### 3.1. The tests

*“How do I check whether my edits to existing functions break other parts of the code?”*

Although the tests folder does have a lot of tests (over 1000 test files), not all of these are possible to run on a developer computer. The main limitation is that some tests use datasets that are only available in the Donders institute. Additionally, tests may have a prohibitive runtime or memory usage.

To encourage developers to run tests during development, and before submitting a pull request, it is recognized that it should be easier to run an appropriate subset of tests. It should be possible to filter tests based on:

- Runtime: to just do a quick check, there should not be hours-long tests in the selection
- memory usage: on a developer computer memory may be limited, there should not be tests in the selection that require more than the available memory
- datafiles used: some tests require external data, and some tests require data that is not publicly available, there should not be tests in the selection that require data that is not available on the developer computer
- Functions under test: to just do a quick check, only select tests that are known to call the function being edited

There exists a function `ft_test` (located in the utilities folder) that was meant to provide much of this functionality. However, it has largely gone unused (the dashboard does not use it) and undocumented (it is only mentioned in passing in one location in the documentation). It includes a custom implementation of (unit)test functionality that is also provided by Matlab itself. Because it is a custom implementation, it cannot use the additional functionality that is available within the Matlab unittest framework (e.g. reporting in junit format), does not allow potential testers to use the Matlab provided documentation to improve the functionality, and does not let developers that are familiar with the Matlab unittest framework to build on that knowledge.

Using `matlab.unittest.TestCase` enables various additional features.

One feature is the use of setup methods, teardown methods and fixtures. These can be used to reset the matlab session to a correct state for the next test also when a test does error, to close all figure windows after a test, or to temporarily manipulate what directories are on the path.

Another feature is the ability to easily collect code coverage information, which can be used to identify functions that are not or only poorly tested, through the `CodeCoveragePlugin`. The code coverage result can be exported in Cobertura XML format, which is a generally used format in continuous integration pipelines, and where various tools are available to help interpret the results.

One such tool is ReportGenerator (<https://github.com/danielpalme/ReportGenerator>), which has the ability to merge multiple reports.

#### **Recommendations:**

To enable developers to run a valid subset of the tests

- Add info on data use to the tests. This can probably be scripted, by looking for and parsing the use of the functions `dccnpath` and `ref_datasets`, and possibly some other functions. Similar to runtime and memory use, have a field in the test description for data, with one of NODATA/PUBLICDATA/RESTRICTEDDATA.

- Make it possible to select a subset of tests based on the mentioned criteria. Implement this by taking advantage of what is already available in the matlab unittest framework. This framework facilitates various selection methods, has additional features that can be useful for follow up improvements to tests usage, and is very well documented. To be able to select on the mentioned criteria, two alternatives present itself:
  - o Alternative one is to wrap each existing test function in a matlab.unittest.TestCase. The information on runtime etc. can then be stored such that direct selection through the matlab unittest framework is possible. It gives the most flexibility in adapting the tests for additional features, and the closest connection with the matlab documentation on using the unittest framework. The work to create the wrappers can be scripted. This option does require that a few changes are made to both the dashboard and how the octave testing calls the tests.
  - o Alternative two is to add a single matlab.unittest.TestCase wrapperfunction around the tests (details in Appendix: A single matlab.unittest.TestCase wrapper) to enable subset selection. This option does not require changes to the tests scripts, however it does introduce an additional, non-standard layer and requires more implementation and maintenance effort.
- Add a test runner utility function, for ease of use and to do the actual selection, and document how to run a subset of tests
- Add all the utility functions for testing in the test directory itself, possibly including the MOxUnit code, as that can make them easy to find. To actually find these utilities easily as entry points for doing testing, the test directory should be reorganized such that the top level does not directly contain all the test scripts (see the additional recommendations below on test directory organization).

#### **Additional recommendations:**

With the above recommendations in place, additional improvements are possible to the test usage. Listed in order of priority these would be

- Organize the tests directory with subdirectories that follow the modularity of the main directory structure. This will help in getting a better view of what is, and what is not tested. Later, it will make it possible to test if indeed specific modules are self-contained (see also the section on self-contained modules later in this document).
- Update the runtime and memory usage information inside the tests, and apply overhead calculations on the outside.

Currently overhead is included in each test's info. Because the tests are executed with <https://github.com/fieldtrip/dashboard/blob/master/schedule-batch.sh> every night, and the cluster scheduler/monitor will kill jobs that exceed requested resources (i.e. run too long), currently the requested time and memory are over-allocated. For a test that takes 1 second, 10 minutes are requested. For a test that takes 1 hour, 2 hours are requested. This also shows from a small sample run where tests were running in under a minute, whereas the test stated a runtime of 10 minutes.

Consider using the runtime (as reported by the matlab unittest framework) and memory usage (see memory profiling, peak memory) as reported by matlab, instead of also including the Matlab startup time and memory use. The Matlab startup time is not relevant when running multiple tests in a single Matlab session. The Matlab memory use will be different for each Matlab version and can also depend on installed toolboxes and apps. Both the startup time and extra memory can easily be added later to the test runtime and memory if needed. For example for the submission bash script, it would be better if this over allocation of time and memory were done in the script, rather than in the specification in the m-file header.



- Add GitHub actions to run a large subset of the tests on pull requests. A developer will typically only test on one Matlab version, and not run all possible tests. To give additional quick feedback on the quality of the pull request, add GitHub actions to run on a pull request. The actions can include both the latest Matlab, and the earliest supported (or for now available as GitHub action: 2020a) Matlab. To have the actions finish faster, multiple different subsets of tests can be run in parallel. The runner utility can be updated to provide the test results in a format which can be parsed by github.
- Add GitHub actions to run a large subset of the tests on Octave. Matlab users will probably not test Octave compatibility, whereas Fieldtrip does want to be compatible with Octave. A possibly useful GitHub action for this effort is <https://github.com/joergbrech/moxunit-action>. Not all matlab unittest framework features are directly compatible with Octave testing, so this will require code additions as well, or possibly the current contrib/MOxUnit\_fieldtrip within Fieldtrip already has most of what is needed.
- Add GitHub actions (triggerable or automatic) that run all nondata tests, if the default on-pull-request action does not include them all.
- Add GitHub action (triggerable or automatic) that runs all the public-data tests. This would need to include an action step to download the required data, and possibly code changes for the tests to be able to find the data.
- Consider updating the dependencies information. This information is aimed at listing the public API functions used, not at giving a full dependency tree. It should be quite possible to parse the output of a profiler session around a test (added e.g. through a trigger in the runner) and retrieve the public API functions that are touched by the test. This can then be used to update the dependencies information.
- Consider updating the fieldtrip/dashboard repository to use the wrapper and runner.

### 3.2. Code organization

*“Where to put my addition, and its dependencies? Does Fieldtrip already have a function that can be used instead of introducing a new dependency? How do I check whether my edits to existing functions break other parts of the code?”*

#### 3.2.1. Code organization in general

How code is organized has a large influence on how easy it is to do quality development. It affects ease of testing, how easy it is to find whether functionality already exists, how easy it is to get an overview of the codebase, where there are virtual boundaries within the codebase.

The current organization is described in <https://www.fieldtriptoolbox.org/development/architecture/>. The main items to mention here are that functions are divided into three categories (high-level, low-level and private) and the directory structure is set up to reflect a modular design. The distinction between high-level and low-level functions is related to the modular design, in that each module should be self-contained, and thus low-level functions should not call high-level functions. The private functions category is related to the modularity in that each module should have a stable public API, and the private functions are explicitly excluded from that API.

The directory organization approximately follows the architecture. However, the “Fieldtrip main functions” from the architecture image are not contained as such and placed above the modules. Instead, all the main functions are next to each other directly in the top level of the directory structure, next to the high level modules (distributed computing and contrib.), and next to the submodules. The de-facto entry point to fieldtrip, the ft\_defaults function, is also located at this level.

**Recommendation:** Reorganize the top level of the directory structure to more closely match the architecture, and make the entry point function stand out. This can be achieved by adding a directory main, that holds all functions that are now at the top level, except for ft\_defaults. The submodules as identified in the architecture would also be moved into this directory. Consider relabeling the “external” submodule to a high level module and keeping it at the top level.

