.

open-Source Media Interpretation by Large feature-space Extraction

Version 2.3, November 2016

Main authors: Florian Eyben, Felix Weninger, Martin Wöllmer, Björn Schuller

E-mails: fe, fw, mw, bs at audeering.com

audEERING GmbH
D-82205 Gilching, Germany
`http://www.audeering.com/`

The official openSMILE homepage can be found at: `http://opensmile.audeering.com/`

This documentation was created by Florian Eyben. Contributions for Android from Gerhard Hagerer.

Contents

Chapter 1

About openSMILE

We start introducing openSMILE by addressing two important questions for users who are new to openSMILE : *What is openSMILE ?* and *Who needs openSMILE ?*. If you want to start using openSMILE right away, then you should start reading section 2, or section 2.3 if you have already managed to install openSMILE.

1.1 What is openSMILE?

The Munich open-Source Media Interpretation by Large feature-space Extraction (openSMILE) toolkit is a modular and flexible feature extractor for signal processing and machine learning applications. The primary focus is clearly put on audio-signal features. However, due to their high degree of abstraction, openSMILE components can also be used to analyse signals from other modalities, such as physiological signals, visual signals, and other physical sensors, given suitable input components. It is written purely in C++, has a fast, efficient, and flexible architecture, and runs on various main-stream platforms such as Linux, Windows, and MacOS. openSMILE is designed for real-time online processing, but can also be used off-line in batch mode for processing of large data-sets. This is a feature rarely found in related feature extraction software. Most of related projects are designed for off-line extraction and require the whole input to be present. openSMILE can extract features incrementally as new data arrives. By using the PortAudio[1] library, openSMILE features platform independent live audio input and live audio playback, which enabled the extraction of audio features in real-time.

To facilitate interoperability, openSMILE supports reading and writing of various data formats commonly used in the field of data mining and machine learning. These formats include PCM WAVE for audio files, CSV (Comma Separated Value, spreadsheet format) and ARFF (Weka Data Mining) for text-based data files, HTK (Hidden-Markov Toolkit) parameter files, and a simple binary float matrix format for binary feature data.

Using the open-source software gnuplot[2], extracted features which are dumped to files can be visualised. A strength of openSMILE , due to its highly modular architecture is that almost all intermediate data which is generated during the feature extraction process (such as windowed audio data, spectra, etc.) can be accessed and saved to files for visualisation or further processing.

[1] `http://www.portaudio.com`
[2] `http://www.gnuplot.info/`

1.2 Who needs openSMILE?

openSMILE is intended to be used for research applications, demonstrators, and prototypes in the first place. Thus, the target group of users is researchers and system developers. Due to its compact code and modular architecture, using openSMILE for the final product is also considerable. However, we would like to stress that openSMILE is distributed under a research only license (see the next section for details).

Currently, openSMILE is used by researchers and companies all around the world, which are working in the field of speech recognition (feature extraction front-end, keyword spotting, etc.), the area of affective computing (emotion recognition, affect sensitive virtual agents, etc.), and Music Information Retrieval (chord labelling, beat tracking, onset detection etc.). With the 2.0 open-source release we target the wider multi-media community by including the popular openCV library for video processing and video feature extraction.

1.3 Licensing

openSMILE follows a dual-licensing model. Since the main goal of the project is a widespread use of the software to facilitate research in the field of machine learning from audio-visual signals, the source code and the binaries are freely available for private, research, and educational use under an open-source license. It is not allowed to use the open-source version of openSMILE for or in any sort of commercial product. Fundamental research in companies, for example, is permitted, but if a product is the result of the research, we require you to buy a commercial development license. Contact us at info@audeering.com (or visit us at `http://www.audeering.com`) for further information.

1.4 History

openSMILE was originally created in the scope of the European EU-FP7 research project SEMAINE (`http://www.semaine-project.eu`) and is used there as the acoustic emotion recognition engine and keyword spotter in a real-time affective dialogue system. To serve the research community open-source releases of openSMILE were made independently of the main project's code releases.

The first publicly available version of openSMILE was contained in the first Emotion and Affect recognition toolkit openEAR as the feature extraction core. openEAR was introduced at the Affective Computing and Intelligent Interaction (ACII) conference in 2009. One year later the first independent release of openSMILE (version 1.0.0) was made, which aimed at reaching a wider community of audio analysis researchers. It was presented at ACM Multimedia 2010 in the Open-Source Software Challenge. This first release was followed by a small bugfix release (1.0.1) shortly. Since then development has taken place in the subversion repository on sourceforge. Since 2011 the development was continued in a private repository due to various internal and third party project licensing issues.

openSMILE 2.0 (rc1) is the next major release after the 1.0.1 version and contains the latest code of the core components with a long list of bugfixes, new components as well as improved old components, extended documentation, a restructured source tree and new major functionality such as a multi-pass mode and support for synchronised audio-visual feature extraction based on openCV.

Version 2.1 contains further fixes, improved backwards compatibility of the standard INTERSPEECH challenge parameter sets, support for reading JSON neural network files created

with the CURRENNT toolkit, a F0 harmonics component, and a fast fast linear SVM sink component for integrating models trained with WEKA SMO, as well as some other minor new components and features. It is the first version published and supported by audEERING.

1.5 Capabilities - Overview

This section gives a brief summary on openSMILE's capabilities. The capabilities are distinguished by the following categories: data input, signal processing, general data processing, low-level audio features, functionals, classifiers and other components, data output, and other capabilities.

Data input: openSMILE can read data from the following file formats:

- RIFF-WAVE (PCM) (for MP3, MP4, OGG, etc. a converter needs to be used)
- Comma Separated Value (CSV)
- HTK parameter files
- WEKA's ARFF format.
- Video streams via openCV.

Additionally, live recording of audio from any PC sound-card is supported via the PortAudio library. For generating white noise, sinusoidal tones, and constant values a signal Generator is provided.

Signal Processing: The following functionality is provided for general signal processing or signal pre-processing (prior to feature extraction):

- Windowing-functions (Rectangular, Hamming, Hann (raised cosine), Gauss, Sine, Triangular, Bartlett, Bartlett-Hann, Blackmann, Blackmann-Harris, Lanczos)
- Pre-/De-emphasis (i.e. 1st order high/low-pass)
- Re-sampling (spectral domain algorithm)
- FFT (magnitude, phase, complex) and inverse
- Scaling of spectral axis via spline interpolation (open-source version only)
- dbA weighting of magnitude spectrum
- Autocorrelation function (ACF) (via IFFT of power spectrum)
- Average magnitude difference function (AMDF)

Data Processing: openSMILE can perform a number of operations for feature normalisation, modification, and differentiation:

- Mean-Variance normalisation (off-line and on-line)
- Range normalisation (off-line and on-line)
- Delta-Regression coefficients (and simple differential)

- Weighted Differential as in [SER07]
- Various vector operations: length, element-wise addition, multiplication, logarithm, and power.
- Moving average filter for smoothing of contour over time.

Audio features (low-level): The following (audio specific) low-level descriptors can be computed by openSMILE :

- Frame Energy
- Frame Intensity / Loudness (approximation)
- Critical Band spectra (Mel/Bark/Octave, triangular masking filters)
- Mel-/Bark-Frequency-Cepstral Coefficients (MFCC)
- Auditory Spectra
- Loudness approximated from auditory spectra.
- Perceptual Linear Predictive (PLP) Coefficients
- Perceptual Linear Predictive Cepstral Coefficients (PLP-CC)
- Linear Predictive Coefficients (LPC)
- Line Spectral Pairs (LSP, aka. LSF)
- Fundamental Frequency (via ACF/Cepstrum method and via Subharmonic-Summation (SHS))
- Probability of Voicing from ACF and SHS spectrum peak
- Voice-Quality: Jitter and Shimmer
- Formant frequencies and bandwidths
- Zero- and Mean-Crossing rate
- Spectral features (arbitrary band energies, roll-off points, centroid, entropy, maxpos, minpos, variance (=spread), skewness, kurtosis, slope)
- Psychoacoustic sharpness, spectral harmonicity
- CHROMA (octave warped semitone spectra) and CENS features (energy normalised and smoothed CHROMA)
- CHROMA-derived Features for Chord and Key recognition
- F0 Harmonics ratios

Video features (low-level): The following video low-level descriptors can be currently computed by openSMILE , based on the openCV library:

- HSV colour histograms
- Local binary patterns (LBP)
- LBP histograms
- Optical flow and optical flow histograms
- Face detection: all these features can be extracted from an automatically detected facial region, or from the full image.

Functionals: In order to map contours of audio and video low-level descriptors onto a vector of fixed dimensionality, the following functionals can be applied:

- Extreme values and positions
- Means (arithmetic, quadratic, geometric)
- Moments (standard deviation, variance, kurtosis, skewness)
- Percentiles and percentile ranges
- Regression (linear and quadratic approximation, regression error)
- Centroid
- Peaks
- Segments
- Sample values
- Times/durations
- Onsets/Offsets
- Discrete Cosine Transformation (DCT)
- Zero-Crossings
- Linear Predictive Coding (LPC) coefficients and gain

Classifiers and other components: Live demonstrators for audio processing tasks often require segmentation of the audio stream. openSMILE provides voice activity detection algorithms for this purpose, and a turn detector. For incrementally classifying the features extracted from the segments, Support Vector Machines are implemented using the LibSVM library.

- Voice Activity Detection based on Fuzzy Logic
- Voice Activity Detection based on LSTM-RNN with pre-trained models
- Turn-/Speech-segment detector
- LibSVM (on-line)

- SVM sink (for loading linear kernel WEKA SMO models)
- GMM (experimental implementation from eNTERFACE'12 project, to be release soon)
- LSTM-RNN (Neural Network) classifier which can load RNNLIB and CURRENNT nets
- Speech Emotion recognition pre-trained models (openEAR)

Data output: For writing data data to files, the same formats as on the input side are supported, except for an additional binary matrix format:

- RIFF-WAVE (PCM uncompressed audio)
- Comma Separated Value (CSV)
- HTK parameter file
- WEKA ARFF file
- LibSVM feature file format
- Binary float matrix format

Additionally, live audio playback is supported via the Portaudio library.

Other capabilites : Besides input, signal processing, feature extraction, functionals, and output components, openSMILE comes with a few other capabilites (to avoid confusion, we do not use the term 'features' here), which make using openSMILE easy and versatile:

Multi-threading Independent components can be run in parallel to make use of multiple CPUs or CPU cores and thus speed up feature extraction where time is critical.

Plugin-support Additional components can be built as shared libraries (or DLLs on windows) linked against openSMILE's core API library. Such plugins are automatically detected during initialisation of the program, if they are placed in the `plugins` subdirectory.

Extensive logging Log messages are handled by a smileLogger component, which currently is capable of saving the log messages to a file and printing them to the standard error console. The detail of the messages can be controlled by setting the log-level. For easier interpretation of the messages, the types *Message* (MSG), *Warning* (WRN), *Error* (ERR), and *Debug* (DBG) are distinguished.

Flexible configuration openSMILE can be fully configured via one single text based configuration file. This file is kept in a simple, yet very powerful, property file format. Thereby each component has it's own section, and all components can be connected via their link to a central data memory component. The configuration file even allows for defining custom command-line options (e.g. for input and output files), and including other configuration files to build configurations with modular configuration blocks. The name of the configuration file to include can even be specified on the commandline, allowing maximum flexibility in scripting.

Incremental processing All components in openSMILE follow strict guidelines to meet the requirements of incremental processing. It is not allowed to require access to the full input sequence and seek back and forth within the sequence, for example. Principally each

component must be able to process its data frame by frame or at least as soon as possible. Some exceptions to this rule have been granted for components which are only used during off-line feature extraction, such as components for overall mean normalisation.

Multi-pass processing in batch mode For some tasks multi-pass processing is required, which obviously can only be applied in off-line (or buffered) mode. openSMILE since version 2.0 supports mutli-pass processing for all existing components.

TCP/IP network support In the commercial version a remote data I/O API is available, which allows to send and receive data (features and messages) from openSMILE, as well as remote control of openSMILE (pause/resume/restart and send config) via a TCP/IP network connection.

Chapter 2

Using openSMILE

Now we describe how to get started with using openSMILE . First, we will explain how to obtain and install openSMILE . If you already have a working installation of openSMILE you can skip directly to section 2.3, where we explain how to use openSMILE for your first feature extraction task. We then give an overview on openSMILE's architecture in section 2.4. This will help you to understand what is going on inside openSMILE , and why certain things are easy, while others may be tricky to do. Next, to make full use of openSMILE's capabilities it is important that you learn to write openSMILE configuration files. Section 4.2 will explain all necessary aspects. Finally, to make your life a bit simpler and to provide common feature sets to the research community for various tasks, some example configuration files are provided. These are explained in section 2.5. Included are al the baseline feature sets for the INTERSPEECH 2009–2013 affect and paralinguistics challenges. In section 2.6 we will teach you how to use the PortAudio interface components to set up a simple audio recorder and player as well as a full live feature extraction system. Section 2.7 will help you to get started with video feature extraction and synchronised audio-visual feature extraction. How you can plot the extracted features using the open-source tool gnuplot, is explained in section 2.8.

2.1 Obtaining and Installing openSMILE

Note for the impatient: If you have already downloaded openSMILE, and are an expert at compiling software on Windows and/or Linux, you may skip to section 2.2.1, which contains the quick-start compilation instructions.

The latest stable release of openSMILE can be found at `http://opensmile.audeering.com/`.

Major releases include binaries for Windows (32-bit) and Linux (64-bit), as well as Android ARM (Since 2.1, android-10). All releases contain the source code, which can be compiled on Linux, Max OS, Windows, and for Android. This is the recommended way for Linux/Unix and Mac OS systems, and is mandatory if you require live audio recording/playback through portAudio. A release package contains the statically linked main executable `SMILExtract` for Linux systems and a `SMILExtract_Release.exe` and `openSmileLib_Release.dll` for Windows systems in the `bin/` folder, example configuration files in the `config/` folder, scripts for visualisation and other tasks such as model-building in the `scripts/` folder, and the source in the `src/` folder.

