Original version at http://iram-institute.org/medias/uploads/class-single-multiple.pdf

IRAM Memo 2009-6
Read-write optimization in **CLASS**

S. Bardeau[1], J. Pety[1,2], S. Guilloteau[3]

1. IRAM (Grenoble)
2. Observatoire de Paris
3. LAB, Observatoire de Bordeaux

March, 13^{th} 2009
Version 1.0

Abstract

CLASS is a **GILDAS**[1] software for reduction and analysis of (sub)–millimeter spectroscopic data. It is daily used to reduce spectra acquired with the IRAM 30m telescope. **CLASS** is currently used in many other facilities (*e.g.* CSO, HHT, Effelsberg) and it is considered for use by Herschel/HIFI. **CLASS** history started in 1983. As a consequence, it was written in FORTRAN 77 and tailored to reduce pointed observations.

When **CLASS** was re-written in Fortran 90, it was decided to split On-The-Fly spectra into one observation each[2]. This eases spectra handling from the user side, but, of course, multiplies the number of observations in a single file (typically by a factor ~ 1000). The increasing number of observations (especially with the OTF acquisition schemes) and number of channels of the back-ends produces larger and larger files. **CLASS** has thus been revised with new file types and optimized with a faster algorithm in order to ensure a maximal data rate from the software side. We will expose here this new algorithm, and show that the major limitation for writing **CLASS** large files is currently on the hardware side.

Keywords: On-The-Fly, **CLASS** file format, data rate, user and system times.
Related documents: IRAM memo 2005-1: *Improved OTF support*

[1] http://www.iram.fr/GILDAS
[2] see IRAM memo 2005-1: *Improved OTF support* at http://iram-institute.org/medias/uploads/class-evol1.pdf

Contents

1 Improved data format and algorithm

1.1 Historical status

In the original **CLASS** file format, any observation could have different versions. When writing an observation N to a **CLASS** file, the algorithm had to reread the $N-1$ observations and all their versions already written and check if a previous version of the same observation was already present. If yes, the version number was increased. This is typically a N^2 algorithm. On latest modern computer, this started becoming a limiting factor (compared to the OS/hardware incompressible time) when writing more than 10^5 spectra per file (see section 2).

1.2 Current version

On september, 29th 2008, an improvement was introduced on the development branch of **CLASS** in its data format. It was officially released in gildas dec08 version. There are now two types for the **CLASS** files, both based on the standard **CLASS** data format. Each type is associated to a new syntax of the command `FILE OUT`:

- `FILE OUT Name MULTIPLE`: File of type MULTIPLE allows to have several version of the same spectrum (same ObservationNumber).
- `FILE OUT Name SINGLE`: For file of type SINGLE, the ObservationNumber is a unique identifier of the spectrum: only one version exists.

The MULTIPLE type is the "historical" one for **CLASS**, but it has been improved, though. Inserting a new observation or a new observation version is done in a sorted list, which is a $N \times \log N$ algorithm (behaving as N when N is large enough). The counterpart is that the list of observations is sorted when opening an output file with MULTIPLE type, but this is a minor cost compared to the subsequent gain.

The SINGLE type has been introduced to simplify and speed up On-The-Fly processing. When writing an observation, its number is automatically set such as it is unique (or it must not exist if it is provided). With this new feature, the algorithm has a N dependency. This can be observed in the benchmarks performed hereafter.

Another benefit of this new type is that merging two set of observations in a SINGLE file ensures that all observations have a different number. With the historical status, or with the MULTIPLE type, two different spectra with the same number are merged under this number but with different version.

1.3 Cross-use of the different types

1.3.1 New types

- Re-opening an output file for appending new observations preserves its SINGLE or MULTIPLE type.
- Re-opening a MULTIPLE file for input+output is allowed, since several versions of the same observation is allowed.
- Re-opening a SINGLE file for input+output is not allowed: writing out a new version of an input spectrum is explicitely forbidden by the SINGLE type.

1.3.2 Backward compatibility

Opening for output a new file with the syntax `FILE OUT NEW` is now forbidden. This syntax is obsolete and produces an error.