### 3.2.2. Private directories versus packages

*“Finally, we have [private](#) functions that by design cannot be called by the end-user.”*

<https://www.fieldtriptoolbox.org/development/architecture/>

Having lots of functions in private directories does make a clear distinction between the public API and the internal functions. However, there are also considerable drawbacks to having many private directories, with many functions in each:

- Within a private directory, the code cannot be organized into directories any further. Thus, it is not possible to apply a modular design there, or to group related functions.
- Further directory subdivision within modules is severely discouraged, since each directory would need its own private directory, with copies of all the functions commonly used within the module. Thus, code organization through grouping of related functions is not used as much as would be helpful for a better understanding of the code dependencies and for the discovery of what functionality is available within a module.
- Since functions in a private directory are only visible from the directory level directly above it, those functions are not reachable from the test directory. Thus, no unit testing is applied to functions in private directories, whereas these are the building blocks that the larger functions are built on.
- There is a lot of file duplication, because to have an internal utility function available for use in different parts of the code, the file needs to be copied to every related private directory.
- It is much harder to discover what utility functions are already available. A utility can be present in another private folder, but not in the one for the level where a function is being added or edited. This leads to functionality duplication (different name, same purpose) and functionality discrepancies (same name, different purpose).

Within the Matlab environment, using private directories used to be the only reliable way of preventing users from developing code that accidentally depends on what Fieldtrip considers internal functions. This is because a common practice among Matlab users is to add all directories and sub directories of a toolbox to the Matlab path. Once that is done, all functions within the directory structure are visible to the code, without any indication of where they are coming from. For example, a user can easily start using the function `istrue` (which converts specific string inputs to a boolean true or false value) in his own code, without realizing that Fieldtrip considers this an internal function, and the next Fieldtrip release will have renamed the function, or changed what it does.

When Fieldtrip started with the current architecture, using packages was not an option yet. Fieldtrip aims to be backward compatible with about 5 years of Matlab versions. So packages being introduced in Matlab R2008a (and soon after also in Octave) were too new a feature for use in Fieldtrip when the current architecture came into being (in March 2011 already the modules and private-folders architecture with file synchronization is clearly visible). Also, within the Matlab community, this feature was at first seen as a part of the object-oriented features that were introduced at the same time.

By now, using packages and namespaces in Matlab is a viable alternative way of distinguishing between public and internal functions. Mathworks themselves are also heavily using packages in any

newer features they introduce to the language. And there too, packages are used to distinguish between public and internal functions. From

[https://nl.mathworks.com/help/matlab/matlab\\_oop/scoping-classes-with-packages.html](https://nl.mathworks.com/help/matlab/matlab_oop/scoping-classes-with-packages.html):

“Internal Packages”; “MathWorks® reserves the use of packages named internal for utility functions used by internal MATLAB code. Functions that belong to an internal package are intended for MathWorks use only. Using functions or classes that belong to an internal package is discouraged. These functions and classes are not guaranteed to work in a consistent manner from one release to the next. Any of these functions and classes might be removed from the MATLAB software in any subsequent release without notice and without documentation in the product release notes.”

**Recommendation:** Use packages instead of private directories for distinguishing between public and internal functions, to resolve the drawbacks of having many private functions as mentioned above. The exact naming of and hierarchy within the packages can be a gradual development. Various things to take into account when deciding on the naming and location of packages have been listed in Appendix: Considerations for the naming convention and subdivision of packages.

### 3.2.3. Self-contained modules

The Fieldtrip architecture page (<https://www.fieldtriptoolbox.org/development/architecture/>) states that one of the aims is for the low-level modules in FieldTrip to be fully self-contained. For the qsub and fileio modules, this goal is materialized in that they have their own repositories within the fieldtrip organization on github (<https://github.com/orgs/fieldtrip/repositories>), next to the main fieldtrip repository.

However, this goal is at odds with the architecture decision to give the utilities module a special treatment, currently phrased as “There is an exception for the utilities directory which allows lower-level functions to be called by the end-user at the level of the main FieldTrip functions.” And indeed functions from the utilities directory are called directly in many places in the other modules. Thus even the fileio module, which does have its own repository, is not usable without also getting the utilities directory from the main fieldtrip repository.

#### **Recommendation:**

- Update the documentation to be more clear that self-contained is not strict, but includes the utilities directory as a dependency.
- For modules / directories that are expected to be self-contained, add tests to check that they are indeed self-contained. Tests for functions in that module should be (able to) run with only that module’s directory added to the Matlab path.
- For modules that are provided as separate repositories, have everything in that repository that is expected for an independent piece of software. This would include:
  - o At least a minimum of documentation
  - o The software in a form that can be run as is, or information how to get the dependencies
  - o Tests that show the software works correctly (possibly just using a subset of the Fieldtrip tests is enough for this, see also the section on tests earlier in this document)
- For modules that are provided as separate repositories, since the modules will need more than just their own directory from the main Fieldtrip repository, the module repository should be set up such that it is easy to synchronize all the needed parts from the Fieldtrip repository to the module repository.

**Additional recommendation:** For modules that are provided as separate repositories, create a script in the module repository that largely automates the synchronization of the various parts (submodule, utilities, tests) from Fieldtrip towards the module repositories. Then use it in a Github

action that automates even more steps of the synchronization. The Github action can be set to automatically run whenever a new Fieldtrip release is created, or triggered manually. And can add the changes directly to the main branch of the module repository, or only create a branch-and-pull-request that is first tested independently (for instance by a github action in the module repository) before the changes are added to the main branch of the module repository. Since the source is here the main Fieldtrip repository, letting this workflow run fully through automated triggers is more feasible than for the externals in Fieldtrip, where the source of the changed code is outside of Fieldtrip.

### 3.2.4.Externals

*Does Fieldtrip already have a function that can be used instead of introducing a new dependency? How do I check whether my edits to existing functions break other parts of the code?*

Fieldtrip includes a directory named “external”, which according to its README, “contains toolboxes that are not maintained by the FieldTrip maintainers. These toolboxes are or can be used in combination with FieldTrip and are provided here with permission of the original toolbox authors as courtesy to the end-users.”

However, looking at the content and commit history of this directory, they show that these toolboxes are not treated the way externals are usually treated. The commit history shows that the source of external toolboxes is modified directly within the fieldtrip repository, separate from changes in the original source. For example, the file synchronization utility also updates files within the externals part of the repository. Also, fieldtrip specific functions are called from within functions in these toolboxes. A quick search shows that at least artinis, bemcp, dipoli, dmlt and mmfmatlabio use ft\_ functions, like ft\_error and ft\_getopt. Because these toolboxes are modified directly in the Fieldtrip repository, and call Fieldtrip specific functions, synchronization with the external source, for instance to push or pull bugfixes, is very hard.

It can be necessary to modify an external, to be able to use it as part of Fieldtrip. In order to keep track of what exactly has changed, and be able to push to and pull from the original source, a separate intermediate repository within the fieldtrip organization can be created for such an external. For these repositories, the main branch follows the outside source unconditionally. And a separate branch is used to apply the Fieldtrip specific edits. This branch is then used to update the specific external within Fieldtrip. And changes to that external within Fieldtrip should be ONLY from synchronization with that branch.

#### **Recommendations:**

- Commits to external subdirectories in Fieldtrip, should only apply changes that are a synchronization to a specific state of the upstream source. A commit to an external in Fieldtrip should be able to indicate the exact external reference that is the basis for the commit.
- For externals with both active outside development and Fieldtrip specific edits, create an intermediate repository within the Fieldtrip GitHub organization, to facilitate the synchronization. When the outside source already has a repository on GitHub, the GitHub fork workflow can be used to create a Fieldtrip specific fork repository.
- For externals with no (relevant) outside development and (ongoing) Fieldtrip specific edits, consider relabeling them as ‘integrated’. This makes it clear that, although they are not considered a part of Fieldtrip, they are also not expected to be directly synchronizable with their upstream, and can be freely edited within Fieldtrip.

**Additional recommendation:** For externals that get their own fork within the Fieldtrip GitHub organization, create a script in the Fieldtrip fork that helps in automating the synchronization of the

external towards Fieldtrip. This can be a part of the branch that contains the Fieldtrip specific changes. Once there is a script, it can be used as the basis for a GitHub action that prepares even more of the synchronization towards Fieldtrip.

### 3.3. Preamble / postamble

*“What should my addition / edit look like (naming, code structure)?”*

The preamble and postamble functions are used in all high level analysis functions, to do bookkeeping that is always nearly the same. However, the current implementation has been found to have certain drawbacks.

The main drawback of the current implementation of preamble and postamble is that it is not transparent that these functions modify the workspace of the function they are called from. While in effect, they do add several new variables, and modify existing variables (in particular the `cfg` variable).

A related drawback is their use of `assignin`, `evalin` and worker scripts, which makes it hard to follow what happens, and where. And the use of `assignin` and `evalin` also makes that currently, nested functions cannot be used in any function that calls preamble or postamble. Because the use `evalin` and `assignin` is not compatible with nested functions, as also stated in the documentation: [https://nl.mathworks.com/help/matlab/matlab\\_prog/resolve-error-attempt-to-add-variable-to-a-static-workspace.html](https://nl.mathworks.com/help/matlab/matlab_prog/resolve-error-attempt-to-add-variable-to-a-static-workspace.html). Even though regular functions and subfunctions are usually to be preferred over nested functions, not being able to use nested functions at all is an unnecessary restriction.