The binary releases are ready to use out-of-the-box. For Linux, a statically linked standalone binary is provided for 64-bit platforms. It is placed in the `bin` directory. Copy the executable

that matches your platform to a folder in your path (on Linux e.g. /usr/local/bin, on Windows e.g. `C:\Windows\`). Please be sure to also copy all DLL files to that path on Windows systems. Executables which are linked against PortAudio have a `PA` appended to their filenames (they are only provided for Windows). To test if your release works type

```
SMILExtract -h
```

in the shell prompt on Unix systems or

```
SMILExtract -h
```

in the Windows command-line prompt. If you see the usage information everything is working.

For Windows binaries with PortAudio support (PA suffix) and binaries without portaudio support are provided. A compiled portaudio DLL is also provided, which is linked against the Windows Media Extensions API. All these executables can be found in the `bin` subdirectory and your version of choice must be copied to `SMILExtract.ext` in the top-level directory of the openSMILE distribution in order to be able to execute the example commands in this tutorial as they are printed (You must also copy the corresponding .dll to the top-level directory, however, without renaming it!).

Note for Linux: The Linux binaries contained in the releases are statically linked binaries, i.e. the shared API *libopensmile* is linked into the binary. The binaries only depend on libc6 and pthreads. The downside of this method is that you cannot use binary plugins with these binaries! In order to use plugins, you must compile the source code to obtain a binary linked dynamically to libopensmile (see section 2.2.2). As no binary release with PortAudio support is provided for Linux, in order to use PortAudio audio recording/playback, you must compile from the source code (see sections 2.2.2 and 2.2.3).

Binaries for ARM Android platforms are provided for version 2.1 and above in the `bin/android-*/.libs` folder. You can use the .libs folder directly in Android eclipse projects and link against it. More dcoumentation for Android building will follow.

No binaries are provided for openSMILE with openCV support. In order to use video features, you must compile from source on a machine with openCV installed. Compilitation on both Windows and Linux is supported. See sections 2.2.4 and 2.2.7. If you have obtained a source only release, read the next section on how to compile and install it.

2.2 Compiling the openSMILE source code

The core of openSMILE compiles without any third-party dependencies, except for *pthreads* on Unix systems. The core version is a command-line feature extractor only. You can not do live audio recording/playback with this version. In order to compile with live audio support, you need the PortAudio[1] library. This will add support for audio recording and playback on Linux, Windows, and Mac OS. Please refer to section 2.2.3 for instructions on how to compile with PortAudio support on Linux and section 2.2.6 for Windows. For openCV support, please refer to sections 2.2.4 and 2.2.7. However, be sure to read the compilation instructions for the standalone version in sections 2.2.2 and 2.2.5 first, as the following sections assume you are familiar with the basics.

2.2.1 Build instructions for the impatient

This section provides quick start build instructions, for people who are familiar with building applications from source on Unix and Windows. If these instructions don't work for you, if you

[1] Available at: `http://www.portaudio.com/`

get build errors, or you require more detailed information, then please refer to the following sections for more detailed instructions, especially for Unix build environments.

We will always distinguish between building with PortAudio support for live audio playback and recording, building with openCV support for video features, and building the standalone version without any third-party dependencies. Building without PortAudio and openCV is easier, as you will get a single statically-linked executable, which is sufficient for all off-line command-line feature extraction tasks.

Unix The very short version: Build scripts are provided. Use `sh buildStandalone.sh` or `sh buildWithPortAudio.sh` (both take the -h option to show a usage). The main binary is installed in `inst/bin/` and called SMILExtract. Run `inst/bin/SMILExtract -h` to see an online help. Libraries are installed in `inst/lib`. For openCV support, pass the path of your openCV installation to the buildWithPortAudio.sh script using the -o option.
The little longer version of the short build instructions: unpack the openSMILE archive by typing:

```
tar -zxvf openSMILE-2.x.x.tar.gz
```

This creates a folder called `openSMILE-2.x.x`. Change to this directory by typing:

```
cd openSMILE-2.x.x
```

Then (assuming you have a running build system installed (autotools, libtool, make, gcc and g++ compiler, ...) and have a bash compatible shell) all you need to do is type:

```
bash buildStandalone.sh
```

or, if the above doesn't work:

```
sh buildStandalone.sh
```

This will configure, build, and install the openSMILE binary `SMILExtract` to the `inst/bin` subdirectory. Add this directory to your path, or copy `inst/bin/SMILExtract` to a directory in your search path. Optionally you can pass an install prefix path to the script as a parameter:

```
sh buildStandalone.sh -p /my/path/to/install/to
```

To compile openSMILE with PortAudio support, *if PortAudio is NOT installed on your system* type (optionally specifying an installation prefix for portaudio and openSMILE as first parameter):

```
sh buildWithPortAudio.sh [-p install-prefix-path]
```

A PortAudio snapshot is included in the `thirdparty` subdirectory. This will be unpacked, configured, and installed into the `thirdparty` directory, which is on the same level as the `opensmile` main directory. openSMILE will then be configured to use this installation. The built executable is called `SMILExtract` and is found in the `inst/bin` sub-directory. Please note, that in this case, it is a wrapper script, which sets up the library path and calls the actual binary `SMILExtract.bin`. Thus, you can also use this method if you have a locally installed (different) PortAudio. Just be sure to alsways run openSMILE through the wrapper script, and not run the binary directly (this will use the system wide, possibly incompatible, PortAudio library).

Please also note that the recommended way of building openSMILE with PortAudio is to use the portaudio.tgz shipped with the openSMILE release. Other versions of portaudio might

be incompatible with openSMILE and might not work correctly (sometimes the PortAudio development snapshots also contain bugs). This is also the only way An obsolete (and currently unsupported) way of compiling with the locally installed portaudio library was provided by builWithInstalledPortaudio.sh. As this script is outdated, we recommend using buildWithPortAudio.sh and removing the linker and include paths which reference the custom portaudio build. The configure script will then automatically detect the system wide installation of portaudio.

Windows

> **Important note for building on Windows:** Some versions of Visual Studio always select the 'Debug' configuration by default instead of the 'Release' configuration. However, you always want to build the 'Release' configuration, unless you are an openSMILE developer. Thus, you mus always select the 'Release' configuration from the drop-down menu, before clicking on 'Build Solution' !!

With version 2.0 we have switched to providing Visual Studio 2010 build files instead of Visual Studio 2005. We do not provide support for any version of visual studio below 2010 anymore.

The very short version of the compile instructions: Open `ide/vs10/openSmile.sln`. Select the configuration you want to build. Release and Debug refer to the standalone versions. The output binary is placed in the msvcbuild folder in the top-level of the distribution. For building with PortAudio support a few more steps are necessary to patch the PortAudio build files. These steps are described in section 2.2.6.
The little longer version of the short build instructions:

Assuming that you have a correctly set up Visual Studio 2010 (or newer) environment, you can open the file `ide/vs10/openSmile.sln`, select the 'Release' configuration, and choose 'Build solution' to build the standalone version of openSMILE for Windows. This will create the command-line utility `SMILExtract.exe` in the msvcbuild directory in the top-level directory of the package, which you can copy to your favourite path or call it directly. For building with PortAudio support a few more steps are necessary to patch the PortAudio build files. These steps are described in section 2.2.6.

2.2.2 Compiling on Linux/Mac

This section describes how to compile and install openSMILE on Unix-like systems step by step (in case the build scripts mentioned in the previous section don't work for you). You need to have the following packages installed: `autotools` (i.e. automake, autoconf, libtool, and m4), `make`, GNU C and C++ compiler `gcc` and `g++`. You will also want to install `perl5` and `gnuplot` in order to run the scripts for visualisation. Please refer to your distribution's documentation on how to install packages. You will also need root privileges to install new packages. We recommend that you use the latest Ubuntu or Debian Linux, where packages can easily be installed using the command `sudo apt-get install package-name`. *Note:* Please be aware that the following instructions assume that you are working with the *bash* shell. If you use a different shell, you might encounter some problems if your shell's syntax differs from bash's.

Start by unpacking the openSMILE package to a directory to which you have write access:

```
tar -zxvf openSMILE-2.x.x.tar.gz
```

Then change to the newly created directory:

```
cd openSMILE-2.x.x/
```

Important: The following is the manual build instructions, which were relevant for version 1.0, but have been replaced by the build scripts (buildStandalone and builWithPortAudio) since version 2.0. Thus, you should prefer the scripts, or read the scripts and modify them, if you need to customize the build process. The following text is only included as a reference and for historical reasons. Refer to the quick build instruction in Section 2.2.1 for using the build scripts.

Next, run the following script *twice* (you may get errors the first time, this is ok):

```
bash autogen.sh
```

Important: You must run `autogen.sh` a second time in order to have all necessary files created! If you do not do so, running `make` after `configure` will fail because `Makefile.in` is not found. If you see warnings in the `autogen.sh` output you can probably ignore them, if you get errors try to run `autogen.sh` a third time.

Note: if you cannot run `./autogen.sh` then run it either as `sh autogen.sh` or change the executable permission using the command `chmod +x autogen.sh`. If you get errors when running this script the second time, your version of autotools might be outdated. Please check that you have at least automake 1.10 and autoconf 2.61 installed (type `autoconf --version` and `automake --version` to obtain the version number).

Now configure openSMILE with

```
./configure
```

to have it installed in the default location /usr or /usr/local (depends on your system), or use the `--prefix` option to specify the installation directory (**important:** you need to use this, if you *don't have root privileges* on your machine):

```
./configure --prefix=/directory/prefix/to/install/to
```

Please make sure you have full write access to the directory you specify, otherwise the `make install` command will fail.

On modern CPUs you can create an optimised executable for your CPU by using the following compiler flags: `-O2 -mfpmath=sse -march=native`. You can pass those flags directly to configure (you may or may not combine this with the `--prefix` option):

```
./configure CXXFLAGS=''-O2 -mfpmath=sse -march=native'' CFLAGS=''-O2
-mfpmath=sse -march=native''
```

Please note that this option is not supported by all compilers.

The default setup will create a `SMILExtract` binary and a `libopensmile.so` shared library. This is usually what you want, especially if you want to use plugins. However, in some cases a portable binary, without library dependencies may be preferred. To create such a statically linked binary pass the following option to the configure script:

```
./configure --enable-static --enable-shard=no
```

Warning: openSMILE plugins will not work with statically linked binaries.

After you have successfully configured openSMILE (i.e. if there were not error messages during configuring - warning messages are fine), you are now ready to compile openSMILE with this command:

```
make -j4 ; make
```

Note: `make -j4` runs 4 compile processes in parallel. This speeds up the compilation process on most machines (also single core). However, running only `make -j4` will result in an error, because `libopensmile` has not been built when `SMILExtract` is build. Thus, you need to run a single `make` again. This should finish without error. If you have trouble with the `-j4` option, simply use `make` without options.

You are now ready to install openSMILE by typing:

```
make install
```

You have to have root privileges to install openSMILE in a standard location (i.e. if you have *not* specified an alternate path to `configure`). It is also possible to run openSMILE without installation directly from the top-level directory of the openSMILE distribution (this should be your current directory at the moment, if you have followed the above steps carefully). In this case you have to prefix the executable with `./` i.e. you have to run `./SMILExtract` instead of `SMILExtract`.

Please note that `make install` currently only installs the openSMILE feature extractor binary `SMILExtract` and the feature extractor's library `libopensmile.so`. Configuration files still remain in the build directory. Therefore, the examples in the following sections will assume that all commands are entered in the top-level directory of the openSMILE distribution.

For splitting openSMILE into an executable and a dynamic library there have been primarily two reasons:

Reusability of source-code and binaries. The openSMILE library contains the API components with all the base classes and the standard set of components distributed with openSMILE. Custom components, or project specific implementations can be linked directly into the SMILExtract executable. Thus, the library can be compiled without any additional third-party dependencies and can be maintained and distributed independently, while other projects using openSMILE can create a GUI frontend, for example, which depends on various GUI libraries, or add components which interface with a middleware, as in the SEMAINE project[2].

Support for linking binary plugins at run-time. Since binary plugins depend on the openSMILE API and various base classes, instances of these base classes may be present only once in the process memory during run-time. This can only be achieved by off-loading these classes to a separate library.

Note: If you have installed openSMILE to a non-default path, you must set your library path to include the newly installed libopensmile before running the SMILExtract binary (replace `/directory/prefix/to/install/to` by the path you have passed to the `--prefix` option of the `configure` script):

```
export LD_LIBRARY_PATH=/directory/prefix/to/install/to/lib
```

You will also need to add the path to the binary to your current `PATH` variable:

```
export PATH="$PATH:/directory/prefix/to/install/to/lib"
```

Attention: You need to do this every time you reboot, log-on or start a new shell. To avoid this check your distribution's documentation on how to add environment variables to your

[2]See: `http://www.semaine-project.eu/`

shell's configuration files. For the bash shell usually a file called `.profile` or `.bashrc` exists in your home directory to which you can add the two export commands listed above. You can also have a look at the script `buildWithPortAudio.sh`, which creates a wrapper shell script for SMILExtract.

2.2.3 Compiling on Linux/Mac with PortAudio

Important: The following is the manual build instructions, which were relevant for version 1.0, but have been replaced by the build scripts (buildStandalone and builWithPortAudio) in version 2.0. Thus, you should prefer the scripts, or read the scripts and modify them, if you need to customize the build process. The following text is only included as a reference and for historical reasons. Refer to the quick build instruction in Section 2.2.1 for using the build scripts.

To compile openSMILE with PortAudio support, the easiest way is to install the latest version of PortAudio via your distribution's package manager (be sure to install a development package, which includes development header files). You can then run the same steps as in section 2.2.2, the configure script should automatically detect your installation of PortAudio.