Backward compatibility is ensured for reading and writing in old files (created with an old version of **CLASS** with the syntax `FILE OUT NEW`): they are transparently opened with the MULTIPLE type.

1.3.3 Transition

During a transition time, users may encounter files with the new types (*e.g.* coming from the telescope) in their old versions of **CLASS** (*e.g.* sep08 or older). In such a case, MULTIPLE files are recognized, as old type. On the contrary, SINGLE files are not recognized and users must upgrade their **CLASS** version to a newer one.

2 Benchmark

2.1 Method

		$N_{spectra}$											
		256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
$N_{channels}$	128	0.5	0.8	1.3	2.3	4.6	9.2	18.1	36.1	72.2	144.3	288.4	576.7
	256	0.7	1.0	1.8	3.3	6.6	13.2	26.1	52.1	104.2	208.4	416.5	833.0
	512	0.9	1.5	2.8	5.3	10.6	21.2	42.1	84.1	168.3	336.5	672.8	**1.3**
	1024	1.4	2.6	4.8	9.3	18.6	37.2	74.1	148.2	296.4	592.8	**1.2**	**2.3**
	2048	2.4	4.6	8.8	17.3	34.6	69.2	138.2	276.3	552.7	**1.1**	**2.2**	**4.3**
	4096	4.4	8.6	16.8	33.3	66.7	133.3	266.3	532.6	**1.0**	**2.1**	**4.2**	**8.3**
	8192	8.5	16.6	32.9	65.4	130.7	261.5	522.6	**1.0**	**2.0**	**4.1**	**8.2**	**16.3**
	16384	16.5	32.6	64.9	129.5	258.9	517.7	**1.0**	**2.0**	**4.0**	**8.1**	**16.2**	**32.3**
	32768	32.6	64.7	129.1	257.7	515.2	**1.0**	**2.0**	**4.0**	**8.0**	**16.1**	**32.2**	**64.4**
	65536	64.7	128.9	257.3	514.1	**1.0**	**2.0**	**4.0**	**8.0**	**16.1**	**32.1**	**64.2**	-
	131072	129.1	257.3	513.8	**1.0**	**2.0**	**4.0**	**8.0**	**16.0**	**32.1**	**64.1**	-	-
	262144	257.7	514.1	**1.0**	**2.0**	**4.0**	**8.0**	**16.0**	**32.1**	**64.1**	-	-	-
	524288	514.9	**1.0**	**2.0**	**4.0**	**8.0**	**16.0**	**32.0**	**64.1**	-	-	-	-
	1048576	**1.0**	**2.0**	**4.0**	**8.0**	**16.0**	**32.0**	**64.1**	-	-	-	-	-

Table 1: **CLASS** file size for several combinations of number of spectra and number of channels per spectrum. Units are Megabytes (resp. Gigabytes) for regular font (resp. bold). Files are of type `SINGLE`. The missing combinations were not performed because of the prohibitive file size.

This benchmark measures the WRITE'ing capabilities of **CLASS**. A new **CLASS** file is opened in the desired mode. A spectrum containing $N_{channels}$ is generated thanks to the command `ANALYSE\MODEL`, and is written $N_{spectra}$ times in the file. The USER and SYSTEM times (described below) are measured at typical stages. This is repeated for several values of $N_{channels}$ and for all output filetypes. Table 1 shows the combinations of $N_{channels}$ and $N_{spectra}$ which have been benchmarked.

The tests were realized on an IRAM machine (64 bits, Fedora 6) with **CLASS** compiled with the `ifort` compiler (version 10), writing on a local hard drive. For the old syntax `FILE OUT NEW`, **CLASS** version of the 22-sep-2008 was used. For the new syntax `FILE OUT SINGLE|MULTIPLE`, current version of **CLASS** was used.

The different figures in this document aim to compare the performances of **CLASS** when writing old and new file types. Nevertheless, the SINGLE and MULTIPLE types provide the same level of performance. The benchmark for the NEW, SINGLE, and MULTIPLE types are thus presented only on Fig. 1. Later on, only the NEW and SINGLE types are compared.

