BuildFilter

BuildFilter builds kMer filters (.mer files) for use by FilterReads. These kMer filters are simply sets of canonical, binary-packed kMers tiled from a set of (reference) sequences.

BuildFilter is a command-line program with the following cryptic usage hint:

```
BuildFilter -k merSize [+/-lcf] [-s] [kMersFN] seqsFN
```

A typical run of BuildFilter is…

```
BuildFilter -k 20 +lcf RDPv16+RefSeq_5-18_16S_NC_20.mer RDPv16+RefSeq_5-18_16S_NC.fa
```

…which will tile the 16S reference sequences found in `RDPv16+RefSeq_5-18_16S_NC.fa` for 20-mers, and turn these into a set of distinct, canonical 20-mers.

Canonical means that the reverse-complement of every kMer is calculated, and only the lexicographically lowest of the kMer and its RC is kept to represent the kMer. For example, with a kMer of ACACACACACA and its RC of TGTGTGTGTGT, the ACACACACACA form will be kept and used to represent both variants. Keeping only canonical kMers makes it much easier to handle sequence data which could have been derived from either strand of a DNA molecule.

BuildFilter can also attempt to recognise and discard low-complexity reads/kMers. These are often not very useful or informative, and can result in overly accepting filters.

The full set of BuildFilter parameters is:

`-k kMer_length`	Length of kMers in the filter. This would normally be at least 20 to take advantage of the distinctiveness of such kMers. kMers cannot be longer than 32 bases.
`+lcf`	Turns on low-complexity filtering of the kMers/reads. This is the default..
`-lcf`	Turns off low-complexity filtering of the kMers/reads. Not usually desirable but needed in cases such as iterative gene-filtering.
`-s`	Search for matching file names in any subdirectories as well as in the current directory. Default is to only look for matching file names in the current directory.
`kMersFN`	(optional) Name for the generated kMers file. If no kMersFn name is given, one is constructed from the first (and only in this case) seqsFN by adding _kMer_length.mer at its end.
`seqsFNP`	List of file names/patterns of sequences to be tiled for kMers. e.g. `RDPv16+RefSeq_5-18_16S_NC.fa` There can be any number of file names (or file name patterns), and all of these will be tiled to generate a single kMer file.

BuildFilter is written in C# and is provided pre-compiled for Windows, OSX and common Linux variants. These pre-compiled code files should be stand-alone and should not require the installation of any additional run-time libraries.

You can compile BuildFilter yourself using Mono (under OSX and Linux), or under Visual Studio on Windows. You'll need the `mono-devel` package installed on Linux or OSX.

The BuildFilter code itself is in Program.cs in this directory, and you'll also need to download kMers.cs, SeqFiles.cs and Sequence.cs from WorkingDocsCoreLibrary. One simple way of compiling Kelpie (if you need to) is to make a directory containing these 4 .cs files and then run `'csc /out:BuildFilter.exe /target:exe *.cs'` from this directory. Earlier versions of mono use 'msc' rather than 'csc' but the syntax of the compilation command is the same. The C# compiler will produce the executable `BuildFilter.exe`. This code file can be run natively on Windows or via `mono BuildFilter` on other platforms.

The native Linux and OSX executables provided with the package were produced by cross-compiling `BuildFilter.exe` with `mkbundle`, targeting various Linux and OSX releases.

```
mkbundle -o ubuntu-18.04\ BuildFilter BuildFilter.exe --simple --cross mono-5.12.0-ubuntu-18.04-x64
```

You can also use `mkbundle` to generate a native executable for your current system. You'll need to know where the `mono-devel` installation put the `machine.config` and the `mono` libraries.

```
mkbundle -o BuildFilter --simple BuildFilter.exe --machine-config /etc/mono/4.5/machine.config -L /apps/mono/5.4.1.7/lib/mono/4.5
```

The mkbundle command has been changing with recent mono releases, and you may need to try other variants, such as:

```
mkbundle -o BuildFilter --cross default BuildFilter.exe
```

It is expected that all use of mono will be replaced late in 2018 once .NET Core 3 arrives, with direct multi-platform code generation.