If you cannot install packages on your system or do not have access to a PortAudio package, or the portaudio version installed on your system does not work with openSMILE, unpack the file `thirdparty/portaudio.tgz` in the `thirdparty` directory (`thirdparty/portaudio`). Then read the PortAudio compilation instructions and compile and install PortAudio according to these instructions. You can the continue with the steps listed in section 2.2.2. If you have installed PortAudio to a non-standard location (by passing the `--prefix` option to PortAudio's `configure`), you have to pass the path to your PortAudio installation to openSMILE's configure script:

```
./configure --with-portaudio=/path/to/your/portaudio
```

After successfully configuring with PortAudio support, type `make -j4; make; make install`, as described in the previous section.

2.2.4 Compiling on Linux with openCV and portaudio support.

This section briefly describes how to install OpenCV and compile openSMILE with openCV-based video feature support.

Installing OpenCV

You need OpenCV version 2.2 or higher as prerequisite (besides openSMILE version 2.0 or above). If you are using Ubuntu 12.04 or higher, you are lucky since Ubuntu provides OpenCV 2.2 or higher through the standard repositories. To install, just execute

```
sudo apt-get install libopencv*
```

in a shell. This will also take care of the dependencies. The installation path of OpenCV (PATH_TO_OPENCV) in this case will be /usr/.

If you are however using a different distribution or any older Ubuntu version, you might have to compile OpenCV yourself. Detailed instructions on this topic can be found here:

`http://opencv.willowgarage.com/wiki/InstallGuide`

Don't forget to execute `sudo make install` at the end of the installation to install OpenCV to the predefined path. You will need this path (PATH_TO_OPENCV) later in the build process. If you did not specify an alternate installation path, it will most likely be /usr/local/. After the installation you might need to update your library paths in /etc/ld.so.conf and add the line /usr/local/lib, if it is not already there.

Compiling openSMILE on Linux with openCV video support

After you have successfully installed OpenCV, openSMILE can be compiled with support for video input through OpenCV. You will use the standard openSMILE unix build scripts buildStandalone.sh and buildWithPortaudio.sh for this purpose, depending on whether you want to build the standalone version with openCV support or if you also need PortAudio support. Please build your version without openCV first, as described in section 2.2.1. If this succeeds, you can re-run the build script and append the `-o` option to specify the path to your openCV installation.

After the build process is complete, you can check with "./SMILExtract -L", whether cOpenCVSource appears in the component list. In case it does not appear, try to rebuild from one more time by running the build script with the openCV option, before asking for help for sending a bug report to the authors.

If you get an error message that some of the libopencv*.so libraries are not found when you run SMILExtract, type this command in the shell before you run SMILExtract:

```
export LD_LIBRARY_PATH="/usr/local/lib"
```

2.2.5 Compiling on Windows

For compiling openSMILE on Microsoft Windows (Vista, and Windows 7 are supported) there are two ways:

- Using Mingw32 and MSYS or Cygwin (deprecated, not supported officially)
- Using Visual Studio 2010 or above (preferred)

The preferred way (and the only officially supported way) is to compile with Visual Studio 2010. If you want to use Mingw32, please refer to `http://www.mingw.org/wiki/msys` for how to correctly set up your Mingw32 and MSYS system with all necessary development tools (autoconf, automake, libtool, and m4 as included in the MSYS DTK). You should then be able to loosly follow the Unix installation instructions in sections 2.2.2 and 2.2.3.

To compile with Microsoft Visual Studio a single solution file is provided in `ide/vs10/openSmile.sln`. You can select several configurations from this solution which represent the various combinations of the standalone version (simply Release and Debug configurations) and support of OpenCV and PortAudio (named accordingly). The steps to build are in detail:

1. Create a folder `opensmile` somewhere on your system.
2. Copy the openSMILE release package to that folder, and unpack it there.
3. This should create a folder `opensmile-2.x` within your `opensmile` folder.
4. Create a folder called `msvcbuild` in your `opensmile` folder.
5. Open opensmile/opensmile-2.x/ide/vs10/openSmile.sln with Visual Studio.
6. Select the "Release" configuration.
7. Choose "Build Solution" from the build menu.
8. If you get errors about missing libraries, repeat the build process a couple of times (see below for explanation).

9. The openSMILE binaries have now been created in the `msvcbuild` folder. The main binary is SMILExtract_Release.exe. It depends on openSmileLib_Release.dll. So in case you copy the two files to a different location, you must copy both.

Due to some issues with Visual Studio not correctly recognizing the project build order in some configurations, the build process might fail if you just select the option "Build solution!". To solve this issue, you have to build the projects manually. First build openSmileLib*, then openSmileLib, then SMILExtract, or try to build the entire solution multiple times until the number of error messages has converged (build failed, try building projects manually) or has reached zero (build succeeded!).

After successfully building the solution you should have an `openSmileLib*.dll` and a `SMILExtract*.exe` in the msvcbuild directory in t top-level directory of unzipped source package (NOT in the ide/vs10/Release* or Debug* folder!). The * referrs to a prefix that is appended depening on the configuration (PortAudio, OpenCV). After building you can now copy the openSmile dll (also the portaudio dll) and the executable to a directory in your path, e.g. `C:\Windows\system32`, or put everything in a single arbitrary directory and run the executable from this directory.

2.2.6 Compiling on Windows with PortAudio

A PortAudio snapshot known to work with openSMILE is provided in the `thirdparty` subdirectory. Alternatively you can download the latest PortAudio SVN snapshot from `http://www.portaudio.com/`. It is a good idea (however not actually necessary) to read the PortAudio compilation instructions for windows before compiling openSMILE .

Now, unpack the Windows PortAudio source tree to the `thirdparty` subdirectory of the openSMILE distribution (top-level in unpacked zip file), which should create a directory called `portaudio` there. If you don't unpack PortAudio to this location, then you need to modify the Visual Studio project files mentioned in the next paragraph and adjust the Include and Linker paths for PortAudio. By default PortAudio will be built supporting all possible media APIs on a Windows system. However, in most cases only the default Windows Media Extensions (WME) are available and absolutely sufficient. Thus, we provide modified build files for PortAudio in the directory `ide/vs10`. To use them (after unpacking PortAudio to the `thirdparty/portaudio` subdirectory), copy the following files from `ide/vs10` to `thirdparty/portaudio/build/msvc`:

`portaudio.vcxproj`, and `portaudio.def`.

The modified build files basically disable the DirectX, ASIO, and wasapi APIs. They add `PA_NO_DS` and `PA_NO_ASIO` to the preprocessor defines (C/C++ settings tab, preprocessor) and disable all the .cpp files in the related hostapi project folders. Moreover, the output path is adjusted to the msvcbuild directory in the top-level directory and the filename of the output dll is set to `portaudio_x86.dll`.

Now, to compile openSMILE with PortAudio support, select the ReleasePortAudio configuration from the solution `ide/vs10/openSmile.sln` and build it. For detailed steps, please follow the instructions for compiling without PortAudio first (previous section).

Due to some issues with Visual Studio not correctly recognizing the project build order in some configurations, the build process might fail if you just select the option "Build solution!". To solve this issue, you have to build the projects manually. First build portaudio, then openSmileLib*, then openSmileLib, then SMILExtract, or try to build the entire solution multiple times until the number of error messages has converged (build failed, try building projects manually) or has reached zero (build succeeded!).

Please note: the PortAudio versions of the openSMILE Visual Studio projects assume that the dll is called `portaudio_x86.dll` and the import library `portaudio_x86.lib` and both are found in the msvcbuild directory in the top-level. This name, however, might be different, depending on your architecture. Thus, you should check this and change the name of the import library in the Linker advanced settings tab.

2.2.7 Compiling on Windows with openCV support.

You first need to download a Windows binary release of openCV version 2.4.5 from
`http://opencv.org/downloads.html`

If you have another version (¿2.2), you will need to modify the names of the .lib files in the solution configuration manually (in opencv.props in ide/vs10). The following instructions assume you have version 2.4.5.

Unpack the openCV distribution to a directory you choose by running the self-extracting .exe. From that directory copy `build/<yourplatform>/vc10/lib` and `build/include/` to `thirdparty/compiled/include` and `thirdparty/compiled/lib`. Make sure to replace `<yourplatform>` by the correct platform (x64 or x86) to match the platform you are building openSMILE for (drop down menu in Visual Studio next to the solution configuration). Please note that currently in the release candidates the build only works for the Win32 (x86) platform. The proper configuration for the x64 platform will follow shortly. If you need to have an x64 version, you will have to copy the configuration from the Win32 platform.

Then build the `ReleaseOpenCV` configuration. If you also want PortAudio support, pick the according configuration (`ReleasePortaudioOpenCV`) and additionally follow the instructions mentioned in section 2.2.6. Due to some issues with Visual Studio not correctly recognizing the project build order in some configurations, the build process might fail if you just select the option "Build solution!". To solve this issue, you have to build the projects manually. First build portaudio (if building with portaudio support), then openSmileLib*, then openSmileLib, then SMILExtract, or try to build the entire solution multiple times until the number of error messages has converged (build failed, try building projects manually) or has reached zero (build succeeded!).

The built executable will be located in `msvcbuild/`. You will find `openSmileLib_<configuration>.dll` and `SMILExtract_<configuration>.exe` there, which you will both need. The .dll should be in the same path as the .exe to run, or copied to a system path such as Windows/system32. You will also need to copy the opencv dlls to a system path or to the same path that contains the SMILExtract binary. The opencv executables are contained in the unpacked openCV distribution from which you copied the lib files. Look in the folder `bin/` which is on the same level as the `lib/` folder.

2.2.8 Compiling for Android and creating the example Android app

This chapter explains how to compile openSMILE for Android. Additionally, a description is given how to integrate the openSMILE Android binary into an Android app to display openSMILE audio analysis results in real-time.

Download and extract the Android NDK Release 10e from the following links:

- OS X:
  `http://dl.google.com/android/ndk/android-ndk-r10e-darwin-x86_64.bin`
- Linux 32 bit:
  `http://dl.google.com/android/ndk/android-ndk-r10e-linux-x86.bin`

- Linux 64 bit:
  `http://dl.google.com/android/ndk/android-ndk-r10e-linux-x86_64.bin`
- Windows 64 bit:
  `http://dl.google.com/android/ndk/android-ndk-r10e-windows-x86_64.exe`
- Windows 32 bit:
  `http://dl.google.com/android/ndk/android-ndk-r10e-windows-x86.exe`

Download and install a recent version of Android Studio from http://developer.android.com/studio.

Go to the root directory of the extracted openSMILE archive. Open the script `buildAndroid.sh` in a text editor. Change the path in the line starting with `export NDK` to the actual path of the NDK you extracted beforehand OR create a symbolic link at /android/ndk-r10e to the root of your Android NDK package (in this case the "realpath" command is required to be installed on your system or supported by your shell (bash, for example)).

Then, run `buildAndroid.sh`. If successful, from the extracted openSMILE archive, copy the directory `openSMILE/progsrc/android-template` to some other location on your computer, or import this folder into android studio directly.

To build the example Android App, follow these steps:

1. Edit app/src/main/jni/sync-build-libopensmile.sh and Android.mk and adapt OPENSMILE_DIR and OPENSMILE_ROOT, respectively, to point to your openSMILE root directory.
2. If you have not yet run buildAndroid.sh in the opensmile trunk, do so now to fully build the android libs
3. To sync the libs with the app project, do:
   (a) cd app/src/main/jni
   (b) sh sync-build-libopensmile.sh
4. Open project in android studio
5. Change sdk.dir and ndk.dir in GradleScripts' local.properties to your SDK and NDK locations.
6. Edit the `OPENSMILE_ROOT` path in app/src/main/jni/Android.mk
7. Edit the ndk-build command (and path) in app/build.gradle or better: add the NDK directory to your system pathm and then use 'ndk-build' as command
8. At this you point you should be able to run the app on your Android smartphone.

If you want to modify your app to make it fit your needs, please consider the following relevant points therefore:

- Copy openSMILE configs and models to app/src/assets. Load these by specifying them in app/src/plugins/Config.scala.
- For a new GUI application, create a new SmilePlugin within app/src/scala/.../plugins. See the existing example and config code and respective documentation there.
- If you want to do something different than just showing some live openSMILE output, take a look at the following classes and respective methods:

1. src/scala/.../MainActivity.scala → SmileThread & onSmileMessageReceived()
2. src/java/.../SmileJNI.java → interface Listener & registerListener()

2.3 Extracting your first features

Now, that you have either successfully downloaded and installed the binary version of openSMILE or have compiled the source code yourself, you are ready to test the program and extract your first features. To check if you can run SMILExtract, type:

```
SMILExtract -h
```

If you see the usage information and version number of openSMILE , then everything is set up correctly. You will see some lines starting with `(MSG)` at the end of the output, which you can safely ignore. To check if your SMILExtract binary supports live audio recording and playback, type:

```
SMILExtract -H cPortaudio
```

If you see various configuration option of the cPortaudio components, then your binary supports live audio I/O. If you see only three lines with messages, then you do not have live audio support. To check if your SMILExtract binary supports video features via OpenCV, type:

```
SMILExtract -H cOpenCV
```

If you see various configuration option of the cPortaudio components, then your binary supports live audio I/O.

Please note: You may have to prefix a "./" on Unix like systems, if SMILExtract is not in your path but in the current directory instead.

Now we will start using SMILExtract to extract very simple audio features from a wave file. You can use your own wave files if you like, or use the files provided in the `wav-samples` directory.

For a quick start, we will use an example configuration file provided with the openSMILE distribution. Type the following command in the top-level directory of the openSMILE package (if you start openSMILE in a different directory you must adjust the paths to the config file and the wave file):

```
SMILExtract -C config/demo/demo1_energy.conf -I wav_samples/speech01.wav -
    O speech01.energy.csv
```

If you get only `(MSG)` and `(WARN)` type messages, and you see `Processing finished!` in the last output line, then openSMILE ran successfully. If something fails, you will get an `(ERROR)` message.