From a first look at what the preamble and postamble worker scripts try to achieve, it seems possible to rework them to a version 2 implementation, that does show transparently what variables are added / edited in the calling function's workspace.

The final implementation could have a single function call for each of preamble and postamble. That call takes multiple cell arrays as input, one for each preamble 'script' that should be executed, followed by any other input data needed by that particular 'script'. The special `cfg` input would be a shared first input. The call will return `cfg`, and for the preamble a single variable that holds any extra info needed by the postambles, and the extra variables that are requested from the `loadvar` option as `varargout` output towards named variables in the caller function.

The final implementation does not have to do any `assignin` calls or addition of extra variables under the hood. And most likely it will not even need to call `evalin`.

Another result will be that the functions that use preamble and postamble, can get rid of the separate `ft_nargin`, `ft_nargout` and `ft_revision` variables.

An intermediate step in the implementation, that could be kept also in the final situation to provide backward compatibility, would be to convert the worker scripts into regular functions, called in a regular fashion from preamble / postamble. And do all the interaction with the caller workspace only in preamble / postamble themselves through `assignin` and `evalin` (to get variable values from the caller) calls. Only one struct-type variable would be added to the caller workspace, to hold all the information that needs to be passed on from preamble to postamble.

**Recommendation:** Replace preamble and postamble with `preamble_v2` and `postamble_v2`, where the `_v2` functions are transparent in what inputs they use and what outputs they provide, and do not modify the caller workspace under-the-hood.

Do a step-by-step implementation, that includes backward compatibility to support users that have written their own analysis functions.

- First implement a `_v2` function for each of the individual worker scripts, that only work with inputs and outputs. Call this new `_v2` version directly in preamble / postamble, and do the caller workspace interaction only in preamble / postamble.
- Then, create the overall `_v2` calls, that can then internally look very much like the preamble/postamble, reusing the calls to the individual parts. In the overall `_v2`, regular inputs / outputs will be used instead of interactions with the caller workspace.
- Apply the overall `_v2` calls in the code base, and update the documentation.

Some example of what the v2 implementations would look like, has been added in Appendix: Code examples for reimplementing of preamble / postamble.

## 4. Code analysis tools

To assess code quality, many tools are available that can help to evaluate specific aspects. Some of these are most useful for a one-time analysis. Others can also be used in an automated fashion to continuously monitor a quality aspect and provide feedback to developers.

This chapter aims to introduce these analysis tools and provide recommendations on how to use them with Fieldtrip. Most of the tools mentioned are provided as part of the Matlab distribution.

### 4.1. Code duplication

Applying a code duplication analysis tool on the current Fieldtrip will give a lot of duplication sites, since there is known function duplication through the synchronization mechanism. Thus it is currently hard to assess if there is also additional code duplication. However, even while looking at only a few random functions, there were indications that there is also in-function code duplication. There are tools that can help to detect code duplication.

- MathWorks has PolySpace for example (<https://nl.mathworks.com/products/polyspace.html>), however, that appears to be for c/c++ code only.
- There is the copy-paste detector (CPD) of PMD, which claims to support Matlab (<https://pmd.github.io>), though only for the CPD part, not for all of PMD. Not yet checked whether this only detects exact matches, or also near-exact duplication.
- It might be possible to adapt pylints similarity checker to run correctly on Matlab code (currently at <https://github.com/pylint-dev/pylint/blob/main/pylint/checkers/similar.py>)

**Recommendation:** Apply a code duplication analysis tool on the code base, once the known function duplication has been removed, to help find additional code duplication.

### 4.2. Mlint and Code Analyzer

The well-known mlint errors and warnings (that show up as squiggles in the matlab editor) can now easily be viewed in one overall report, since as of Matlab 2022b there is Code Analyzer App (<https://nl.mathworks.com/help/matlab/ref/codeanalyzer-app.html>), and a codeIssues function (<https://blogs.mathworks.com/developer/2023/03/15/static-analysis-code-checking-and-linting-with-codeissues/>). Previously it was also possible to collect these messages, but not this easily. Running the analysis on the full fieldtrip folder gives [128 error, 18143 warning, 29909 info] messages. Note that this is a total. For instance, the warning "TRY statement should have a CATCH statement to check for unexpected errors." is given on 235 different lines in the code, concentrated in even fewer different files.

For a screenshot of the errors summary, see Appendix: Code Analyzer app results.

#### Recommendations:

- Make a pass over the code base to get rid of the error messages.
- Update the coding guidelines to require that files should in principle have no mlint error or warning messages (which can be checked with the codeIssues function).

#### Additional recommendations:

- Make a pass over the list of warning messages, and determine which are serious enough to require fixing. Warnings can indicate various types of problems, of which some are more serious than others.
- Add an automated test to detect whether edits increase the number of error and warning messages. And schedule a periodic update of the reference number-of-errors and number-of-warnings (it should be going down over time).

### 4.3. Code Compatibility Analyzer

Compatibility of the code with a given Matlab version, can be checked with the Code Compatibility Analyzer. Through the Matlab command line interface the functions `analyzeCodeCompatibility` and `codeCompatibilityReport` have been available since 2017b, and as an App since 2022b. This check currently identifies [6 files with syntax errors, 25 + 14 cases of code that will not work in the 2022b version and sometimes have not worked since 2015b, 94 cases of possibly changed behavior]. This in addition to 40 + 9 cases that might give problems in the future, and 2772 improvement suggestions. Note that there is an overlap with the `mlint` / Code Analyzer results (see above), so problems may show up in both reports.

**Recommendation:** Make a pass over the code base to fix the syntax errors and the not-in-2022b cases.

#### Additional recommendations:

- Make a pass over the cases of possibly changed behavior, since these may cause different results in different Matlab versions, to check whether they are problematic or not. Often code that is known to still be ok, can be adapted to no longer show up in the analysis report.
- Add an automated test to detect whether edits increase the number of errors and serious warnings (will-not-work, changed-behaviour). And schedule a periodic update of the reference number-of-errors (it should be going down over time)

### 4.4. Checkcode function

The function `checkcode`, with the option `-cyc` or `-modcyc`, will give “the McCabe cyclomatic complexity of each function in the file. In general, lower complexity values indicate programs that are easier to understand and modify. Evidence suggests that programs with higher complexity values are more likely to contain errors. Frequently, you can lower the complexity of a function by dividing it into smaller, simpler functions.”

Given that the analysis functions follow a similar pattern in general, that introduces some complexity, a low complexity number is not to be expected for the main analysis functions. Thus the general guidelines with boundaries at 10, 20 and 50

([https://nl.mathworks.com/help/matlab/matlab\\_prog/measure-code-complexity-using-cyclomatic-complexity.html](https://nl.mathworks.com/help/matlab/matlab_prog/measure-code-complexity-using-cyclomatic-complexity.html)) may not directly apply. However, complexity numbers above 100 do indicate that a function implementation will probably be hard to follow.

With some scripting `checkcode` could be used to find the fieldtrip files that are the most. A first example and result can be found in Appendix: Results of `checkCode` complexity report.

#### Recommendations:

- When code is being refactored, apply the `checkcode` function to evaluate the before and after complexity, and aim for a reduction in complexity.
- Add (more) tips on general ways to reduce complexity to the coding guidelines. Quick gains are using switch-case instead of if-elseif, and placing large chunks of related code into subfunctions.

### 4.5. Dependency analysis

A dependency analysis gives information on how functions interact with other functions. Matlab does have functionality for dependency analysis, however as far as I can find it is only available within the Matlab Project functionality. Thus first a Matlab Project needs to be created for Fieldtrip, before the dependency analysis can be used. Creating a project did work, although it took quite



some time. However, accessing the dependency analysis through the GUI of the Matlab Project failed, a message came up that it was unable to finish creating a graph.

There is also a command line interface to the dependency analysis (since 2019a), that still relies on first creating a Matlab Project. The reference for the Matlab Project (<https://nl.mathworks.com/help/matlab/ref/matlab.project.project.html>) shows the command line interface for matlab projects, including examples of interacting with the dependencies information. More examples, for instance of how to get dependency information for particular files, can be found on [https://nl.mathworks.com/help/matlab/matlab\\_prog/create-and-edit-projects-programmatically.html](https://nl.mathworks.com/help/matlab/matlab_prog/create-and-edit-projects-programmatically.html), section Get File Dependencies.

Note that even once the Matlab Project is created, calling the `updateDependencies` method again takes a long time.