2.2 File size

One important parameter to keep in mind is the output file size which becomes significant for several combinations of number of spectra and number of channels per spectrum. Table 1 shows the observed file

size for the benchmarked values of these parameters. Typically, the file weights:

- 1 MByte if $N_{spectra} \times N_{channels} \sim 130,000$
- 10 MBytes if $N_{spectra} \times N_{channels} \sim 2,000,000$
- 100 MBytes if $N_{spectra} \times N_{channels} \sim 25,000,000$
- 1 GByte if $N_{spectra} \times N_{channels} \sim 270,000,000$
- 10 GBytes if $N_{spectra} \times N_{channels} \sim 2,700,000,000$

2.3 Times

2.3.1 USER time

The USER time is the time spent in **CLASS** and Gildas code in general. **CLASS** developpers can act on this time by improving the algorithms.

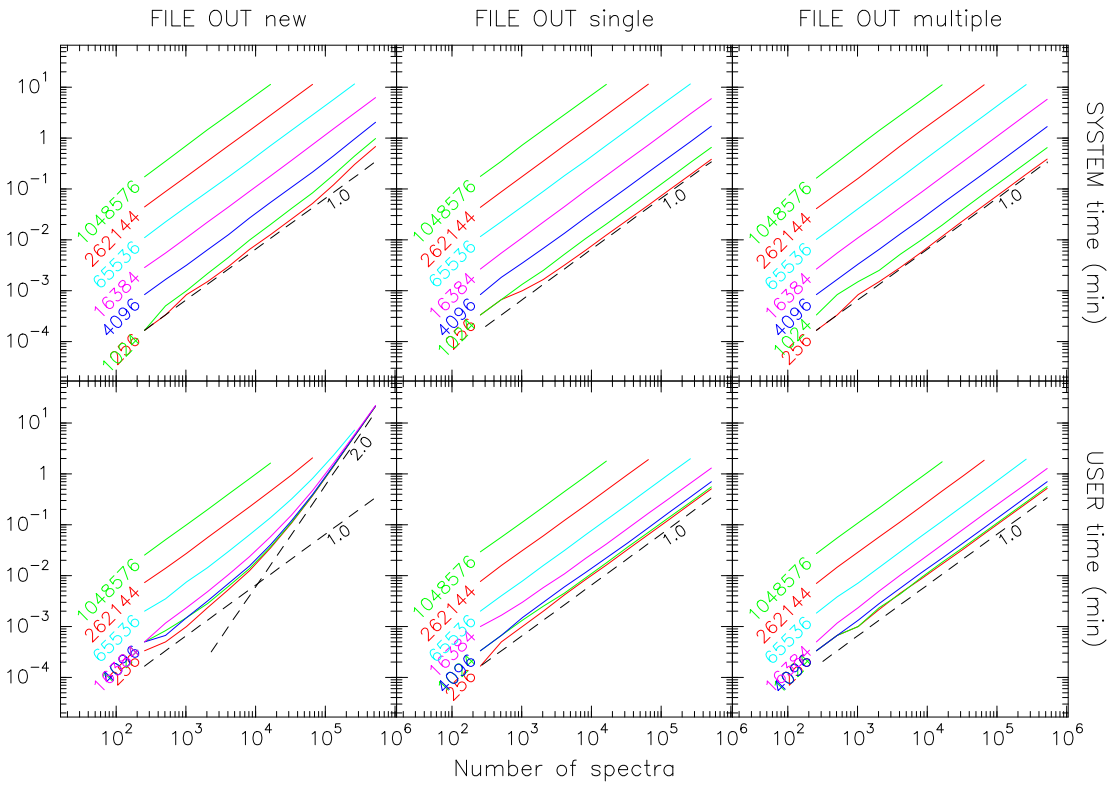

Figure 1: SYSTEM and USER times for the three **CLASS** output files type: `NEW` (old type), `SINGLE` and `MULTIPLE` (new type). Numbers indicate the number of channels per spectrum. Dashed lines are power law functions ($y \propto x^{\alpha}$) with their exponent.