> *Note for windows users:* Due to faulty exception handling, if an exception indicating an error is thrown in the DLL and caught in the main executable, Windows will display a program crash dialogue. In most cases openSMILE will have displayed the error message beforehand, so can just close the dialogue. In some cases however, Windows kills the program before it can display the error message. If this is the case, please use Linux, or contact the authors and provide some details on your problem.

Now, if openSMILE ran successfully, open the file `speech01.energy.csv` in a text editor to see the result. You can also plot the result graphically using gnuplot. This is discussed in section 2.8.

Next, we will generate the configuration file from the above simple example ourselves, to learn how openSMILE configuration files are written. openSMILE can generate configuration file templates for simple scenarios. We will use this function to generate our first configuration file, which will be capable of reading a wave file, compute frame energy, and saving the output to a CSV file. First, create a directory `myconfig` which will hold your configuration files. Now type the following (without newlines) to generate the first configuration file:

```
SMILExtract -cfgFileTemplate -configDflt cWaveSource,cFramer,cEnergy,
    cCsvSink -l 1 2> myconfig/demo1.conf
```

The `-cfgFileTemplate` option instructs openSMILE to generate a configuration file template, while the `-configDflt` option is used to specify a comma separated list of components which shall be part of the generated configuration. The `-l 1` option sets the log-level to one, to suppress any messages, which should not be in the configuration file (you will still get ERROR messages on log-level one, e.g. messages informing you that components you have specified do not exist, etc.). The template text is printed to standard error, thus we use `2>` to dump it to the file `myconfig/demo1.conf`. If you want to add comments describing the individual option lines in the generated configuration file, add the option `-cfgFileDescriptions` to the above command-line.

The newly generated file consists of two logical parts. The first part looks like this (please note, that comments in the examples are started by ; or // and may only start at the beginning of a line):

```
 ;= component manager configuration (= list of enabled components!) =

[componentInstances:cComponentManager]
 // this line configures the default data memory:
instance[dataMemory].type = cDataMemory
instance[waveSource].type = cWaveSource
instance[framer].type = cFramer
instance[energy].type = cEnergy
instance[csvSink].type = cCsvSink
// Here you can control the amount of detail displayed for the
// data memory level configuration. 0 is no information at all,
// 5 is maximum detail.
printLevelStats = 1
 // You can set the number of parallel threads (experimental):
nThreads = 1
```

It contains the configuration of the component manager, which determines what components are instantiated when you call SMILExtract. There always has to be one cDataMemory component, followed by other components. The name given in `[]` specifies the name of the component instance, which must be unique within one configuration.

The next part contains the component configuration sections, where each begins with a section header:

```
[waveSource:cWaveSource]
...
[framer:cFramer]
...
[energy:cEnergy]
...
```

```
[csvSink:cCsvSink]
...
```

The section header follows this format: `[instanceName:componentType]`. The template component configuration sections are generated with all available values set to their default values. This functionality currently is still experimental, because some values might override other values, or have a different meaning if explicitly specified. Thus, you should carefully check all the available options, and list only those in the configuration file which you require. Even if in some cases you might use the default values (such as the number of spectral bands, etc.) it is considered good practice to include these in the configuration file. This will ensure compatibility with future versions, in case the defaults - for whatever reason - might change. Moreover, it will increase the readability of your configuration files because all parameters can be viewed in one place without looking up the defaults in this manual.

Next, you have to configure the component connections. This can be done by assigning so called data memory "levels" to the dataReader and dataWriter components which are always contained in each source, sink, or processing component by modifying the `reader.dmLevel` and `writer.dmLevel` lines. You can choose arbitrary names for the writer levels here, since the dataWriters register and create the level you specify as `writer.dmLevel` in the data memory. You then connect the components by assigning the desired read level to `reader.dmLevel`. Thereby the following rules apply: for one level only **one** writer may exist, i.e. only one component can write to a level; however, there is no limit to the number of components that read from a level, and one component can read from more than one level if you specify multiple level names separated by a `;`, such as `reader.dmLevel = energy;loudness` to read data from the levels `energy` and `loudness`. Data is thereby concatenated column wise.

For our example configuration we want the cFramer component to read from the input PCM stream, which is provided by the cWaveSource component, create frames of 25 ms length every 10 ms and write these frames to a new level we call "energy"), thus we change the following lines:

```
[waveSource:cWaveSource]
writer.dmLevel = <<XXXX>>
```

to

```
[waveSource:cWaveSource]
writer.dmLevel = wave
```

and the framer section

```
[framer:cFramer]
reader.dmLevel = <<XXXX>>
writer.dmLevel = <<XXXX>>
...
```

to (note, that we removed a few superfluous frameSize* options and changed frameStep to 0.010):

```
[framer:cFramer]
reader.dmLevel = wave
writer.dmLevel = waveframes
copyInputName = 1
frameMode = fixed
frameSize = 0.025000
frameStep = 0.010000
frameCenterSpecial = left
```

```
noPostEOIprocessing = 1
```

Next, the cEnergy component shall read the audio frames and compute the signal log energy, and the cCsvSink shall write them to a CSV format file. Thus, we change the corresponding lines to:

```
[energy:cEnergy]
reader.dmLevel = waveframes
writer.dmLevel = energy
...
rms = 0
log = 1
...
[csvSink:cCsvSink]
reader.dmLevel = energy
filename = myenergy.csv
...
```

We are now ready to run SMILExtract with our own configuration file:

```
SMILExtract -C myconfig/demo1.conf
```

This will open the file "input.wav" in the current directory (be sure to copy a suitable wave file and rename it to "input.wav"), do the feature extraction, and save the result to "myenergy.csv". The result should be the same as with the example configuration file.

If you want to be able to pass the input file name and the output file name on the SMILExtract command-line, you have to add a command to the configuration file to define a custom command-line option. To do this, change the filename lines of the wave source and the csv sink to:

```
[waveSource:cWaveSource]
...
filename = \cm[inputfile(I):file name of the input wave file]
...
[csvSink:cCsvSink]
...
filename = \cm[outputfile(O):file name of the output CSV file]
...
```

You can now run:

```
SMILExtract -C myconfig/demo1.conf -I wav\_samples/speech01.wav -O
   speech01.energy.csv
```

This concludes the introductory section. We hope that you now understand the basics of how to use and configure openSMILE , and are ready to take a look at the more complex examples, which are explained in section 2.5. Among these are also standard baseline feature sets of international research competitions. The section also explains several commandline options that the standard feature set configuration files all provide and that can be used to influence parameters of the data input and output.

To explore the full potential of openSMILE configuration files, please read section 4.2, which provides description of the format, and section 4.3, which describes the function and configuration options of all components in detail. If you are interested what is going on inside openSMILE,

which components exist besides those which are instantiable and connectable via the configuration files, and to learn more about the terminology used, then you should read section 2.4 which describes the program architecture in detail.

2.4 What is going on inside of openSMILE

The SMILExtract binary is the main application which can run all configuration files. If you take a look at the source code of it (which is found in `SMILExtract.cpp`), you will see that it is fairly short. It uses the classes from the openSMILE API to create the components and run the configurations. These API functions can be used in custom applications, such as GUI front-ends etc. Therefore, they will be described in more detail in the developer's documentation in section 5. However, to obtain a general understanding what components make openSMILE run, how they interact, and in what phases the program execution is split, a brief overview is given in this section.
openSMILE's application flow can be split into three general phases:

Pre-config phase Command-line options are read and the configuration file is parsed. Also, usage information is displayed, if requested, and a list of built-in components is generated.

Configuration phase The component manager is created and instantiates all components listed in its `instances` configuration array. The configuration process is then split into 3 phases, where components first register with the component manager and the data memory, then perform the main configuration steps such as opening of input/output files, allocation of memory, etc., and finally finalise their configuration (e.g. set the names and dimensions of their output fields, etc.). Each of the 3 phases is passed through several times, since some components may depend on other components having finished their configuration (e.g. components that read the output from another component and need to know the dimensionality of the output and the names of the fields in the output). Errors, due to mis-configurations, bogus input values, or inaccessible files, are likely to happen during this phase.

Execution phase When all components have been initialised successfully, the component manager starts the main execution loop (also referred to as tick-loop). Every component has a tick() method, which implements the main incremental processing functionality and reports on the status of the processing via its return value.

In one iteration of the execution loop, the component manager calls all tick() functions in series (*Note:* the behaviour is different, when components are run in multiple threads). The loop is continued as long as at least one component's tick() method returns a non-zero value (which indicates that data was processed by this component).

If all components indicate that they did not process data, it can be safely assumed that no more data will arrive and the end of the input has been reached (this may be slightly different for on-line settings, however, it is up to the source components to return a positive return value or pause the execution loop, while they are waiting for data).

When the end of the input is reached, the component manager signals the end-of-input condition to the components by running one final iteration of the execution loop. After that the execution loop will be ran a new, until all components report a failure status. This second phase is referred to end-of-input processing. It is mainly used for off-line processing, e.g. to compute features from the last (but incomplete) frames, to mean normalise a complete sequence, or to compute functionals from a complete sequence.

openSMILE contains three classes which cannot be instantiated from the configuration files. These are the commandline parser (cCommandlineParser), the configuration manager (cConfigManager), and the component manager (cComponentManager). We will now briefly describe the role of each of these in a short paragraph. The order of the paragraph corresponds to the order the classes are created during execution of the SMILExtract program.

The commandline parser This class parses the command-line and provides options in an easily accessible format to the calling application. Simple command-line syntax checks are also performed. After the configuration manager has been initialised and the configuration has been parsed, the command-line is parsed a second time, to also get the user-defined command-line options set in the current configuration file.

The configuration manager The configuration manager loads the configuration file, which was specified on the SMILExtract command-line. Thereby, configuration sections are split and then parsed individually. The configuration sections are stored in an abstract representation as ConfigInstance classes (the structure of these classes is described by a ConfigType class). Thus, it is easy to add additional parsers for formats other than the currently implemented ini-style format.

The component manager The component manager is responsible of instantiating, configuring, and executing the components. The details have already been described in the above section on openSMILE's application flow. Moreover, the component manger is responsible of enumerating and registering components in plugins. Therefore, a directory called `plugins` is scanned for binary plugins. The plugins found are registered, an become useable exactly in the same way as built-in components. A single plugin binary thereby can contain multiple openSMILE components.

The components instantiated by the component manager are all descendants of the cSmileComponent class. They have two basic means of standardised communication: a) directly and asynchronously, via smile messages, and b) indirectly and synchronously via the data memory.

Method a) is used to send out-of-line data, such as events and configuration changes directly from one smile component to another. Classifier components, for example, send a 'classificationResult' message, which can be caught by other components (esp. custom plug-ins), to change their behaviour or send the message to external sources.

Method b) is the standard method for handling of data in openSMILE. The basic principle is that of a data source producing a frame of data and writing it to the data memory. A data processor reads this frame, applies some fancy algorithm to it, and writes a modified output frame back to a different location in the data memory. This step can be repeated for multiple data processors. Finally, a data sink reads the frame and passes it to an external source or interprets (classifies) it in some way. The advantage of passing data indirectly is that multiple components can read the same data, and data from past frames can stored efficiently in a central location for later use.

2.4.1 Incremental processing

The data-flow in openSMILE is handled by the cDataMemory component. This component manages multiple data memory 'levels' internally. These levels are independent data storage locations, which can be written to by exactly one component and read by an arbitrary number of components. From the outside (the component side) the levels appear to be a $Nx\infty$ matrix,

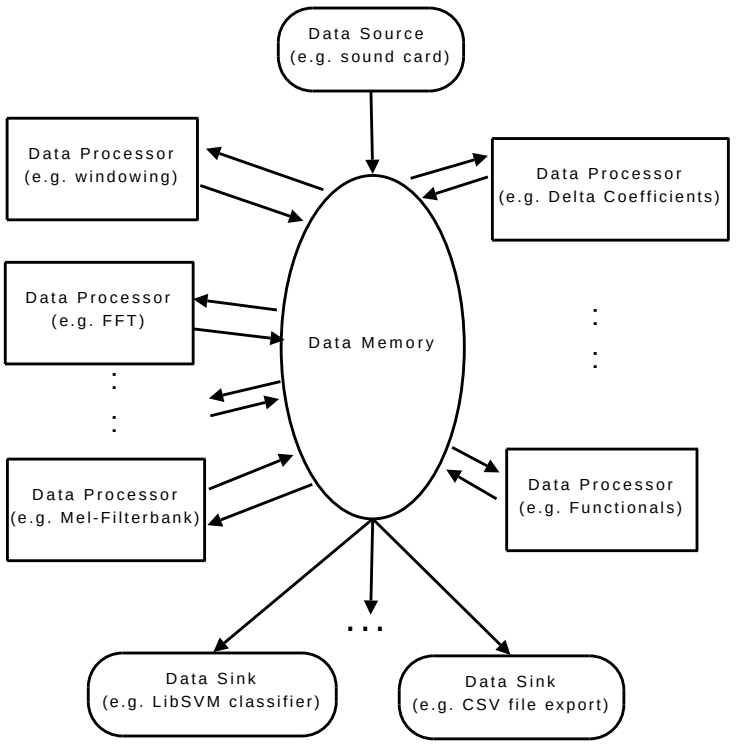

Figure 2.1: Overview on openSMILE's component types and openSMILE's basic architecture.

with N rows, whereby N is the frame size. Components can read / write frames (=columns) at / to any location in this virtual matrix. If this matrix is internally represented by a ring-buffer, a write operation only succeeds if there are empty frames in the buffer (frames that have not been written to, or frames that have been read by all components reading from the level), and a read operation only succeeds if the referred frame index lies no more than the ring buffer size in the past. The matrices can also be internally represented by a non-ring buffer of fixed size (nT=size, growDyn=0, isRb=0), or variable size (nT=initial size, growDyn=1, isRb=0). In the case of the variable size a write will always succeed, except when there is no memory left; for a fixed frame size a write will succeed until the buffer is full, after that the write will always fail. For fixed buffers, reads from 0 to the current write position will succeed.

Figure 2.1 shows the overall data-flow architecture of openSMILE, where the data memory is the central link between all dataSource, dataProcessor, and dataSink components.