**Recommendation:** Do not invest time yet in dependency analysis. This tooling can be useful if there is a particular goal to be achieved that requires information on function dependencies. However, there does not appear to be such a goal at present. Also, the use of many private directories with duplicated functions makes that the dependency analysis will not be exhaustive in showing what functions depend on a certain lower level function, making it less useful in evaluating the impact of changes to a lower level function.

#### 4.6. Memory profiling

Matlab has a built in profiling function, `profile`, that can give detailed information on where time is spent when running code. As with all profiling this introduces some overhead, thus the results may not be precisely represent where the time is spent when running without the profiler. But in general doing a profiling session will give useful information as to where code might be improved to reduce runtime.

This `profile` function also has an (undocumented) option `'-memory'` to track not only runtime, but also memory usage (use `'-nomemory'` to explicitly turn it off again). This can be very useful if memory usage is suspected to be the cause of problems. However, note that enabling this option introduces even more overhead, and depending on the computer used may considerably slow down the code (it can go from 3 to 10, but also from 3 to 160 seconds). Thus, this option should not be used to evaluate the effect on runtime directly, only to identify how memory is allocated and freed. Changes to the runtime of the code should be evaluated separately. And, where possible it should be applied only to the piece of code that is under investigation (e.g. using a script with specially prepared inputs), and not directly at a high level.

Interpreting results of a memory profiling session can be tricky, but also very useful. For instance, the number of times a line is executed needs to be taken into account when looking at the effect of that line on memory. Total / cumulative memory usage is not immediately available as information in the report. Thus a timing bottleneck, even if it has significant memory usage, may just be a symptom of overall high memory usage. And fixing that one location's memory will just move the problem further on in the code.

**Recommendation:** When performance of the code is being evaluated, use the `timeit` and `profile` functions to identify runtime bottlenecks. Then, follow up with a memory profiling run, only on the bottleneck location, using dedicated inputs, to get information on the amount of memory interaction at that location.

## 5. Appendix: Specific code pieces

While looking through the Fieldtrip code base in order to get an overview, and be able to make general recommendations, also various functions were looked at in a bit more detail. Below are questions and remarks that do not directly apply to Fieldtrip in general, but only to specific functions.

<https://github.com/fieldtrip/website/pull/657>

`ft_defaults`: mainly used in the top-level functions, as these are all considered entrypoints. However, `ft_defaults` is also called in some not-top-level functions. In functions in the `contrib`, `realtime`, `utilities` and even in the external folder?

Why is `spike` located within `contrib`?

What even is `contrib`?

Why does `getopt` have a mex version?

The history of `mesh_sphere` in the `external/dipoli` is odd for an external. And it looks like it is now totally unused there.

In `ft_freqanalysis`, the switch case statement around line 280, doing method specific preparation. Most of the cases look similar, with a list of `cfg` fields setting, and some error checks.

- The case for `mtmconvol` looks quite different from the other ones, whereas it could be structured more similar, especially with a small subfunction to set the `cfg.toi` value.
- The case for `mvar` suddenly does a function call and an early return, instead of only `cfg` preparation. To me, this should be more explicit, which can be achieved by treating that method separately, before going into the switch case.

In `ft_crossfrequencyanalysis`

- Multiple places where there is a `strcmp(xxx, 'no')`, whereas there is also an `istrue` function.
- Why is the `cfg` option output for these even a string, instead of a boolean value? Why not have `ft_checkconfig` or `ft_getopt` return a boolean value for such fields?
- In the switch case for “do the actual computation” around line 215, multiple cases have the exact same code.
- The case ‘mi’ has an if-else, where the inner part of both branches is exactly the same. Why is that part not a subfunction? (A nested function might be even nicer, since that could go directly at that location, but the preamble prevents use of nested functions?)

Naming of `ft_xxxx_option` versus `ft_xxxx` combined with `cfg.method`

There are many analysis functions that have a switch case on `cfg.method`. However, there are also some top-level functions that appear to have the method as a suffix. Why?

In `ft_artifact_nan`, there is a double loop, where every inner loop iteration causes a memory allocation. As also suggested by the `mlint` squiggle, it would be more efficient to pre-allocate, just before the inner loop.

Fieldtrip `compat` folder has entries that go back to before 2013b. However, the `table` datatype is being used in multiple places in `fieldtrip`, so even though the `matlab2013b` folder has a `istable` function, isn't backward compatibility currently limited to 2013b and later?

Fixcoordsys uses an if-elseif construct that has no final else statement. Not sure if the static code analysis tools pick that up. However, even if it is correct to do-nothing, there should be an else that describes the motivation why it is correct. Similar for switch case statements, they should always have a final otherwise.

Website, <https://www.fieldtriptoolbox.org/development/architecture/>: “All high-level functions within the FieldTrip directories may call functions within the same directory, from other directories at the same hierarchical level, or directories lower in the hierarchy. But, low-level functions should not call high-level functions. There is an exception for the utilities directory which allows lower-level functions to be called by the end-user at the level of the main FieldTrip functions.”

I think I understand the idea, however I don't understand the last sentence's phrasing.

Something that was not looked at yet, is an answer to “would like to refactor some of the plotting related functions, but not sure how”

Most plotting functions are like regular fieldtrip functions, but with a lot of subfunctions for e.g. callbacks. Only `ft_plot_mesh_interactive` uses a `classdef`, and is actually the only object-oriented file in the fieldtrip code (except for certain places in the external folder).

## 6. Appendix: Questions and selected answers from first impressions as a software developer

There are about 100 top level functions. This is quite overwhelming.

- How to find the right function? What functions have related functionality? Why aren't the data conversion (x2y) functions together in a separate directory?

*"Finding the right function is indeed a challenge. Only a selection of all analysis functions is linked on <https://www.fieldtriptoolbox.org/reference/>. And since these are direct links to the github repository, the help content of the functions is not part of the website, and not indexed by search engines."*

- Automatically generating technical documentation, to be included as a part of a website should be feasible. Sphinx for instance (<https://www.sphinx-doc.org/en/master/>) has extensions that let it parse Matlab files (<https://github.com/sphinx-contrib/matlabdomain>) and produce output in markdown format or restructured text. Thus the technical documentation can either be made a part of the official Fieldtrip website, or be hosted separately on Read the Docs (<https://readthedocs.org/>).

Many top level folders.

- How do they relate to each other? Do they all serve a similar purpose?

Only some of the top-level folders are considered part of the public interface, according to what is put on the matlab path when running `ft_defaults`, and the information in <https://www.fieldtriptoolbox.org/development/architecture/> The architecture page has some background on the current architecture.

- Why are there many non-interface folders on the top level, instead of having them all together in one folder, called developers or resources or extras or ...? Or even in one or more subfolders of utilities?

There is very little hierarchy in the directory structure. No namespaces / packages at all.

- Why not? How do you know what functions are related?

A lot of functions in each 'private' directory.

- What is the interdependency between the functions in a private folder? How to apply unit testing to functions in private directories?

A lot of function duplication.

- How to know that indeed all copies are synchronized, so bugs are fixed? How to check that each dependent function still works correctly after synchronization of the dependencies? How to know if the same function name is not used for different functionality elsewhere in the code? The duplicate functions are synchronized based on file change time on disk?

*"There's a `synchronize_private.sh` script in `fieldtrip/bin`, that ideally lists the locations of where the functions in `../private` are duplicate. This script is also used to synchronize the various copies, once one of the copies is changed on github"*

Especially in the private folders, there is a mix of functions that do and do not have the `ft_` prefix.

- When does a function get the `ft_` prefix?

*"Sometimes there is a public function without the prefix, that is usually for historical reasons, to keep the API stable. In private directories, also historical reasons to do have the `ft_` prefix sometimes, but there it should not be used. Coding guideline should be updated for the private functions not to have the prefix."*

Most analysis functions are named with the scheme `ft_ANALYSISNAME`, and have internal branching on `cfg.method`. However, some are named with the scheme `ft_ANALYSISNAME_SPECIFIC`.

- Why don't all analyses follow the internal-branching strategy?

*"This applies to statistics functions. These are present so that we keep elaborate documentation to a specific method on one location"*

Functions with many lines of code.

- How to do unit testing, when the units are very big?

The analysis functions appear to follow a similar code structure. This does help in recognizing the main parts of the functions (initialization, input checking, data preparation, main analysis, output organization, finalization).

The source of external toolboxes is modified directly within the fieldtrip repository.

- How to get changes in from and out to the original toolbox? How to know what has and has not been synchronized with the external source?

There is a test directory, however a brief look through its readme and the follow-up links towards the dashboard, gives the impression that I will be unable to test my own changes. And the test directory has more than 1000 files in one directory!