Fig. 1 displays the USER time as a function of the number of spectra written, for several number of channels per spectrum and for all the possible file types. We can observe that:

- with the NEW (old) file type, for more than $\sim 10^4$ spectra per file, the USER time was diverging. We can notice that the limit functions were not depending on the number of channels (lines converge), and we can observe the N^2 dependency predicted at section 1.1.

- the USER time was considerably improved thanks to the new file types. For high number of spectra, it is a power law which power was decreased from 2 to 1, as expected in section 1.2.

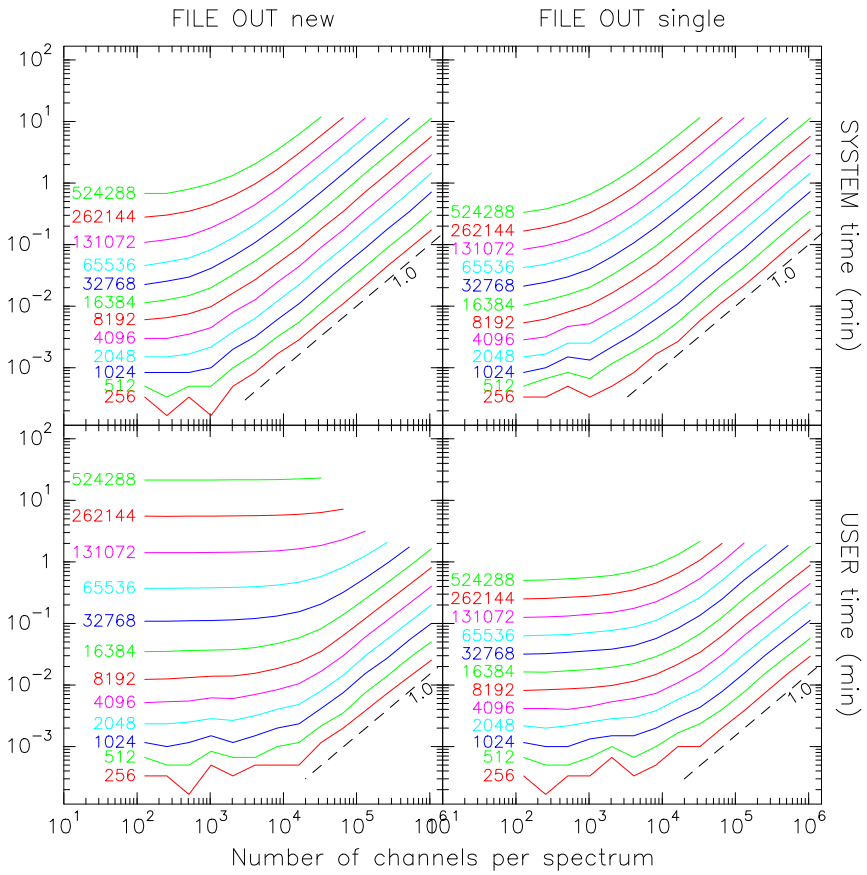

Figure 2: USER time for two **CLASS** output files type, but with time vs number of channels per spectrum. Left: keyword `NEW` (old type), right: `SINGLE` (new type). Numbers indicate the number of spectra written, and the dashed lines a linear increase of the time with the number of channels per spectrum.

Fig. 2 shows the same USER time values, but the number of channels per spectrum is now used as X-axis, for different number of spectra written. We can confirm that:

- the USER time is proportional to the number of channels, when this number is large enough,
- with the SINGLE and MULTIPLE file types, USER time is much lower than with the NEW file type.

2.3.2 SYSTEM time

The SYSTEM time is the time spent in system Input/Output. In this context, it is roughly the time the Operating System spends to write the file on buffer/disk.

From fig. 1 and fig. 2 we can see that:

- the SYSTEM time has not changed between old and new **CLASS** output file types. This was expected since the size of the output file has not significatively changed.
- the SYSTEM time is proportional to the number of spectra written.

- the SYSTEM time is proportional to the number of channel per spectrum, when this number is high (typically more 1024 channels)
- on the other hand, there seems to be a threshold under which we cannot fall, e.g. writing spectra with 128 or 256 channels costs roughly the same SYSTEM time.