The ring-buffer based incremental processing is illustrated in figure 2.2. Three levels are present in this setup: wave, frames, and pitch. A cWaveSource component writes samples to the 'wave' level. The write positions in the levels are indicated by a red arrow. A cFramer produces frames of size 3 from the wave samples (non-overlapping), and writes these frames to the 'frames' level. A cPitch (a component with this name does not exist, it has been chosen here only for illustration purposes) component extracts pitch features from the frames and writes them to the 'pitch' level. In figure 2.2 (right) the buffers have been filled, and the write pointers have been warped. Data that lies more than 'buffersize' frames in the past has been overwritten.

Figure 2.3 shows the incremental processing of higher order features. Functionals (max and min) over two frames (overlapping) of the pitch features are extracted and saved to the level 'func'.

The size of the buffers must be set correctly to ensure smooth processing for all blocksizes. A 'blocksize' thereby is the size of the block a reader or writer reads/writes from/to the dataMemory at once. In the above example the read blocksize of the functionals component would be 2

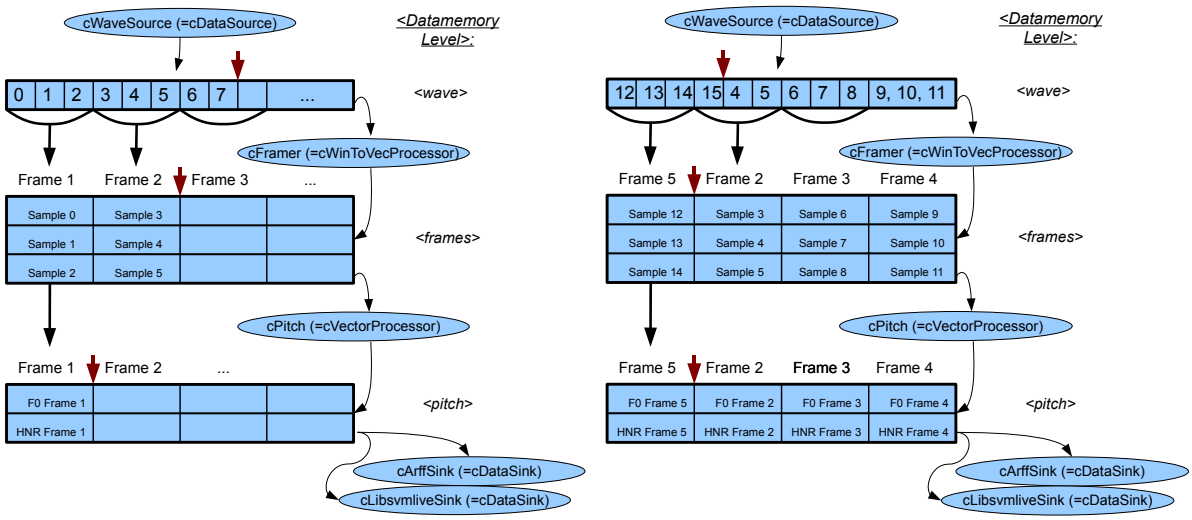

Figure 2.2: Incremental processing with ring-buffers. Partially filled buffers (left) and filled buffers with warped read/write pointers (right).

because it reads two pitch frames at once. The input level buffer of 'pitch' must be at least 2 frames long, otherwise the functionals component will never be able to read a complete window from this level.

openSMILE handles automatic adjustment of the buffersizes. Therefore, readers and writers must register with the data memory during the configuration phase and publish their read and write blocksizes. The minimal buffersize is computed based on these values. If the buffersize of a level is set smaller than the minimal size, the size will be increased to the minimum possible size. If the specified size (via configuration options) is larger than the minimal size, the larger size will be used. *Note:* this automatic buffersize setting only applies to ring-buffers. If you use non-ring buffers, or if you want to process the full input (e.g. for functionals of the complete input, or mean normalisation) it is always recommended to configure a dynamically growing non-ring buffer level (see the cDataWriter configuration for details).

2.4.2 Smile messages

This section has yet to be written. In the meantime, please refer to the file `doc/developer/messages.txt` for a minimal documentation of currently available smile messages. See also the `smileComponent.hpp` source file, which contains the structural definitions of smile messages.

2.4.3 openSMILE terminology

In the context of the openSMILE data memory various terms are used which require clarification and a precise definition, such as 'field', 'element', 'frame', and 'window'.

You have learnt about the internal structure of the dataMemory in section 2.4.1. Thereby a level in the data memory represents a unit which contains numeric data, frame meta data, and temporal meta data. Temporal meta data is present on the one hand for each frame, thereby describing frame timestamps and custom per frame meta information, and on the other hand globally, describing the global frame period and timing mode of the level.

If we view the numeric contents of the data memory level as a 2D <nFields x nTimestemps> matrix, 'frames' correspond to the columns of this matrix, and 'windows' or 'contours' correspond the rows of this matrix. The frames are also referred to as (column-)'vectors' in some places.

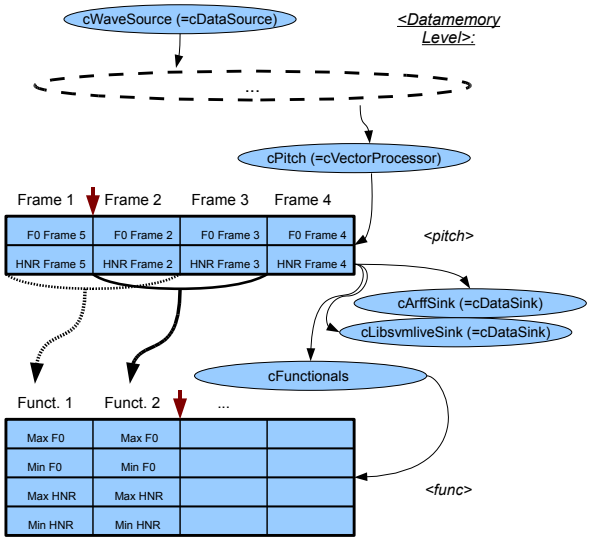

Figure 2.3: Incremental computation of high-level features such as statistical functionals.

(*Note:* when exporting data to files, the data – viewed as matrix – is transposed, i.e. for text-based files (CSV, ARFF), the rows of the file correspond to the frames.) The term 'elements' – as used in openSMILE – refers to the actual elements of the frames/vectors. The term 'field' refers to a group of elements that belongs together logically and where all elements have the same name. This principle shall be illustrated by an example: A feature frame containing the features 'energy', 'F0', and MFCC 1-6, will have $1 + 1 + 6 = 8$ elements, but only 3 fields: the field 'energy' with a single element, the field 'F0' with a single element, and the (array-) field 'MFCC' with 6 elements (called 'MFCC[0]' – 'MFCC[1]').

2.5 Default feature sets

For common tasks from the Music Information Retrieval and Speech Processing fields we provide some example configuration files in the `config/` directory for the following frequently used feature sets. These also contain the baseline acoustic feature sets of the 2009–2013 INTERSPEECH challenges on affect and paralinguistics:

- Chroma features for key and chord recognition
- MFCC for speech recognition
- PLP for speech recognition
- Prosody (Pitch and loudness)
- The INTERSPEECH 2009 Emotion Challenge feature set
- The INTERSPEECH 2010 Paralinguistic Challenge feature set
- The INTERSPEECH 2011 Speaker State Challenge feature set
- The INTERSPEECH 2012 Speaker Trait Challenge feature set
- The INTERSPEECH 2013 ComParE feature set
- The MediaEval 2012 TUM feature set for violent scenes detection.

- Three reference sets of features for emotion recognition (older sets, obsoleted by the new INTERSPEECH challenge sets)
- Audio-visual features based on INTERSPEECH 2010 Paralinguistic Challenge audio features.

These configuration files can be used as they are, or as a basis for your own feature files.

> *Note:* If you publish results with features extracted by openSMILE, we would appreciate it if you share your configuration files with the research community, by uploading them to your personal web-pages and providing the URL in the paper, for example. Please also indicate which official version of openSMILE you have used to allow others to reproduce your results.

2.5.1 Common options for all standard configuration files

Since version 2.2 most standard feature extraction configuration files include a standard interface for specifying commandline options for audio input and for feature output in various formats (WEKA Arff, HTK binary, CSV text). Common configuration file includes are used for this purpose, which can be found in the folder `config/shared`.

The options are defined in the configuration files themselves, see Section 4.2 for details on this mechanism and the syntax. In order to view the available options for a configuration file, use the following command:

```
SMILExtract -C config/putconfigfilenamehere.conf -ccmdHelp
```

The following options are available for audio input for all standard configuration files:

```
-inputfile, -I <filename>     Path and name of input audio file.

-start         <t in seconds> Where to start analysis, relative
                                to the beginning of the file (0).

-end           <t in seconds> Where to end analysis, relative
                                to beginning of file (0).
                                Default (-1) is end of file.
```

These options are defined in `config/shared/standard_wave_input.conf.inc`.

The following options are available for controlling the buffer and segmentation behaviour:

```
-frameModeFunctionalsConf <file> Include, which configures the frame
                                 mode setting for all functionals
                                 components.
  Default: shared/FrameModeFunctionals.conf.inc

-bufferModeRbConf shared/BufferModeRb.conf.inc
-bufferModeRbLagConf shared/BufferModeRbLag.conf.inc
-bufferModeConf shared/BufferMode.conf.inc
```

`frameModeFunctionalsConf` is the most important option. It controls the time units on which the functionals components operate. The following examples (contents of the included file) illustrate the four most common use-cases.

A. summary over complete input:

```
frameMode = full
frameSize = 0
frameStep = 0
frameCenterSpecial = left
```

B. Multiple summaries over fixed size (sliding) windows (5 seconds long, shifted forward at intervals of 2 seconds):

```
frameMode = fixed
frameSize = 5
frameStep = 2
frameCenterSpecial = left
```

C. Summaries over a given list of segments (4.2 seconds to 5.6 seconds, 7.0 to 9.0 seconds, 10 seconds to end of file):

```
frameMode = list
frameList = 4.2s-5.6s,7.0s-9s,10s-E
frameCenterSpecial = left
```

D. Summaries over variable segments, which are detected on-the-fly by a cTurnDetector component, and received via smile messages:

```
frameMode = var
frameCenterSpecial = left
```

A cTurnDetector component must be present in the configuration with the messageRecp option pointing to all functionals components in the configuration. See the online help of cTurnDetector for details, or `config/emobase_live4.conf` for a simple example.

The `bufferMode` configuration files set the size of the dataMemory level output buffers for general components which process frame-by-frame (`bufferModeRbConf`), for components which operate with lagged data (e.g., F0 after Viterbi smoothing) (`bufferModeRbLagConf`), and for components which produce output that is to be summarised over time by other components (e.g., functionals, or classifiers that use context) (`bufferModeConf`). The buffer size configured by the latter must match the frameMode setting in frameModeFunctionalsConf, i.e., the buffer specified in bufferModeConf must be at least the size of the requested unit of segmentation (frameMode). In case the complete input is to be summarised (frameMode = full), the output buffer must be configured to grow dynamically (growDyn=1) and not act as as ring/cyclical buffer (isRb=0), e.g.:

```
writer.levelconf.growDyn = 1
writer.levelconf.isRb = 0
writer.levelconf.nT = 100
```

The value of `nT` is not relevant, it just sets the initial size of the buffer (in number of LLD frames). This configuration is not suitable for live mode, as it will occupy infinite amounts of RAM over time and will make the openSMILE process crash at some point in time.

Thus, for live demos, the buffer size must be constrained and the maximum size of segments to summarise features over must also be constrained. In variable mode (when receiving messages from cTurnDetector), this is achieved by the maximum turn length settings in cTurnDetector. Otherwise, the frameSize setting e.g. should be less then the buffer size (nT). An example for a ring-buffer configuration for live mode would be:

```
writer.levelconf.growDyn = 0
writer.levelconf.isRb = 1
```

```
writer.levelconf.nT = 1000
```

This corresponds to a buffer size of 10 seconds, if the frame rate of LLD features is 10 ms, which is the default in almost all configurations.

The following options are available for controlling the output data formats (for configurations which provide feature summaries via statistical functionals, such as all INTERSPEECH and AVEC challenge sets):

```
============================
-instname       <string>    Usually the input filename, saved in
                              first column in CSV and ARFF output.
                              Default is "unknown".

============================
-lldcsvoutput, -D <filename> Enables LLD frame-wise output to CSV.

-appendcsvlld   <0/1>       Set to 1 to append to existing CSV
                              output file. Default is overwrite (0).

-timestampcsvlld <0/1>      Set to 0 to disable timestamp output
                              to CSV in second column. Default is 1.

-headercsvlld   <0/1>       Set to 0 to disable header output (1st
                              line) to CSV. Default is 1 (enabled)

============================
-lldhtkoutput   <filename>  Enables LLD frame-wise output to
                              HTK format.

============================
-lldarffoutput, -D <filename> Enables LLD frame-wise output to ARFF.

-appendarfflld   <0/1>      Set to 1 to append to existing ARFF
                              output file. Default is overwrite (0).

-timestamparfflld <0/1>     Set to 0 to disable timestamp output
                              to ARFF in second column. Default 1.

-lldarfftargetsfile <file>  Specify the configuration include, that
                              defines the target fields (classes)
                              Default: shared/arff_targets_conf.inc

============================
-output, -O     <filename>  The default output option. To ARFF
                              file format, for feature summaries.

-appendarff     <0/1>       Set to 0 to not append to existing ARFF
                              output file. Default is append (1).

-timestamparff  <0/1>       Set to 1 to enable timestamp output
```

```
                                to ARFF in second column. Default 0.

-arfftargetsfile <file>     Specify the configuration include, that
                              defines the target fields (classes)
                              Default: shared/arff_targets_conf.inc

==============================
-csvoutput      <filename>  The default output option. To CSV
                              file format, for feature summaries.

-appendcsv      <0/1>       Set to 0 to not append to existing CSV
                              output file. Default is append (1).

-timestampcsv   <0/1>       Set to 0 to disable timestamp output
                              to CSV in second column. Default 1.

-headercsv      <0/1>       Set to 0 to disable header output (1st
                              line) to CSV. Default is 1 (enabled)

==============================
-htkoutput     <filename>   Enables output to HTK format of
                              feature summaries (functionals).
```

These options are defined in `config/shared/standard_data_output.conf.inc`.