- How can I run the tests locally? Which tests are unit tests for specific lower-level functions, which are tests on public-API functions, which are system tests on a workflow? What set of tests can be run quickly to give basic confidence in changes? Over 50 `inspect_type` tests, who runs these? When is the output of the `inspect_tests` checked? Why haven't they been converted to regular tests? Why are the `obsolete_tests` still in the test directory?

Lots of mex files, all over the code base, and mixed with other files.

- Why is the source code sometimes directly next to the m and mex files?

*"The idea is that the fieldtrip-native mex-files' source code is in fieldtrip/src/, all the other Mex-files seem to be in external packages mostly, and a bit in the realtime module. I found one instance of a \*.c file in a private folder. This one should be removed. Also we have discussed removing the external/dm1t altogether, which would get rid of a large chunk of code anyhow."*

The modules of Fieldtrip are supposed to be independent: "We aim for the low-level modules in FieldTrip that they have a consistent API and are fully self-contained, i.e. if you copy the corresponding directory out of the main FieldTrip directory, they should still work. This facilitates the low level code to be reused in other projects."

- Should they not be separate repositories then, so it is possible to test this goal is achieved? Which tests are for what module? Is this statement true for all the low-level modules? Is this feature ever used?

The `compat` folder has code to make fieldtrip compatible with older Matlab releases. The oldest release mentioned is R2010b. The website states that 5 years of backward compatibility is at least aimed for. The dashboard script (<https://github.com/fieldtrip/dashboard/blob/master/schedule-matlabs.sh>) shows that only R2012a and later is tested. The code has locations where the `table` datatype is used, introduced in R2013b.

- With the `table` datatype being used in multiple places in fieldtrip, isn't backward compatibility currently limited to 2013b and later?
- With only 2012a and later being tested, compatibility for earlier versions does no longer exist effectively?

- Should not some of the compat directories be removed then? Which ones exactly? What policy is used here?

## 7. Appendix: A single matlab.unittest.TestCase wrapper

The standard way of using a `matlab.unittest.TestCase`, defines the tests as methods of a class that derives from `matlab.unittest.TestCase`. Selecting a subset of those tests can be done by grouping them in different methods blocks, each with a different set of tags. See for example

[https://nl.mathworks.com/help/matlab/matlab\\_prog/tag-unit-tests.html](https://nl.mathworks.com/help/matlab/matlab_prog/tag-unit-tests.html)

Reusing the current implementation of the tests is possible, by setting up a `matlab.unittest.TestCase` that collects the available test, and through test parameterization enables selecting specific subsets.

The main ideas can be derived from this example,

[https://nl.mathworks.com/help/matlab/matlab\\_prog/define-parameters-at-suite-creation-time.html](https://nl.mathworks.com/help/matlab/matlab_prog/define-parameters-at-suite-creation-time.html)

The wrapper would need to first collect all available tests, and parse them to extract information on runtime, memory use, data use etc. This can probably re-use code available in `ft_test`. Each of these properties will become a separate testparameter, and by defining the `ParameterCombination` attribute on the single test method as `sequential`, one testcase per test function will be created.

The test runner utility function can then apply constraints on the parameters

(<https://nl.mathworks.com/help/matlab/ref/matlab.unittest.constraints-package.html> and

<https://nl.mathworks.com/help/matlab/ref/matlab.unittest.testsuite.selectif.html>), to select a subset of tests, for instance tests with a runtime of less than 3 minutes, using `NODATA` and having a name starting with `test_`.

## 8. Appendix: Considerations for the naming convention and subdivision of packages

A first subdivision would already need to contain separate packages for

- Common functions: all functions that are currently synchronized over multiple private directories, and any other functions that are duplicated but not yet synchronized
- Toplevel functions: the non-common functions in the private directory at the repository top level
- Utilities: the non-common functions in the private directory inside the utilities directory
- Specest: the non-common functions in the private directory inside the specest directory
- And many more: a separate package for the non-common functions in the private directory inside each separate submodule directory

Eventually there should be additional hierarchy / grouping within the packages.

To make clear that these are for internal use, a naming convention would include “internal” in the name. To distinguish from internal packages from other sources, the naming convention would start with “ft”.

Each level of hierarchy in package names requires a similar hierarchy in directory structure. Using a '.' in the name of a package directory does not create a multilayered package (actually, it will break Matlab's import mechanism such that the contents of the directory cannot be used).

As an example, possible names for the package containing the common functions could be 'ft.internal.common', 'ft\_internal.common' or 'ft\_internal\_common', each requiring a different directory layout. Similarly, the package containing the toplevel internal functions could be 'ft.internal.toplevel', 'ft\_internal.toplevel' or 'ft\_internal\_toplevel'.

The package `ft_internal.common` would contain the functions that are used by multiple different modules, i.e. the files that are now synchronized over multiple directories. The name `ft_internal.shared` would also be possible, however that name has more potential to be misinterpreted by users, as the internal and shared parts give conflicting messages. The name `ft_internal.utilities` would have been possible, except that name is already to be used by the utilities' module for its internal functions.

A module should only use functions from `ft_internal.common` and its own package. This to keep as close as possible to the self-contained goal (see also the section on self-contained modules, and the next consideration).

The module specific package can be inside the module directory. The `ft_internal.common` package might be best put in the utilities directory, next to the `ft_internal.utilities` package. Since functions from the utilities directory are used in other submodules, this directory is already needed alongside other submodules when those submodules are used independently. Adding the common package here, has the advantage that it will be taken along automatically.

A namespace in Matlab can be built from directories in multiple locations. Thus, it is possible to have a directory “+ft\_internal” at multiple locations, and they will all be part of the `ft_internal` package.

If packages are at first only going to be used for the organization of the internal functions, and the public APIs will remain packageless, it might not be worth it to have an extra layer of packaging / directories called “+ft\_internal”, like in “fieldtrip/specest/+ft\_internal/+specest/ETC”. Instead it could be considered to not have a shared `ft_internal` package, but multiple packages that start with



the same name, and thus have single package directory layer per module, like "fieldtrip/specest/+ft\_internal\_specest/ETC".

## 9. Appendix: Code examples for reimplementing of preamble / postamble

These are meant as example, the actual implementation may have to look somewhat different

### 9.1. Current analysis function

An example of what the current implementation looks like

```
function [outputs] = ft_important_analysis_current(cfg, inputs)

% The current implementation
% Based mainly on ft_componentanalysis

% these are used by the ft_preamble/ft_postamble function and
scripts
ft_revision = '$Id$';
ft_nargin   = nargin;
ft_nargout  = nargout;

% do the general setup of the function
ft_defaults
ft_preamble init
ft_preamble debug
ft_preamble loadvar data
ft_preamble provenance data
ft_preamble randomseed

% the ft_abort variable is set to true or false in ft_preamble_init
if ft_abort
    return
end

% then
% here
% do
% the
% actual
% analysis
comp = 1;

% do the general cleanup and bookkeeping at the end of the function
ft_postamble debug
ft_postamble randomseed
ft_postamble previous data
ft_postamble provenance comp
ft_postamble history comp
ft_postamble savevar comp

% and sometimes a small code piece after the postamble
end
```

### 9.2. Analysis function with preamble and postamble v2

What the analysis function would look like with the v2 of preamble / postamble

```
function [outputs] = ft_important_analysis_final(cfg, inputs)

% The final implementation
% Based mainly on ft_componentanalysis

% do the general setup of the function (no separate ft_nargin etc
% variables)
```

```

[cfg, p_amble, data] = ft_preamble_v2(cfg, ...
    {'init', nargin, nargout}, ...
    'debug', ...
    {'loadvar', 'data'}, ...
    {'provenance', 'data'}, ...
    'randomseed');
% Note that for loadvar and provenance, the name given as extra
input will
% not be used to name variables. The input is meant to indicate the
number
% of arguments, and whether they are a cell array or separate
variables.
if p_amble.ft_abort
    return
end

% then
% here
% do
% the
% actual
% analysis
comp = 1;

% do the general cleanup and bookkeeping at the end of the function
[cfg, comp] = ft_postamble_v2(cfg, ...
    {'inputs', data, comp}, ...
    'debug', ...
    'randomseed', ...
    {'previous', 1}, ...
    {'provenance', 2}, ...
    {'history', 2}, ...
    {'savevar', 2});
% Note that the number input to previous, provenance, history and
savevar
% indicates which of the inputs are input for that part. Instead of
a
% scalar, this can also be a vector of numbers.
% The number input would be a required input, and have a scalar 0 to
% indicate no input data used. It can be followed by an optional
input
% string / chararray 'vararg' to indicate that the input is a cell
array.

% and sometimes a small code piece after the postamble
end