Finally, SYSTEM time is proportional to the amount of data written, and thus the file size. This is particularly true when this amount is high and hides over minor system actions.

2.3.3 USER time vs SYSTEM time

We aim to track here where are the limitations depending on the context (number of channels per spectrum, number of spectrum per file).

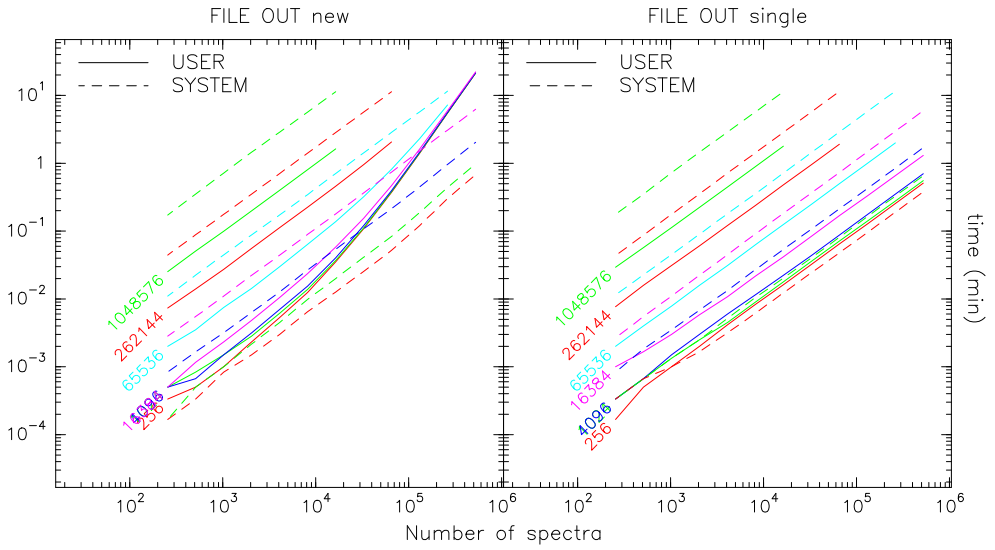

Figure 3: USER vs SYSTEM time for the NEW and SINGLE **CLASS** output file types.

Fig. 3 shows that:

- with the NEW file type, there was always a point where the USER time was equal to the SYSTEM time when writing more and more spectra (e.g. they were equal for 30,000 spectra of 4096 channels).
- in the same context, USER time became a limiting factor a high number of spectra, e.g. it was $\sim$ 1 order of magnitude higher than the SYSTEM time when writing 524,288 spectra with 4,096 channels.
- the above point is not true anymore with the current version of **CLASS**, and the USER time is allways lower than the SYSTEM time (typically a half with 4,096 channels per spectrum).

We can conclude that now (on the test machine), the SYSTEM time is the limiting factor. The data rate is limited by both the hard disk driver (software) and its writing speed (hardware). There is no easy way to act on this time except from using a more performant hardware.

2.4 Data rate

Finally, one of the main purpose of the new **CLASS** file types was to increase the data rate to the output file. Fig. 4 shows this rate for the old and new types. We can see (top-left) that, with the old file type,