For configurations which provide Low-Level-Descriptor (LLD) features only (i.e. which do not summarise features by means of statistical functionals over time), the following output options are available:

```
==============================
-csvoutput      <filename>  The default output option. To CSV
                              file format, for frame-wise LLD.

-appendcsv      <0/1>       Set to 1 to append to existing CSV
                              output file. Default is overwrite (0).

-timestampcsv   <0/1>       Set to 0 to disable timestamp output
                              to CSV in second column. Default 1.

-headercsv      <0/1>       Set to 0 to disable header output (1st
                              line) to CSV. Default is 1 (enabled)

==============================
-output, -O     <filename>  Default output to HTK format of
                              feature summaries (functionals).

==============================
-arffoutput     <filename>  The default output option. To ARFF
                              file format, for frame-wise LLD.

-appendarff     <0/1>       Set to 0 to not append to existing ARFF
```

```
                                output file. Default is append (1).

  -timestamparff  <0/1>       Set to 0 to disable timestamp output
                                to ARFF in second column. Default 1.

  -arfftargetsfile <file>     Specify the configuration include, that
                                defines the target fields (classes)
                                Default: shared/arff_targets_conf.inc
```

These options are defined in `config/shared/standard_data_output_lldonly.conf.inc`.

Note: Since version 2.2 you can specify a '?' instead of a filename. This will disable the corresponding output component, i.e., it will not write an output file. In the standard data output interface all filenames default to '?', except for the standard output options (-O) which default to `output.htk` or `output.arff`.

All configuration files which support the standard data output format can be used in the Windows batch feature extraction GUI (source code in C# for VS10 in `progsrc/openSMILEbatchGUI/`). This tool allows to run openSMILE on several files in a folder automatically. It allows to select audio files and specify the file output type via a graphical interface.

2.5.2 Chroma features

The configuration file `config/chroma_fft.conf` computes musical Chroma features (for 12 semitones) from a short-time FFT spectrogram (window-size 50 ms, rate 10 ms, Gauss-window). The spectrogram is scaled to a semi-tone frequency axis scaling using triangular filters. To use this configuration, type:

```
SMILExtract -C config/chroma_fft.conf -I input.wav -O chroma.csv
```

The resulting CSV file contains the Chroma features as ascii float values separated by ';', one frame per line. This configuration uses the 'cTonespec' component to compute the semitone spectrum. We also provide a configuration using the experimental 'cTonefilt' as a replacement for 'cTonespec' in the file `config/chroma_filt.conf`.

We also provide an example configuration for computing a single vector which contains the mean value of the Chroma features computed over the complete input sequence. Such a vector can be used for recognising the musical key of a song. The configuration is provided in `config/chroma_fft.sum.conf`. It uses the 'cFunctionals' component to compute the mean values of the Chroma contours. Use it with the following command-line:

```
SMILExtract -C config/chroma_fft.sum.conf -I input.wav -O chroma.csv
```

`chroma.csv` will contain a single line with 12 values separated by ';', representing the mean Chroma values.

2.5.3 MFCC features

For extracting MFCC features (HTK compatible) the following four files are provided (they are named after the corresponding HTK parameter kinds they represent):

MFCC12_0_D_A.conf This configuration extracts Mel-frequency Cepstral Coefficients from 25 ms audio frames (sampled at a rate of 10 ms) (Hamming window). It computes 13 MFCC (0-12) from 26 Mel-frequency bands, and applies a cepstral liftering filter with a weight parameter of 22. 13 delta and 13 acceleration coefficients are appended to the MFCC.

MFCC12_E_D_A.conf This configuration is the same as MFCC12_0_D_A.conf, except that the log-energy is appended to the MFCC 1-12 instead of the 0-th MFCC.

MFCC12_0_D_A_Z.conf This configuration is the same as MFCC12_0_D_A.conf, except that the features are mean normalised with respect to the full input sequence (usually a turn or sub-turn segment).

MFCC12_E_D_A_Z.conf This configuration is the same as MFCC12_E_D_A.conf, except that the features are mean normalised with respect to the full input sequence (usually a turn or sub-turn segment).

The frame size is set to 25 ms at a rate of 10 ms. A Hamming function is used to window the frames and a pre-emphasis with $k = 0.97$ is applied. The MFCC 0/1-12 are computed from 26 Mel-bands computed from the FFT power spectrum. The frequency range of the Mel-spectrum is set from 0 to 8 kHz. These configuration files provide the `-I` and `-O` options. The output file format is the HTK parameter file format. For other file formats you must change the 'cHtkSink' component type in the configuration file to the type you want. An example command-line is given here:

```
SMILExtract -C config/MFCC12_E_D_A.conf -I input.wav -O output.mfcc.htk
```

2.5.4 PLP features

For extracting PLP cepstral coefficients (PLP-CC) (HTK compatible) the following four files are provided (they are named after the corresponding HTK parameter kinds they represent):

PLP_0_D_A.conf This configuration extracts Mel-frequency Cepstral Coefficients from 25 ms audio frames (sampled at a rate of 10 ms) (Hamming window). It computes 6 PLP (0-5) from 26 Mel-frequency bands using a predictor order of 5, and applies a cepstral liftering filter with a weight parameter of 22. 6 delta and 6 acceleration coefficients are appended to the PLP-CC.

PLP_E_D_A.conf This configuration is the same as PLP_0_D_A.conf, except that the log-energy is appended to the PLP 1-5 instead of the 0-th PLP.

PLP_0_D_A_Z.conf This configuration is the same as PLP_0_D_A.conf, except that the features are mean normalised with respect to the full input sequence (usually a turn or sub-turn segment).

PLP_E_D_A_Z.conf This configuration is the same as PLP_E_D_A.conf, except that the features are mean normalised with respect to the full input sequence (usually a turn or sub-turn segment).

The frame size is set to 25 ms at a rate of 10 ms. A Hamming function is used to window the frames and a pre-emphasis with $k = 0.97$ is applied. The PLP 0/1-5 are computed from 26 auditory Mel-bands (compression factor 0.33) computed from the FFT power spectrum. The predictor order of the linear predictor is 5. The frequency range of the Mel-spectrum is set from 0 to 8 kHz. These configuration files provide the `-I` and `-O` options. The output file format is the HTK parameter file format. For other file formats you must change the 'cHtkSink' component type in the configuration file to the type you want. An example command-line is given here:

```
SMILExtract -C config/PLP_E_D_A.conf -I input.wav -O output.plp.htk
```

2.5.5 Prosodic features

Example configuration files for extracting prosodic features are provided in the files

`config/prosodyAcf.conf`, and `config/prosodyShs.conf`.

These files extract the fundamental frequency (F0), the voicing probability, and the loudness contours. The file `prosodyAcf.conf` uses the 'cPitchACF' component to extract the fundamental frequency via an autocorrelation and cepstrum based method. The file `prosodyShs.conf` uses the 'cPitchShs' component to extract the fundamental frequency via the sub-harmonic sampling algorithm (SHS). Both configurations set the CSV format as output format. An example command-line is given here:

```
SMILExtract -C config/prosodyShs.conf -I input.wav -O prosody.csv
```

2.5.6 Extracting features for emotion recognition

Since openSMILE is used by the openEAR project [EWS09] for emotion recognition, various standard feature sets for emotion recognition are available as openSMILE configuration files.

The INTERSPEECH 2009 Emotion Challenge feature set The INTERSPEECH 2009 Emotion Challenge feature set (see [SSB09]) is represented by the configuration file `config/emo_IS09.conf`. It contains 384 features as statistical functionals applied to low-level descriptor contours. The features are saved in Arff format (for WEKA), whereby new instances are appended to an existing file (this is used for batch processing, where openSMILE is repeatedly called to extract features from multiple files to a single feature file). The names of the 16 low-level descriptors, as they appear in the Arff file, are documented in the following list:

pcm_RMSenergy Root-mean-square signal frame energy

mfcc Mel-Frequency cepstral coefficients 1-12

pcm_zcr Zero-crossing rate of time signal (frame-based)

voiceProb The voicing probability computed from the ACF.

F0 The fundamental frequency computed from the Cepstrum.

The suffix **_sma** appended to the names of the low-level descriptors indicates that they were smoothed by a moving average filter with window length 3. The suffix **_de** appended to **_sma** suffix indicates that the current feature is a 1st order delta coefficient (differential) of the smoothed low-level descriptor. The names of the 12 functionals, as they appear in the Arff file, are documented in the following list:

max The maximum value of the contour

min The minimum value of the contour

range = max-min

maxPos The absolute position of the maximum value (in frames)

minPos The absolute position of the minimum value (in frames)

amean The arithmetic mean of the contour

linregc1 The slope (m) of a linear approximation of the contour

linregc2 The offset (t) of a linear approximation of the contour

linregerrQ The quadratic error computed as the difference of the linear approximation and the actual contour

stddev The standard deviation of the values in the contour

skewness The skewness (3rd order moment).

kurtosis The kurtosis (4th order moment).

The INTERSPEECH 2010 Paralinguistic Challenge feature set The INTERSPEECH 2010 Paralinguistic Challenge feature set (see Proceedings of INTERSPEECH 2010) is represented by the configuration file `config/IS10_paraling.conf`. The set contains 1 582 features which result from a base of 34 low-level descriptors (LLD) with 34 corresponding delta coefficients appended, and 21 functionals applied to each of these 68 LLD contours (1 428 features). In addition, 19 functionals are applied to the 4 pitch-based LLD and their four delta coefficient contours (152 features). Finally the number of pitch onsets (pseudo syllables) and the total duration of the input are appended (2 features).

The features are saved in Arff format (for WEKA), whereby new instances are appended to an existing file (this is used for batch processing, where openSMILE is repeatedly called to extract features from multiple files to a single feature file). The names of the 34 low-level descriptors, as they appear in the Arff file, are documented in the following list:

pcm_loudness The loudness as the normalised intensity raised to a power of 0.3.

mfcc Mel-Frequency cepstral coefficients 0-14

logMelFreqBand logarithmic power of Mel-frequency bands 0 - 7 (distributed over a range from 0 to 8 kHz)

lspFreq The 8 line spectral pair frequencies computed from 8 LPC coefficients.

F0finEnv The envelope of the smoothed fundamental frequency contour.

voicingFinalUnclipped The voicing probability of the final fundamental frequency candidate. Unclipped means, that it was not set to zero when is falls below the voicing threshold.

The suffix **_sma** appended to the names of the low-level descriptors indicates that they were smoothed by a moving average filter with window length 3. The suffix **_de** appended to **_sma** suffix indicates that the current feature is a 1st order delta coefficient (differential) of the smoothed low-level descriptor. The names of the 21 functionals, as they appear in the Arff file, are documented in the following list:

maxPos The absolute position of the maximum value (in frames)

minPos The absolute position of the minimum value (in frames)

amean The arithmetic mean of the contour

linregc1 The slope (m) of a linear approximation of the contour

linregc2 The offset (t) of a linear approximation of the contour

linregerrA The linear error computed as the difference of the linear approximation and the actual contour

linregerrQ The quadratic error computed as the difference of the linear approximation and the actual contour

stddev The standard deviation of the values in the contour

skewness The skewness (3rd order moment).

kurtosis The kurtosis (4th order moment).

quartile1 The first quartile (25% percentile)

quartile2 The first quartile (50% percentile)

quartile3 The first quartile (75% percentile)

iqr1-2 The inter-quartile range: quartile2-quartile1

iqr2-3 The inter-quartile range: quartile3-quartile2

iqr1-3 The inter-quartile range: quartile3-quartile1

percentile1.0 The outlier-robust minimum value of the contour, represented by the 1% percentile.

percentile99.0 The outlier-robust maximum value of the contour, represented by the 99% percentile.

pctlrange0-1 The outlier robust signal range 'max-min' represented by the range of the 1% and the 99% percentile.

upleveltime75 The percentage of time the signal is above (75% * range + min).

upleveltime90 The percentage of time the signal is above (90% * range + min).

The four pitch related LLD (and corresponding delta coefficients) are as follows (all are 0 for unvoiced regions, thus functionals are only applied to voiced regions of these contours):

F0final The smoothed fundamental frequency contour

jitterLocal The local (frame-to-frame) Jitter (pitch period length deviations)

jitterDDP The differential frame-to-frame Jitter (the 'Jitter of the Jitter')

shimmerLocal The local (frame-to-frame) Shimmer (amplitude deviations between pitch periods)

19 functionals are applied to these 4+4 LLD, i.e. the set of 21 functionals mentioned above without the minimum value (the 1% percentile) and the range.

The INTERSPEECH 2011 Speaker State Challenge feature set The configuration file for this set can be found in `config/IS11_speaker_state.conf`.

Details on the feature set will be added to the openSMILE book soon. Meanwhile, we refer to the Challenge paper:

> Björn Schuller, Anton Batliner, Stefan Steidl, Florian Schiel, Jarek Krajewski: "The INTERSPEECH 2011 Speaker State Challenge", Proc. INTERSPEECH 2011, ISCA, Florence, Italy, pp. 3201-3204, 28.-31.08.2011.

The INTERSPEECH 2012 Speaker Trait Challenge feature set The configuration file for this set can be found in `config/IS12_speaker_trait.conf`.

Details on the feature set will be added to the openSMILE book soon. Meanwhile, we refer to the Challenge paper:

> Björn Schuller, Stefan Steidl, Anton Batliner, Elmar Nöth, Alessandro Vinciarelli, Felix Burkhardt, Rob van Son, Felix Weninger, Florian Eyben, Tobias Bocklet, Gelareh Mohammadi, Benjamin Weiss: "The INTERSPEECH 2012 Speaker Trait Challenge", Proc. INTERSPEECH 2012, ISCA, Portland, OR, USA, 09.-13.09.2012.