```

### 9.3. Preamble v2

#### Some details of v2 of preamble and underlying functions

```

function [cfg, p_amble, varargout] = ft_preamble_v2(cfg, varargin)
% NOT an actual implementation of ft_preamble_v2
% Just shows possible function signatures of ft_preamble_v2 and the
% underlying functions.

[cfg, p_amble] = ft_preamble_init(cfg, nr_argin, nr_argout);

[cfg, p_amble] = ft_preamble_debug(cfg, p_amble);

[cfg, p_amble, varargout] = ft_preamble_loadvar(cfg, p_amble,
load_args);























```

```
[cfg, p_amble] = ft_preamble_provenance(cfg, p_amble, prov_args);  
[cfg, p_amble] = ft_preamble_randomseed(cfg, p_amble);  
end
```

## 10. Appendix: Code Analyzer app results

Showing only the errors here

Analysis Date: 4/6/2023, 1:58:13 PM

Error (128)
▶  The statements under this elseif condition cannot be reached because it is a duplicate of the if condition on line 438. Re...
▶  Using integers to specify location not supported. Use the location parameter name and value pair instead. (21)
▶  Use of two dots (..) is an invalid MATLAB construction. (20)
▶  This condition has no effect because all blocks in this if statement are identical. Remove the condition or change the cod...
▶  The case value '4' is a duplicate of one on line 2026. (10)
▶  Since both operands are identical, the second operand has no effect on the '  ' operation. Change one of the operands or...
▶  A '(' might be missing a closing ')', causing invalid syntax at '162'. (5)
▶  Parse error at FUNCTION: usage might be invalid MATLAB syntax. (4)
▶  This logical comparison always returns false. Did you mean to use < to evaluate function argument: isnan(...<...)? (3)
▶  The comparison will likely fail due to case mismatch. (3)
▶  'princomp' has been removed. With appropriate code changes, use 'pca' instead. (2)
▶  Code analysis did not complete. File contains too many syntax errors. (1)
▶  A quoted character vector is unterminated. (1)
▶  Use a newline, semicolon, or comma before this statement. (1)
▶  'svmtrain' has been removed. With appropriate code changes, use 'fitsvm' instead. (1)
▶  'svmclassify' has been removed. With appropriate code changes, use 'ClassificationSVM' instead. (1)
▶  Variable must be explicitly defined before first use. (1)
▶  'fwrite' is writing to a file that is opened with read permission only. Open a file using 'fopen(...,'W',...)' instead. (1)
▶  To pass MException properties to the warning function, use a format specifier. For example, warning(E.identifier, '%s', E....
▶  The expression cannot be assigned to multiple values. (1)
▶  Initialize VARARGOUT with a CELL. (1)
▶  An END might be missing (after line 111), possibly matching IF. (1)

## 11. Appendix: Memory profiling examples

### 11.1. Example of single call memory report

In timelockanalysis, there is the following line

```
datacov = ft_checkdata(datacov, 'datatype', 'timelock');
```

From a specific run that calls timelockanalysis once, the memory profiling shows the following:

↳ Lines that take the most time or memory

Line Number	Code	Calls	Total Time (s)	Allocated Memory (Kb)	Freed Memory (Kb)	Peak Memory (Kb)	% Time	Time Plot
<a href="#">129</a>	datacov = ft_checkdata(datacov, 'data...	1	1.100	533504.00	359604.00	531740.00	10.9%	

And going to the line mentioned

```

1.202    1      326112.00    72.00    1896.00    126    datacov = ft_selectdata(tmpcfg, data);
                                127    % restore the provenance information
0.038    1                                128    [dum, datacov] = rollback_provenance(cfg, datacov); % not sure what to do here
1.100    1      533504.00    359604.00    531740.00    129    datacov = ft_checkdata(datacov, 'datatype', 'timelock');
                                130
< 0.001    1                                131    if isfield(datacov, 'trial')
```

Using the matlab debugger, it is possible to stop at that line, and check the size of the datacov variable before and after the line is executed.

Runnig whos before ft\_checkdata is called shows

```

Name      Size      Bytes  Class  Attributes
datacov   1x1      337531528  struct
```

Runnig whos after ft\_checkdata is called shows

```

Name      Size      Bytes  Class  Attributes
datacov   1x1      545328786  struct
```

Thus, what we see happening: datacov the variable is reused in this line, however the output is larger than the input. Thus, new memory needs to be allocated to hold the output, and the old memory location will no longer be needed. The Allocated Memory is (approximately) equal to the new size of datacov, and the Freed Memory is (approximately) equal to the old size. The input memory location is freed, since the reference to that data is not used anymore.

### 11.2. Example of multi call memory report

In ft\_freqanalysis, ft\_specest\_mtmconvol is called in a loop. Within ft\_specest\_mtmconvol, there is another loop.

In this example, ft\_freqanalysis calls ft\_specest\_mtmconvol 77 times.

Profile Summary (Total time: 4.936 s)

↳ Flame Graph

Generated 07-Apr-2023 12:21:45 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
<a href="#">ft_freqanalysis</a>	1	4.895	0.332	
<a href="#">ft_specest_mtmconvol</a>	77	3.609	3.105	
<a href="#">ft_checkdata</a>	2	0.479	0.009	
<a href="#">ft_datatype_sens</a>	32	0.386	0.079	

Then within ft\_specest\_mtmconvol there is a loop that is run with 15 iterations (on average, each call to ft\_specest\_mtmconvol does not necessarily run the inside loop with the same number of iterations) leading to the lines within that loop being called  $77 \times 15 = 1155$  times.

ft\_specest\_mtmconvol (Calls: 77, Time: 3.609 s)

► Flame Graph

Generated 07-Apr-2023 12:22:42 using performance time.  
 Function in file Z:\Checkouts\Mathworks\FieldTrip\fieldtrip\specest\ft\_specest\_mtmconvol.m  
[Copy to new window for comparing multiple runs](#)

▼ Parents (calling functions)

Function Name	Function Type	Calls
ft_freqanalysis	Function	77

▼ Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
356	dum = fftshift(iff(datspectrum .* repmat(wltspectrm{ifreqoi...	1155	2.610	72.3%	
338	[st, cws] = dbstack;	1155	0.229	6.3%	

From the above timing profiling, we also see that about 50% of the total runtime of the freqanalysis call (2.6 of 4.9 seconds), appears to be spent on line 356 within ft\_specest\_mtmconvol. Below the relevant part of the code is shown:

```

333     % start fft'ing
334     datspectrum = fft(ft_preproc_padding(dat, padtype, 0, postpad), [], 2);
335     spectrum = complex(zeros([nchan ntimeboi sum(ntaper)]));
336     for ifreqoi = 1:nfreqoi
337         str = sprintf('frequency %d (%.2f Hz), %d tapers', ifreqoi,freqoi(ifreqoi),ntaper(ifreqoi));
338         [st, cws] = dbstack;
339         if length(st)>1 && strcmp(st(2).name, 'ft_freqanalysis') && verbose
340             % specest_mtmconvol has been called by ft_freqanalysis, meaning that ft_progress has been initial
341             ft_progress(fbopt.i./fbopt.n, ['trial %d, ',str,'\n'], fbopt.i);
342         elseif verbose
343             fprintf([str, '\n']);
344         end
345         for itap = 1:ntaper(ifreqoi)
346             % compute indices that will be used to extracted the requested fft output
347             nsampleifreqoi = timwin(ifreqoi) .* fsample;
348             reqtimeboiind = find((timeboi >= (nsampleifreqoi ./ 2)) & (timeboi < (ndatsample - (nsample
349             reqtimeboi = timeboi(reqtimeboiind);
350
351             % compute datspectrum*wavelet, if there are reqtimeboi's that have data
352             % HERE, TRYOUTS
353             % dum = datspectrum .* wltspectrm{ifreqoi}(itap,:);
354             % dum = datspectrum .* repmat(wltspectrm{ifreqoi}(itap,:),[nchan 1]);
355             % dum = fftshift(iff(dum, [], 2),2); % fftshift is necessary to implement zero-phase/acyclic/aca
356             dum = fftshift(iff(datspectrum .* repmat(wltspectrm{ifreqoi}(itap,:),[nchan 1]), [], 2),2); % fft
357             tmp = complex(nan(nchan,ntimeboi),nan(nchan,ntimeboi));
358             tmp(:,reqtimeboiind) = dum(:,reqtimeboi);
359             tmp = tmp .* sqrt(2 ./ timwinsample(ifreqoi));
360             spectrum(:, :, freqtapind{ifreqoi}(itap)) = tmp;
361         end
362     end % for nfreqoi
363 end % switch dimord
364

```

The code on line 356 is doing multiple things: calling two functions and creating some new data on the fly with an additional function call, all-in one line. Doing multiple things in one line can sometimes use less memory, but it can also use more memory than dividing the actions over multiple lines. Thus, this is a good occasion to enable memory profiling.