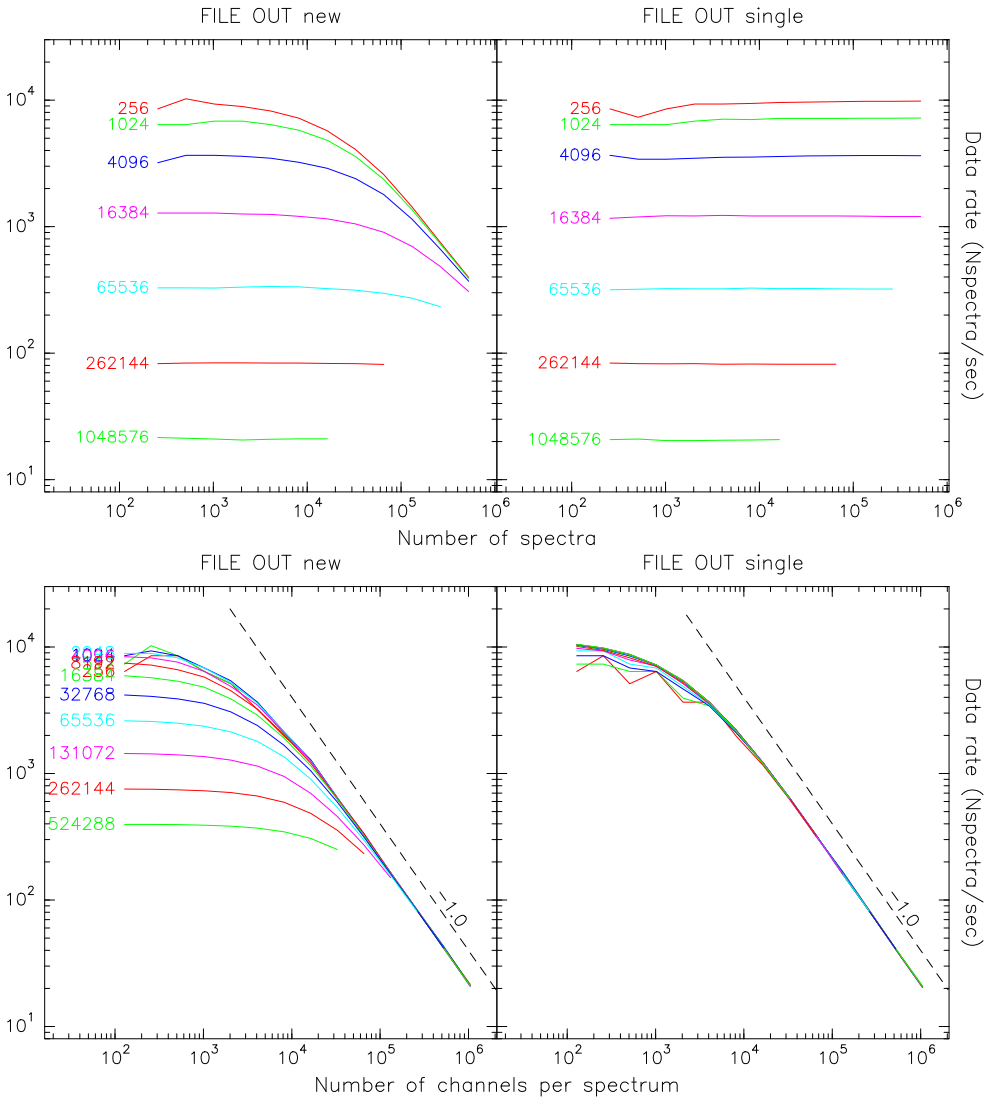

Figure 4: Top: data rate (number of spectra written in file per second) as a function of the number of spectra, for two **CLASS** output file types. Numbers indicate the number of channels per spectrum. Bottom: data rate as a function of the number of channels per spectrum. Numbers indicate the number of spectra written, and the dashed line a linear decrease of the rate with the number of channels. Left: `NEW` (old file type), right: `SINGLE` (new type).

this rate was decreasing when writing more and more spectra, e.g. (with 16384 channels per spectrum) falling from 1300 spectra written per second (for the first thousands of spectra) to 300 when writing the $\sim$ 500,000ths. This is fixed with the new file types: the data rate is constant over the number of spectra written (top-right).

Again on the figure (bottom-right), with the new file types, we can observe that the data rate is allways the same for a given number of channels. For a low number of channels ($< 10,000$), we can observe that the USER time has an effect on this rate. But for higher number of channels, the rate is dominated by the SYSTEM time: it decreases linearly with the amount of data written. Typical values are:

- 10,000 spectra per second with 128 channels,
- 1,200 spectra per second with 16,384 channels,
- 20 spectra per second with 1,048,576 channels

3 Acknowledgments

We wish to thank P. Hily-Blant for useful discussion about the improvement of the WRITE algorithm and its implementation.