The INTERSPEECH 2013 ComParE Challenge feature set The configuration file for this set can be found in `config/IS13_ComParE.conf`. A configuration that extracts only the low-level descriptors of the ComParE feature set is provided in `config/IS13_ComParE_lld.conf`. The configuration for the vocaliations (laughter, etc.) sub-challenge is also included in `config/IS13_ComParE`

Details on the feature set will be added to the openSMILE book soon. Meanwhile, we refer to the Challenge paper:

> Björn Schuller, Stefan Steidl, Anton Batliner, Alessandro Vinciarelli, Klaus Scherer, Fabien Ringeval, Mohamed Chetouani, Felix Weninger, Florian Eyben, Erik Marchi, Marcello Mortillaro, Hugues Salamin, Anna Polychroniou, Fabio Valente, Samuel Kim: "The INTERSPEECH 2013 Computational Paralinguistics Challenge: Social Signals, Conflict, Emotion, Autism", to appear in Proc. INTERSPEECH 2013, ISCA, Lyon, France, 2013.

The MediaEval 2012 TUM feature set for violent video scenes detection The feature set for the work on violent scenes detection in popular Hollywood style movies as presented in:

> Florian Eyben, Felix Weninger, Nicolas Lehment, Gerhard Rigoll, Björn Schuller: "Violent Scenes Detection with Large, Brute-forced Acoustic and Visual Feature Sets", Proc. MediaEval 2012 Workshop, Pisa, Italy, 04.-05.10.2012.

can be found for various settings in `config/mediaeval2012_tum_affect`.

The file `MediaEval_Audio_IS12based_subwin2.conf` contains the configuration which extracts the static audio features from 2 second sub-windows. `MediaEval_Audio_IS12based_subwin2_step0.5.` extracts the same features, but for overlapping 2 second windows with a shift of 0.5 seconds. For the video features the file `MediaEval_VideoFunctionals.conf` is provided, which requires a CSV file containing the low-level descriptors (can be extracted with openCV) as input and outputs and ARFF file with the video features as used in the paper.

The openSMILE/openEAR 'emobase' set The old baseline set (see the 'emobase2' set for the new baseline set) of 988 acoustic features for emotion recognition can be extracted using the following command:

```
SMILExtract -C config/emobase.conf -I input.wav -O output.arff
```

This will produce an ARFF file with a header containing all the feature names and one instance, containing a feature vector for the given input file. To append more instances to the same ARFF file, simply run the above command again for different (or the same) input files. The ARFF file will have a dummy class label called emotion, containing one class *unknown* by default. To change this behaviour and assign custom classes and class labels to an individual instance, use a command-line like the following:

```
SMILExtract -C config/emobase.conf -I inputN.wav -O output.arff -instname
inputN -classes {anger,fear,disgust} -classlabel anger
```

Thereby the parameter `-classes` specifies the list of nominal classes including the {} characters, or can be set to *numeric* for a numeric (regression) class. The parameter `-classlabel` specifies the class label/value of the instance computed from the currently given input (-I). For further information on these parameters, please take a look at the configuration file `emobase.conf` where these command-line parameters are defined.

The feature set specified by `emobase.conf` contains the following low-level descriptors (LLD): Intensity, Loudness, 12 MFCC, Pitch (F_0), Probability of voicing, F_0 envelope, 8 LSF (Line Spectral Frequencies), Zero-Crossing Rate. Delta regression coefficients are computed from these LLD, and the following functionals are applied to the LLD and the delta coefficients: Max./Min. value and respective relative position within input, range, arithmetic mean, 2 linear regression coefficients and linear and quadratic error, standard deviation, skewness, kurtosis, quartile 1–3, and 3 inter-quartile ranges.

The large openSMILE emotion feature set For extracting a larger feature set with more functionals and more LLD enabled (total 6 552 features), use the configuration file

`config/emo_large.conf`.

Please read the configuration file and the header of the generated arff file in conjunction with the matching parts in the component reference section (4.3) for details on the contained feature set. A documentation has to be yet written, volunteers are welcome!

The openSMILE 'emobase2010' reference set This feature set is based on the INTERSPEECH 2010 Paralinguistic Challenge feature set. It is represented by the file

`config/emobase2010.conf`.

A few tweaks have been made regarding the normalisation of duration and positional features. This feature set contains a greatly enhanced set of low-level descriptors, as well as a carefully selected list of functionals compared to the older 'emobase' set. This feature set is recommended as a reference for comparing new emotion recognition feature sets and approaches to, since it represents a current state-of-the-art feature set for affect and paralinguistic recognition.

The set contains 1 582 features (same as the INTERSPEECH 2010 Paralinguistic Challenge set) which result from a base of 34 low-level descriptors (LLD) with 34 corresponding delta coefficients appended, and 21 functionals applied to each of these 68 LLD contours (1 428 features). In addition, 19 functionals are applied to the 4 pitch-based LLD and their four delta coefficient contours (152 features). Finally the number of pitch onsets (pseudo syllables) and the total duration of the input are appended (2 features). The only difference to the INTERSPEECH 2010 Paralinguistic Challenge set is the normalisation of the 'maxPos' and 'minPos' features which are normalised to the segment length in the present set.

Audio-visual features based on INTERSPEECH 2010 audio features. The folder `config/audiovisual` contains two configuration files for video features (video.conf) and synchronised audio-visual feature extraction (audiovideo.conf). These files are used for the examples in section 2.7. The audio features and the set of functionals which is applied to both the audio and the video low-level features is taken from the INTERSPEECH 2010 Paralinguistic Challenge feature set (section 2.5.6 for details).

The video features contain RGB and HSV colour histograms, local binary patterns (LBP), and an optical flow histogram. They can either be extracted from the complete image or only the facial region. The latter is automatically detected via the OpenCV face detector. The face detection can be controlled in the configuration file `audiovideo.conf` in the section

```
[openCVSource:cOpenCVSource]
```

with the option `extract_face`. The number of histogram bins can also be changed in this section.

2.6 Using Portaudio for live recording/playback

The components `cPortaudioSource` and `cPortaudioSink` can be used as replacements for `cWaveSource` and `cWaveSink`. They produce/expect data in the same format as the wave components.

Two example configuration files are provided which illustrate the basic use of PortAudio for recording live audio to file (`config/demo/audiorecorder.conf`) and for playing live audio from a file (`config/demo/audioplayer.conf`).

Using these configurations is very simple. To record audio to a file, type:

```
SMILExtract -C config/demo/audiorecorder.conf -sampleRate 44100 -channels
   2 -O output.wave
```

To stop the recording, quit the program with Ctrl+C. To play the recorded audio use this command:

```
SMILExtract -C config/demo/audioplayer.conf -I output.wave
```

On top of these two simple examples, a live feature extraction example is provided, which captures live audio and extracts prosodic features (pitch and loudness contours) from the input. The features are saved to a CSV file. To use this configuration, type:

```
SMILExtract -C config/liveProsodyAcf.conf
```

The recording has started once you see the message

```
(MSG) [2] in cComponentManager : starting single thread processing loop
```

You can now speak an example sentence or play some music in front of your microphone. When you are done, press Ctrl+C to terminate openSMILE. A CSV file called `prosody.csv` has now been created in the current directory (use the `-O` command-line option to change the file name). You can now plot the loudness and pitch contours using gnuplot, for example, as is described in the next section.

2.7 Extracting features with openCv

openSMILE can extract audio and video features simultaneously and time synchronised. An example is provided in the configuration file `config/audiovisual/audiovideo.conf`.

For this example to work, you need:

- a video file in a supported format (rule of thumb: if FFMPEG can open it, openCV/openSMILE can too)
- the audio track of the video file in a separate file (.wav format)

You can use mplayer or ffmpeg for example to extract the audio track of the video. These tools often create wave files with a WAVEext header, which unfortunately is not (yet) supported by openSMILE . Thus, openSMILE will complain that the wave file is not in the correct format. If this happens you can convert the wave files with the commandline tool sox (available both for Linux and Windows) to a supported format:

```
sox input.wav -c 1 -2 -s output.wav
```

With some (older) versions of sox simply

```
sox input.wav output.wav
```

will also work. However, recent versions tend to do a simple copy operation when the source parameters match the input parameters. We thus have to change the parameters, by e.g., reducing the number of channels to 1 and possibly converting to 16-bit signed integer sample format.

Once openSMILE accepts the wave file, the analysis can be started by executing (all on a single line):

```
   ./SMILExtract -C config/av_fusion/audiovideo.conf \
-V VIDEO_FILE -A AUDIO_FILE -N NAME -a AGE \
-g GENDER -e ETHNICITY -O VIDEO_ARFF -P AUDIO_ARFF
```

in a shell, whereas the following replacements should be done:

- AUDIO_FILE and VIDEO_FILE should be replaced by the path to the respective audio (.wav) and video input files (can contain the audio track, it is ignored by OpenCV)
- NAME denotes the title for the arff instance and can be freely chosen from alphanumeric characters and _.
- AGE, GENDER and ETHNICITY represent the ground-truth class labels for this particular pair of audio/video, if you want them to be included in an ARFF file, which you use to train a classifier.
- VIDEO_ARFF and AUDIO_ARFF should be replaced by the desired filename for the respective output arffs.

After execution, two new files will have been created: VIDEO_ARFF and AUDIO_ARFF which contain the audio and video descriptors respectively, time synchronised. If those files already exist, the content is appended accordingly.

2.8 Visualising data with Gnuplot

In order to visualise feature contours with gnuplot, you must have perl5 and gnuplot installed. On Linux perl should be installed by default (if not, check your distributions documentation on how to install perl), and gnuplot can be either installed via your distribution's package manager (On Ubuntu: `sudo apt-get install gnuplot-nox`), or compiled from the source (http://www.gnuplot.info). For windows, gnuplot binaries are available from the project webpage (http://www.gnuplot.info). For perl5, download the ActiveState Perl distribution from http://www.activestate.com/ and install it. Moreover, you will need the bash shell to execute the `.sh` scripts (if you don't have the bash shell, you must type these commands manually on the windows command-prompt).

A set of scripts, which are included with openSMILE in the directory `scripts/gnuplot`, simplifies the file conversion process and simple plotting tasks. The set of these scripts is by far not complete, but we feel this is not necessary. These script serve as templates which you can easily adjust for you own tasks. They convert (transpose) the CSV files generated by openSMILE to a representation which can be read by gnuplot directly and call gnuplot with some default plot scripts as argument.

For some ready-to-use examples, see the scripts `plotchroma.sh` to plot Chroma features as a 'Chromagram' and `plotaudspec.sh` to plot an auditory spectrum. The following commands give a step-by-step guide on how to plot Chroma features and an auditory spectrum (*Note*: we assume that you execute these commands in the top-level directory of the openSMILE distribution, otherwise you may need to adjust the paths):
First, you must extract chroma features from some music example file, e.g.:

```
SMILExtract -C config/chroma_fft.conf -I wav_samples/music01.wav -O chroma
    .csv
```

Then you can plot them with

```
cd scripts/gnuplot
sh plotchroma.sh ../../chroma.csv
```

If your gnuplot installation is set up correctly you will now see a window with a 'Chromagram' plot.
For the auditory spectrum, follow these steps:

```
SMILExtract -C config/audspec.conf -I wav_samples/music01.wav -O audspec.
    csv
```

Then you can plot them with

```
cd scripts/gnuplot
sh plotaudspec.sh ../../audspec.csv
```

If your gnuplot installation is set up correctly you will now see a window with a auditory spectrogram plot.

Two more universally usable scripts are also included. First we describe the `plotmatrix.sh` script. Its usage is the same as for the `plotaudspec.sh` and `plotchroma.sh` scripts:

```
cd scripts/gnuplot
sh plotmatrix.sh <filename of csv-file saved by openSMILE>
```

This script plots the matrix as a 2D-surface map with the gnuplot script `plotmatrix.gp`, thereby using a grey-scale color-map, displaying time in frames on the x-axis and the feature index on the y-axis.

To plot feature contours such as pitch or energy, first extract some example contours using the `prosodyAcf.conf` configuration, for example (You can also use the live feature extractor configuration, mentioned in the previous section):

```
SMILExtract -C config/prosodyAcf.conf -I wav_samples/speech02.wav -O
    prosody.csv
```

Next, you can plot the pitch contour with

```
cd scripts/gnuplot
sh plotcontour.sh 3 ../../prosody.csv
```

The general syntax of the `plotcontour.sh` script is the following:

```
sh plotcontour.sh <index of feature to plot> <filename of CSV file
    containing features>
```

The index of the feature to plot can be determined by opening the CSV file in a text editor (gedit, kate, vim, or Notepad++ on Windows, for example). The first two features in the file are always 'frameIndex' and 'frameTime' with indices 0 and 1.

Plotting of features in real-time when performing on-line feature extraction is currently not supported. However, since features are extracted incrementally anyways, it is possible to write a custom output plugin, which passes the data to some plotting application in real-time, or plots the data directly using some GUI API.

Chapter 3

Description of algorithms

As you might have noted, this document does not describe details of the feature extraction algorithms implemented in openSMILE. There are two resources to get more details on the algorithms:

1. Read the source!
2. Read the book *Real-time Speech and Music Classification by Large Audio Feature Space Extraction* by F. Eyben published by Springer[1] (eBook ISBN: 978-3-319-27299-3). All important algorithms are described in detail there and a precise and most up-to-date summary of standard acoustic parameter sets up to ComParE 2013 and GeMAPS is given. It is also a good reading for people who are new to the field of audio analysis and machine learning for audio.

[1] `http://www.springer.com/de/book/9783319272986`

Chapter 4

Reference section

This section includes a list of components included with openSMILE and detailed documentation for each component (section 4.3). The components are grouped in logical groups on behalf of their functionality. The following section (4.1) documents available command-line options and describes the general usage of the `SMILExtract` command-line tool. A documentation of the configuration file format can be found in section 4.2.