When we enable memory profiling, the Allocated Memory and Freed Memory columns show that, over the total of 1155 calls, more than 2 Gb is allocated and freed on this line, so on average 2Mb per call. This is of the same order of magnitude as the peak memory of 4Mb, so it may be that each iteration of the inner loop is allocating and freeing memory. This warrants a better look at the code, whether it is possible to improve it such that the iterations of the inner loop are able to reuse the memory location.

ft\_specest\_mtmconvol (Calls: 77, Time: 3.574 s, 2773028.00 Kb, 2688308.00 Kb, 4024.00 Kb)

Flame Graph

Generated 07-Apr-2023 12:27:12 using performance time.  
 Function in file Z:\Checkouts\Mathworks\FieldTrip\fieldtrip\specest\ft\_specest\_mtmconvol.m  
[Copy to new window for comparing multiple runs](#)

Parents (calling functions)

Function Name	Function Type	Calls
ft_freqanalysis	Function	77

Lines that take the most time or memory

Line Number	Code	Calls	Total Time (s)	Allocated Memory (Kb)	Freed Memory (Kb)	Peak Memory (Kb)	% Time	Time Plot
356	dum = fftshift(iff(datspectrum .* repmat(wl...	1155	2.525	2421844.00	2270044.00	4024.00	70.7%	
334	datspectrum = fft(ft_preproc_padding(dat, pa...	77	0.094	159696.00	1908.00	2104.00	2.6%	
335	spectrum = complex(zeros([nchan ntimeboi sum...	77	0.051	110280.00	0.00	1436.00	1.4%	
95	dat = ft_preproc_polyremoval(dat, polyorder,...	77	0.105	79760.00	6104.00	1052.00	2.9%	

```

354 % dum = datspectrum .* repmat(wltspectrum(ifreqoi)(itap,:),[nchan 1]);
355 % dum = fftshift(iff(dum, [], 2),2); % fftshift is necessary to implement zero-phase/acyclic/
2.525 1155 2421844.00 2270044.00 4024.00 356 dum = fftshift(iff(datspectrum .* repmat(wltspectrum(ifreqoi)(itap,:),[nchan 1]), [], 2),2); %
0.050 1155 16.00 32.00 16.00 357 tmp = complex(nan(nchan,ntimeboi),nan(nchan,ntimeboi));
0.053 1155 0.00 16.00 0.00 358 tmp(:,reqtimeboiind) = dum(:,reqtimeboi);
0.051 1155 16.00 0.00 16.00 359 tmp = tmp .* sqrt(2 ./ timwinsample(ifreqoi));
0.017 1155 32.00 0.00 16.00 360 spectrum(:,:,freqtapind(ifreqoi)(itap)) = tmp;
0.001 1155 361 end
0.001 1155 1092.00 0.00 1092.00 362 end % for nfreqoi
< 0.001 77 363 end % switch dimord
364
    
```

In this case, it is indeed possible to reduce the amount of memory allocation and freeing, by preusing the 'dum' variable that is already present, and splitting the line so the matlab interpreter is able to determine that the same variable is both input and output to the nested fftshift and ifft calls.

```

354 % dum = datspectrum .* repmat(wltspectrum(ifreqoi)(itap,:),[nchan 1]);
355 % dum = fftshift(iff(dum, [], 2),2); % fftshift is necessary to imple
1.286 1155 162992.00 96.00 2100.00 356 dum = datspectrum .* repmat(wltspectrum(ifreqoi)(itap,:),[nchan 1]);
1.806 1155 1532.00 2104.00 968.00 357 dum = fftshift(iff(dum, [], 2),2); % fftshift is necessary to impleme
0.051 1155 2184.00 16.00 1928.00 358 tmp = complex(nan(nchan,ntimeboi),nan(nchan,ntimeboi));
0.061 1155 16.00 0.00 16.00 359 tmp(:,reqtimeboiind) = dum(:,reqtimeboi);
0.064 1155 16.00 80.00 16.00 360 tmp = tmp .* sqrt(2 ./ timwinsample(ifreqoi));
0.019 1155 361 spectrum(:,:,freqtapind(ifreqoi)(itap)) = tmp;
0.002 1155 362 end
0.002 1155 363 end % for nfreqoi
< 0.001 77 364 end % switch dimord
    
```

Note that dum will be created 77 times, for each call to ft\_specest\_mtmconvol. This does match the Allocated Memory of 77\*2Mb approx 160Mb. And note that the Freed Memory over the now two lines is nearly 0 compared to the earlier 2Gb. Thus in this case indeed, within each call to ft\_specest\_mtmconvol, the memory allocated to dum is reused in each loop iteration. This is possible, because the size of dum does not change from the functions applied to it. Both the data put into it in the first line and the fftshift and ifft calls in the second line, all produce output that is the same size. And the fft functions have apparently been implemented by Matlab to efficiently use the already allocated memory when possible.



## 12. Appendix: Results of checkCode complexity report

The following snippet can be used to get the complexity of all m files in a directory

```
a = dir("*.m");
b = {a.name};
c = cell(size(b));
for i = 1:length(b)
    info = checkcode(b{i}, '-modcyc');
    c{i} = info(1).message;
end
for i = 1:length(c)
    disp(c{i})
end
```

Applied to the main fieldtrip directory, the results are

The modified cyclomatic complexity is 0.  
The modified cyclomatic complexity of 'besa2fieldtrip' is 51.  
The modified cyclomatic complexity of 'bis2fieldtrip' is 1.  
The modified cyclomatic complexity of 'data2bids' is 287.  
The modified cyclomatic complexity of 'edf2fieldtrip' is 6.  
The modified cyclomatic complexity of 'fieldtrip2besa' is 13.  
The modified cyclomatic complexity of 'fieldtrip2bis' is 1.  
The modified cyclomatic complexity of 'fieldtrip2ctf' is 4.  
The modified cyclomatic complexity of 'fieldtrip2fiff' is 17.  
The modified cyclomatic complexity of 'fieldtrip2homer' is 6.  
The modified cyclomatic complexity of 'fieldtrip2spss' is 2.  
The modified cyclomatic complexity of 'ft\_analysispipeline' is 21.  
The modified cyclomatic complexity of 'ft\_annotate' is 2.  
The modified cyclomatic complexity of 'ft\_anonymizedata' is 12.  
The modified cyclomatic complexity of 'ft\_appenddata' is 52.  
The modified cyclomatic complexity of 'ft\_appendfreq' is 22.  
The modified cyclomatic complexity of 'ft\_appendlayout' is 28.  
The modified cyclomatic complexity of 'ft\_appendsens' is 58.  
The modified cyclomatic complexity of 'ft\_appendsource' is 75.  
The modified cyclomatic complexity of 'ft\_appendspike' is 23.  
The modified cyclomatic complexity of 'ft\_appendtimelock' is 22.  
The modified cyclomatic complexity of 'ft\_artifact\_clip' is 27.  
The modified cyclomatic complexity of 'ft\_artifact\_ecg' is 39.  
The modified cyclomatic complexity of 'ft\_artifact\_eog' is 5.  
The modified cyclomatic complexity of 'ft\_artifact\_jump' is 5.  
The modified cyclomatic complexity of 'ft\_artifact\_muscle' is 5.  
The modified cyclomatic complexity of 'ft\_artifact\_nan' is 16.  
The modified cyclomatic complexity of 'ft\_artifact\_threshold' is 41.  
The modified cyclomatic complexity of 'ft\_artifact\_tms' is 16.  
The modified cyclomatic complexity of 'ft\_artifact\_zvalue' is 53.  
The modified cyclomatic complexity of 'ft\_audiovideobrowser' is 38.  
The modified cyclomatic complexity of 'ft\_badchannel' is 17.  
The modified cyclomatic complexity of 'ft\_badsegment' is 17.  
The modified cyclomatic complexity of 'ft\_channelnormalise' is 13.  
The modified cyclomatic complexity of 'ft\_channelrepair' is 59.  
The modified cyclomatic complexity of 'ft\_clusterplot' is 67.  
The modified cyclomatic complexity of 'ft\_combineplanar' is 37.

