TDF / TimsData SDK documentation

License

This is a preview given to dedicated partners only. The documentation is not yet complete. Please do not distribute in any way at this point.

This file documents TDF (Tims Data Format) v3.0.

Introduction

Bruker TIMS data is stored in two files:

1. **analysis.tdf** – an SQLite database containing all metadata, accessible from practically any programming language with a corresponding SQLite API,
2. **analysis.tdf_bin** – containing peak data in a compressed binary form; accessible through a DLL enclosed in this SDK for Windows and Linux, which exports a simple C-language API and is thus wrappable for practically any programming language.

These two files are always stored together in a **folder with a .d extension**; this folder also contains other files produced Bruker's acquisition control software. Such a folder always contains a single **analysis**, i.e., a single direct-infusion or LC experiment.

The TDF stores direct-infusion and LC experiments in the same way: it contains a number of so-called **frames**, each recorded at a specific analysis time. A frame is a set of mobility-resolved mass spectra, called **scans**. A scan is a mass spectrum in the form of a **peak list**, i.e., a list of mass /intensity pairs. It is important to stress that these mass spectra do *not* represent continuously sampled detector data; the digitizer used in our instruments directly performs peak picking on the sampled data and only reports peak positions and intensities, which are then stored in the TDF (this is called the **peak-list mode**).

Requirements

The SQLite file can be read from practically any programming language using the SQLite API preferred by the application programmer. Note: we use the relatively recent WITHOUT ROWID extension in our SQLite schema, which requires SQLite >= 3.8.2.

The binary file can (at the moment) only be accessed using the DLLs provided by this SDK, for Windows 32-/64-bit, and Linux 64-bit. There is no need to "install" the DLL in any way. It is sufficient to copy it into the same directory as the executable of your application.

Under Windows, the DLL requires that Microsoft's "Visual C++ Redistributable for Visual Studio 2017" be installed. However, you are free to use any compiler you want to code your application; it may still be linked to the timsdata DLL, even if you're using a different version of Visual C++ (or an altogether

different programming language), because the DLL's public interface consists of simple C-style functions only.

Versioning

Every TDF stores a **version number ("major.minor")**. In future evolutions of this schema, we will guarantee backwards-compatibility in the sense that every application that can read a TDF in format "a.b" can also read the version "a.c" without code changes, even if c > b.

We do this by only *adding* new tables, or new columns to existing tables (not removing columns), and not changing the semantics of existing fields. If we need to break any of these rules, we will increase the major number.

To ensure compatibility in this way, the application programmer should not rely on the ordering of the columns within any table, in particular, avoid using `SELECT *` to query all fields in a table, since this would change the program's behavior in the presence of new or re-ordered columns.

Documentation

For now, please refer to the enclosed source files to get an idea how to work with TDF files:

- the C header file, declaring and documenting the functions exported by the DLL
- a C++ header file that wraps around the C interface to make its usage more convenient
- a Python wrapper and example program

The basic idea is that you go into the `Frames` table of the SQLite file, extract the `Id` of the frame(s) you want to read, and use that to ask the DLL to give you the peak-list data via the `tims_read_scans_v2()` call (s. timsdata.h).

The peak-list data is then available as pairs of **index** and intensity. The index refers to the position of the peak in the sampled waveform before peak picking in the digitizer. It can be converted to m/z values using the `tims_index_to_mz()` call (s. timsdata.h). Note that the latter also takes the frame number as an input parameter, because the index→m/z transformation can, in principle, be different for each frame recorded in the TDF, as a consequence of online recalibration procedures. Depending on your application, these differences may be negligibly small, however.