4.1 General usage - SMILExtract

The `SMILExtract` binary is a very powerful command-line utility which includes all the built-in openSMILE components. Using a single ini-style configuration file, various modes of operation can be configured. This section describes the command-line options available when calling `SMILExtract`. Some options take an optional parameter, denoted by [parameter-type], while some require a mandatory parameter, denoted by <parameter-type>.

Usage: SMILExtract [-option (value)] ...

-C, -configfile	*<string>* Path to openSMILE config file. *Default:* 'smile.conf'
-l, -loglevel	*<int>* Verbosity level of log messages (MSG, WRN, ERR, DBG) (0-9). 1: only important messages, 2,3: more detailed messages, 4,5: very detailed debug messages (if -debug is enabled), 6+: currently unused. *Default:* 2
-d, -debug	Show debug log-messages (DBG) (this is only available if the binary was compiled with the _DEBUG preprocessor flag) *Default:* off
-h	Show usage information and exit.
-L, -components	Show full component list (this list includes plugins, if they are detected correctly), and exit.
-H, -configHelp	[*componentName:string*]

Show the documentation of configuration options of all available components - including plugins - and exit. If the optional string parameter is given, then only documentation of components beginning with the given string will be shown.

-configDflt [*string*]
Show default configuration file section templates for all components available (empty parameter), or selected components beginning with the string given as parameter. In conjunction with the 'cfgFileTemplate' option a comma separated list of components can be passed as parameter to 'configDflt', to generate a template configuration file with the listed components.

-cfgFileTemplate Experimental functionality to print a configuration file template containing the components specified in a comma separated string as argument to the 'configDflt' option.

-cfgFileDescriptions If this option is set, then option descriptions will be included in the generated template configuration files.
Default: off

-c, -ccmdHelp Show user defined command-line option help in addition to the standard usage information (as printed by '-h'). Since openSMILE provides means to define additional command-line options in the configuration file, which are available only after parsing the configuration file, and additional command-line option has been introduced to show a help on these options. A typical command-line to show this help would be `SMILExtract -c -C myconfigfile.conf`.

-logfile *<string>*
Specifies the path and filename of the log file to use. Make sure the path of the log-file is writeable.
Default: 'smile.log'

-appendLogfile If this option is specified, openSMILE will append log messages to an existing log-file instead of overwriting the log-file at every program start (which is the default behaviour).

-nologfile If this option is specified, openSMILE does not write to a log file (use this on a read-only filesystems, for example).

-noconsoleoutput If this option is specified, no log-output is displayed in the console. Logging to the log file is not affected by this option, see 'nologfile' for disabling the log-file.

-t, -nticks *<int>*
Number of ticks (=component loop iterations) to process (-1 = infinite) (Note: this only works for single thread processing, i.e. nThreads=1 set in the config file). This option is not intended for normal use. It is for debugging component execution code only.
Default: -1

4.2 Understanding configuration files

openSMILE configuration files follow an INI-style file format. The file is divided into sections, which are introduced by a section header:

```
[sectionName:sectionType]
```

The section header, opposed to standard INI-format, always contains two parts, the section name (first part) and the section type (second part). The two parts of the section header are separated by a colon (:). The section body (the part after the header line up to the next header line or the end of the file) contains attributes (which are defined by the section type; a description of the available types can be seen using the `-H` command-line option as well as in section 4.3). Attributes are given as `name = value` pairs. An example of a generic configuration file section is given here:

```
[instancename:configType]  <-- this specifies the header
variable1 = value          <-- example of a string variable
variable2 = 7.8            <-- example of a "numeric" variable
variable3 = X              <-- example of a "char" variable
subconf.var1 = myname      <-- example of a variable in a sub type
myarr[0] = value0          <-- example of an array
myarr[1] = value1
anotherarr = value0;value1 <-- example of an implicit array
noarray = value0\;value1   <-- use \; to quote the separator ';'
strArr[name1] = value1     <-- associative arrays, name=value pairs
strArr[name2] = value2
; line-comments may be expressed by ; // or # at the beginning
```

Principally the config type names can be any arbitrary names. However, for consistency the names of the components and their corresponding configuration type names are identical. Thus, to configure a component `cWaveSource` you need a configuration section of type `cWaveSource`.

In every openSMILE configuration file there is one mandatory section, which configures the component manager. This is the component, which instantiates and runs all other components. The following sub-section describes this section in detail.

4.2.1 Enabling components

The components which will be run, can be specified by configuring the `cComponentManager` component, as shown in the following listing (the section always has to be called `componentInstances`):

```
[componentInstances:cComponentManager] <-- don't change this
; one data memory component must always be specified!
; the default name is 'dataMemory'
; if you call your data memory instance 'dataMemory',
; you will not have to specify the reader.dmInstance variables
; for all other components!
; NOTE: you may specify more than one data memory component
; configure the default data memory:
instance[dataMemory].type=cDataMemory
; configure an example data source (name = source1):
instance[source1].type=cExampleSource
```

The associative array `instance` is used to configure the list of components. The component instance names are specified as the array keys and are freely definable. They can contain all characters except for], however, it is recommended to only use alphanumeric characters, _, and -. The component types (i.e. which component to instantiate), are given as value to the `type` option.

Note: for each component instance specified in the `instance` array a configuration section in the file *must* exist (*except for the data memory components!*), even if it is empty (e.g. if you want to use default values only). In this case you need to specify only the header line `[name:type]`.

4.2.2 Configuring components

The parameters of each component can be set in the configuration section corresponding to the specific component. For a wave source, for example, (which you instantiate with the line

```
instance[source1].type = cWaveSource
```

in the component manager configuration) you would add the following section (note that the name of the configuration section must match the name of the component instance, and the name of the configuration type must match the component's type name):

```
[source1:cWaveSource]
 ; the following sets the level this component writes to
 ; the level will be created by this component
 ; no other components may write to a level having the same name
writer.dmLevel = wave
filename = input.wav
```

This sets the file name of the wave source to `input.wav`. Further, it specifies that this wave source component should write to a data memory level called `wave`. Each openSMILE component, which processes data has at least a data reader (of type cDataReader), a data writer (of type cDataWriter), or both. These sub-components handle the interface to the data memory component(s). The most important option, which is mandatory, is `dmLevel`, which specifies the level to write to or to read from. Writing is only possible to one level and only one component may write to each level. We would like to note at this point that the levels do not have to be specified implicitly by configuring the data memory – in fact, the data memory is the only component which does not have and does not require a section in the configuration file – rather, the levels are created implicitly through `writer.dmLevel = newlevel`. Reading is possible from more than one level. Thereby, the input data will be concatenated frame-wise to one single frame containing data from all input levels. To specify reading from multiple levels, separate the level names with the array separator ';', e.g.:

```
reader.dmLevel = level1;level2
```

The next example shows the configuration of a `cFramer` component `frame`, which creates (overlapping) frames from raw wave input, as read by the wave source:

```
[frame:cFramer]
reader.dmLevel=wave
writer.dmLevel=frames
frameSize = 0.0250
frameStep = 0.010
```

The component reads from the level `wave`, and writes to the level `frames`. It will create frames of 25 ms length at a rate of 10 ms. The actual frame length in samples depends on the sampling rate, which will be read from meta-information contained in the `wave` level. For more examples please see section 2.5.

4.2.3 Including other configuration files

To include other configuration files into the main configuration file use the following command on a separate line at the location where you want to include the other file:

```
\{path/to/config.file.to.include\}
```

This include command can be used anywhere in the configuration file (as long it is on a separate line). It simply copies the lines of the included file into the main file while loading the configuration file into openSMILE .

4.2.4 Linking to command-line options

openSMILE allows for defining of new command-line options for the `SMILExtract` binary in the configuration file. To do so, use the `\cm` command as value, which has the following syntax:

```
\cm[longoption(shortoption){default value}:description text]
```

The command may be used as illustrated in the following example:

```
[exampleSection:exampleType]
myAttrib1 = \cm[longoption(shortopt){default}:descr. text]
myAttrib2 = \cm[longoption{default}:descr. text]
```

The `shortopt` argument and the `default value` are optional. Note that, however, either `default` and/or `descr. text` are required to define a *new* option. If neither of the two is specified, the option will not be added to the command-line parser. You can use this mode to reference options that were already added, i.e. if you want to use the value of an already existing option which has been defined at a prior location in the config file:

```
[exampleSection2:exampleType]
myAttrib2 = \cm[longoption]
```

An example for making a filename configurable via the command-line, is given here:

```
filename = \cm[filename(F){default.file}:use this option to specify the
    filename for the XYZ component]
```

You can call `SMILExtract -c -C yourconfigfile.conf` to see your command-line options appended to the general usage output.

Please note: When specifying command-line options as a value to an option, the `\cm` command is the only text allowed at the right side of the equal sign! Something like `key = value \cm[...]` is currently not allowed. We understand that this may be a useful feature, thus it may appear in one of the following releases. The \cm command may also only appear in the value field of an assignment *and (since version 2.0) also instead of a filename in the config file include command.*

4.2.5 Defining variables

This feature is not yet supported, but is planned for addition. This should help avoid duplicate values and increase maintainability of configuration files. A current workaround is to define a commandline option with a given default value instead of a variable.

4.2.6 Comments

Single line comments may be initiated by the following characters at the beginning of the line (only whitespaces may follow the characters): `; # // %`

If you want to comment out a partial line, please use `//`. Everything following the double slash on this line (and the double slash itself) will be considered a comment and will be ignored.

Multi-line comments are now supported via the C-style sequences `/*` and `*/`. In order to avoid parser problems here, please make sure these sequences are on a separate line, e.g.

```
/*
[exampleSection2:exampleType]
myAttrib2 = \cm[longoption]
*/
```

and not:

```
/*[exampleSection2:exampleType]
myAttrib2 = \cm[longoption]*/
```

The latter case is supported, however, you must ensure that the closing `*/` is *not* followed by *any* whitespaces.

4.3 Component description and on-line help

This section contained a list of all components and available options for each component in previous versions of this document. Due to the rapid development, new options and new components are frequently added, which caused this document to quickly get out of sync. We decided therefore not to maintain this section anymore and instead remove it completely to avoid confusing users with outdated documentation.

The only way to get information about components is now through openSMILE's online help feature. From the commandline you can get a list of available components (and a short description for each) with the command

```
SMILExtract -L
```

All available configuration options for a specific component (replace cMyComponentName by the actual name), as well as the description of their use and meaning, can be obtained with the command

```
SMILExtract -H cMyComponentName
```

4.4 Feature names

This section will soon contain a table which maps feature names (as generated by default settings), onto a corresponding description or links to the component reference sections where the features are described. Please note, that this information is also contained in the documentation of each component, but is summarised here for easier viewing. You must also be aware that most feature names in openSMILE can be changed via options in the configuration files. Thus, the names listed in this section might not be those you get with your configuration file. However, we consider it a bad practice to use your own names for the features if there is no obvious reason to do so. Using the default names ensures compatibility of feature files and feature selection lists.

openSMILE follows a strict naming scheme for features (data fields). Each component (except the sink components), assigns names to its output fields. All cDataProcessor descendants

have two options to control the naming behaviour, namely 'nameAppend' and 'copyInputName'. 'nameAppend' specifies a suffix which is appended to the field name of the previous level. A '–' is inserted between the two names (if 'nameAppend' is not empty or (null)). 'copyInputName' controls whether the input name is copied and the suffix 'nameAppend' and any internal hard-coded names are appended (if it is set to 1), or if the input field name is discarded and only the component's internal names and an appended suffix are used.

The field naming scheme is illustrated by the following example. Let's assume you start with an input field 'pcm'. If you then compute delta regression coefficients from it, you end up with the name 'pcm-de'. If you apply functionals (extreme values max and min only), then you will end up with two new fields: 'pcm-de-max' and 'pcm-de-min'. Theoretically, if the 'copyInputName' is always set, and a suitable suffix to append is specified, the complete processing chain can be deducted from the field name. In practice, however, this would lead to quite long and redundant feature names, since most speech and music features base on framing, windowing, and spectral transformation. Thus, most of these components do not append anything to the input name and do only copy the input name. In order to discard the 'pcm' from the wave input level, components that compute features such as mfcc, pitch, etc. discard the input name and use only a hard-coded name or a name controlled via 'nameAppend'.

Chapter 5

Developer's Documentation

The developer's documentation has not yet been included in this document. Fragments of the documentation covering various aspects briefly are found in the `doc/developer` directory.

Writing plugins If you are interested in writing plugins for openSMILE, read the document `doc/developer/implementing_components.txt` and write a component cpp and hpp file. Then have a look at the Makefiles (Linux) in the `plugindev` directory for building your plugin on linux, and the Visual Studio Solution files (openSmilePlugin) for windows in the `ide/vs05/` folder.

The main source file of a plugin is the `plugindev/pluginMain.cpp` file. This file includes the individual component files this plugin shall contain, similar to the component list in the `componentManager.cpp` file, which manages the openSMILE built-in components.

Chapter 6

Additional Support

If you have questions which are not covered by this documentation please contact Florian Eyben (fe at audeering.com).

Chapter 7

Acknowledgement

The development of openSMILE and openEAR has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 211486 (SEMAINE).

Bibliography

[EWS09] Florian Eyben, Martin Wöllmer, and Björn Schuller. openear - introducing the munich open-source emotion and affect recognition toolkit. In *Proceedings of the 4th International HUMAINE Association Conference on Affective Computing and Intelligent Interaction 2009 (ACII 2009)*, volume I, pages 576–581. IEEE, 2009.

[SER07] Björn Schuller, Florian Eyben, and Gerhard Rigoll. Fast and robust meter and tempo recognition for the automatic discrimination of ballroom dance styles. In *Proceedings ICASSP 2007*, Honolulu, Hawaii, 2007.

[SSB09] Björn Schuller, S. Steidl, and Anton Batliner. The interspeech 2009 emotion challenge. In *Interspeech (2009), ISCA, Brighton, UK*, 2009.