The modified cyclomatic complexity of 'ft\_componentanalysis' is 120.  
The modified cyclomatic complexity of 'ft\_conjunctionanalysis' is 39.  
The modified cyclomatic complexity of 'ft\_connectivityanalysis' is 266.  
The modified cyclomatic complexity of 'ft\_connectivityplot' is 72.  
The modified cyclomatic complexity of 'ft\_connectivitysimulation' is 38.  
The modified cyclomatic complexity of 'ft\_crossfrequencyanalysis' is 33.  
The modified cyclomatic complexity of 'ft\_databrowser' is 98.  
The modified cyclomatic complexity of 'ft\_defacemesh' is 2.  
The modified cyclomatic complexity of 'ft\_defacevolume' is 34.  
The modified cyclomatic complexity of 'ft\_defaults' is 99.  
The modified cyclomatic complexity of 'ft\_definetrial' is 25.  
The modified cyclomatic complexity of 'ft\_denoise\_dssp' is 8.  
The modified cyclomatic complexity of 'ft\_denoise\_hfc' is 11.  
The modified cyclomatic complexity of 'ft\_denoise\_pca' is 28.  
The modified cyclomatic complexity of 'ft\_denoise\_prewhiten' is 20.  
The modified cyclomatic complexity of 'ft\_denoise\_ssp' is 16.  
The modified cyclomatic complexity of 'ft\_denoise\_synthetic' is 13.  
The modified cyclomatic complexity of 'ft\_denoise\_tsr' is 17.  
The modified cyclomatic complexity of 'ft\_detect\_movement' is 13.  
The modified cyclomatic complexity of 'ft\_dipolefitting' is 92.  
The modified cyclomatic complexity of 'ft\_dipolesimulation' is 27.  
The modified cyclomatic complexity of 'ft\_electrodeplacement' is 47.  
The modified cyclomatic complexity of 'ft\_electroderealign' is 92.  
The modified cyclomatic complexity of 'ft\_electrodermalactivity' is 5.  
The modified cyclomatic complexity of 'ft\_eventtiminganalysis' is 36.  
The modified cyclomatic complexity of 'ft\_examplefunction' is 2.  
The modified cyclomatic complexity of 'ft\_freqanalysis' is 166.  
The modified cyclomatic complexity of 'ft\_freqanalysis\_mvar' is 29.  
The modified cyclomatic complexity of 'ft\_freqbaseline' is 21.  
The modified cyclomatic complexity of 'ft\_freqdescriptives' is 28.  
The modified cyclomatic complexity of 'ft\_freqgrandaverage' is 37.  
The modified cyclomatic complexity of 'ft\_freqinterpolate' is 6.  
The modified cyclomatic complexity of 'ft\_freqsimulation' is 127.  
The modified cyclomatic complexity of 'ft\_freqstatistics' is 19.  
The modified cyclomatic complexity of 'ft\_globalmeanfield' is 3.  
The modified cyclomatic complexity of 'ft\_headmovement' is 30.  
The modified cyclomatic complexity of 'ft\_hearttrate' is 28.  
The modified cyclomatic complexity of 'ft\_interactiverealign' is 45.  
The modified cyclomatic complexity of 'ft\_interpolatenan' is 10.  
The modified cyclomatic complexity of 'ft\_lateralizedpotential' is 12.  
The modified cyclomatic complexity of 'ft\_layoutplot' is 21.  
The modified cyclomatic complexity of 'ft\_math' is 71.  
The modified cyclomatic complexity of 'ft\_megplanar' is 22.  
The modified cyclomatic complexity of 'ft\_megrealign' is 26.  
The modified cyclomatic complexity of 'ft\_meshrealign' is 17.  
The modified cyclomatic complexity of 'ft\_movieplotER' is 10.  
The modified cyclomatic complexity of 'ft\_movieplotTFR' is 51.  
The modified cyclomatic complexity of 'ft\_multiplotCC' is 20.  
The modified cyclomatic complexity of 'ft\_multiplotER' is 91.  
The modified cyclomatic complexity of 'ft\_multiplotTFR' is 83.  
The modified cyclomatic complexity of 'ft\_mvaranalysis' is 86.

The modified cyclomatic complexity of 'ft\_neighbourplot' is 26.  
The modified cyclomatic complexity of 'ft\_networkanalysis' is 59.  
The modified cyclomatic complexity of 'ft\_prepare\_headmodel' is 68.  
The modified cyclomatic complexity of 'ft\_prepare\_layout' is 213.  
The modified cyclomatic complexity of 'ft\_prepare\_leadfield' is 36.  
The modified cyclomatic complexity of 'ft\_prepare\_mesh' is 23.  
The modified cyclomatic complexity of 'ft\_prepare\_montage' is 22.  
The modified cyclomatic complexity of 'ft\_prepare\_neighbours' is 68.  
The modified cyclomatic complexity of 'ft\_prepare\_sourcemodel' is 122.  
The modified cyclomatic complexity of 'ft\_preprocessing' is 98.  
The modified cyclomatic complexity of 'ft\_recodeevent' is 26.  
The modified cyclomatic complexity of 'ft\_redefintrial' is 63.  
The modified cyclomatic complexity of 'ft\_regressconfound' is 26.  
The modified cyclomatic complexity of 'ft\_rejectartifact' is 102.  
The modified cyclomatic complexity of 'ft\_rejectcomponent' is 27.  
The modified cyclomatic complexity of 'ft\_rejectvisual' is 37.  
The modified cyclomatic complexity of 'ft\_removeplateartifact' is 8.  
The modified cyclomatic complexity of 'ft\_reproducescript' is 3.  
The modified cyclomatic complexity of 'ft\_resampleddata' is 51.  
The modified cyclomatic complexity of 'ft\_respiration' is 16.  
The modified cyclomatic complexity of 'ft\_scalpcurrentdensity' is 22.  
The modified cyclomatic complexity of 'ft\_singleplotER' is 84.  
The modified cyclomatic complexity of 'ft\_singleplotTFR' is 65.  
The modified cyclomatic complexity of 'ft\_sliceinterp' is 80.  
The modified cyclomatic complexity of 'ft\_sourceanalysis' is 218.  
The modified cyclomatic complexity of 'ft\_sourcedescriptives' is 306.  
The modified cyclomatic complexity of 'ft\_sourcegrandaverage' is 26.  
The modified cyclomatic complexity of 'ft\_sourceinterpolate' is 96.  
The modified cyclomatic complexity of 'ft\_sourcemovie' is 34.  
The modified cyclomatic complexity of 'ft\_sourceparcellate' is 36.  
The modified cyclomatic complexity of 'ft\_sourceplot' is 267.  
The modified cyclomatic complexity of 'ft\_sourceplot\_interactive' is 22.  
The modified cyclomatic complexity of 'ft\_sourcestatistics' is 22.  
The modified cyclomatic complexity of 'ft\_sourcewrite' is 17.  
The modified cyclomatic complexity of 'ft\_statistics\_analytic' is 5.  
The modified cyclomatic complexity of 'ft\_statistics\_crossvalidate' is 15.  
The modified cyclomatic complexity of 'ft\_statistics\_montecarlo' is 78.  
The modified cyclomatic complexity of 'ft\_statistics\_mvpa' is 87.  
The modified cyclomatic complexity of 'ft\_statistics\_stats' is 35.  
The modified cyclomatic complexity of 'ft\_steadystatesimulation' is 26.  
The modified cyclomatic complexity of 'ft\_stratify' is 95.  
The modified cyclomatic complexity of 'ft\_timelockanalysis' is 29.  
The modified cyclomatic complexity of 'ft\_timelockbaseline' is 35.  
The modified cyclomatic complexity of 'ft\_timelockgrandaverage' is 27.  
The modified cyclomatic complexity of 'ft\_timelocksimulation' is 18.  
The modified cyclomatic complexity of 'ft\_timelockstatistics' is 27.  
The modified cyclomatic complexity of 'ft\_topoplotCC' is 39.  
The modified cyclomatic complexity of 'ft\_topoplotER' is 8.  
The modified cyclomatic complexity of 'ft\_topoplotIC' is 16.  
The modified cyclomatic complexity of 'ft\_topoplotTFR' is 8.  
The modified cyclomatic complexity of 'ft\_virtualchannel' is 51.

The modified cyclomatic complexity of 'ft\_volumebiascorrect' is 14.  
The modified cyclomatic complexity of 'ft\_volumedownsample' is 14.  
The modified cyclomatic complexity of 'ft\_volumelookup' is 77.  
The modified cyclomatic complexity of 'ft\_volumenormalise' is 51.  
The modified cyclomatic complexity of 'ft\_volumerealign' is 105.  
The modified cyclomatic complexity of 'ft\_volumereslice' is 15.  
The modified cyclomatic complexity of 'ft\_volumesegment' is 100.  
The modified cyclomatic complexity of 'ft\_volumewrite' is 40.  
Function return value might be unset.  
The modified cyclomatic complexity of 'homer2fieldtrip' is 20.  
The modified cyclomatic complexity of 'imotions2fieldtrip' is 32.  
The modified cyclomatic complexity of 'loreta2fieldtrip' is 14.  
The modified cyclomatic complexity of 'nutmeg2fieldtrip' is 56.  
The modified cyclomatic complexity of 'spass2fieldtrip' is 15.  
The modified cyclomatic complexity of 'spm2fieldtrip' is 4.  
The modified cyclomatic complexity of 'xdf2fieldtrip' is 38.