The QisMLib

April 6, 2018

What is QisMLib ?

QisMLib is system of dynamic libraries (DLs - .dll on Windows, .so on Linux) and C++ APIs implemented by those DLs, centered around a core GDSII and OASIS data processor. A client application can use these APIs to extract and process all kinds of information from GDSII/OASIS files. It is the next step in the evolution of Artwork's QisLib with support for multiple GDSII/OASIS databases at any given time and multiple independent and thread-safe spatial query objects (exploders) (Web Page: http://www.artwork.com/gdsii/qislib_mt/index.htm)

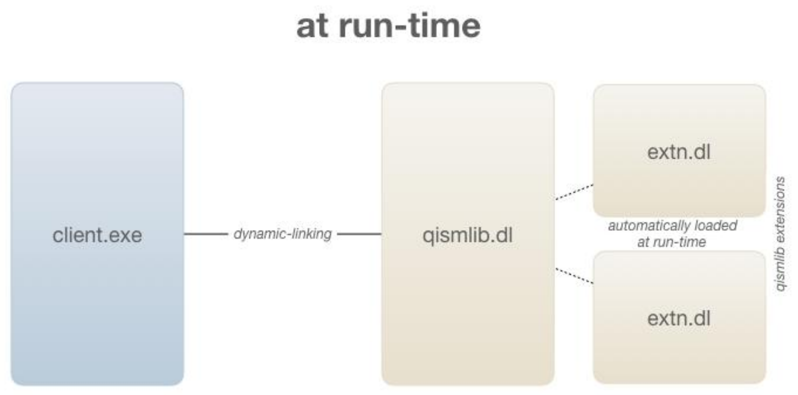
How is QisMLib structured ?

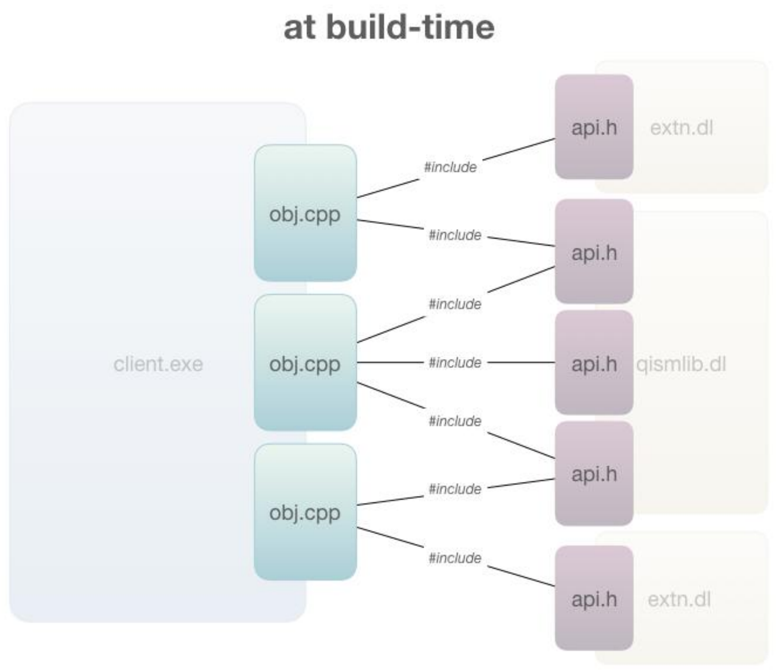
At the core of the QisMLib system is a single DL - qismlib64.dll (Windows)/libqism64.so (Linux) that hosts the GDSII/OASIS data processor and provides a core set of APIs to work with. All the other DLs in this system are implemented as **extensions (plug-ins)** that can be made available at installation time and build upon the core QisMLib APIs to offer interesting data processing functions (such as rasterization, layer synthesis, net-tracing, data extraction etc.)

The core DL provides the following APIs:

- ***QisMLib** (qismlib.h) - Gateway to the QisMLib system. Allows an application to load one or more GDSII/OASIS/DBLOAD files and get access to other APIs in the system
- **QisMFile** (qismfile.h) - API to work with a single file database already loaded into QisMLib. Multiple QisMFile objects represent multiple databases loaded into QisMLib at any given time
- ***QisMExploder** (qismexploder.h) - API to perform spatial queries (explosion) over a view (cell, layers, nesting level, window) of a qismfile object. Multiple QisMExploder objects associated with the same or different qismfile objects can perform spatial queries and extract data vectors (boundaries, paths, texts, single and arrayed cell references), each in its own thread, independent of other exploders in the system
- ***QisMBool** (qismbool.h) - API to perform multi-threaded boolean operations (union, difference, xor, intersection) and other operations such as clipping, sizing etc. on large sets of polygons
- ***QisMDraw** (qismdraw.h) - API to render a view of a qismfile object directly to the client screen (CWnd* on Windows or Pixmap on Linux) or to a GIF, XPM, Windows Bitmap formatted memory buffer

* These APIs are individually licensed. Number of licenses required for each API depends on the functionality and use of that particular API



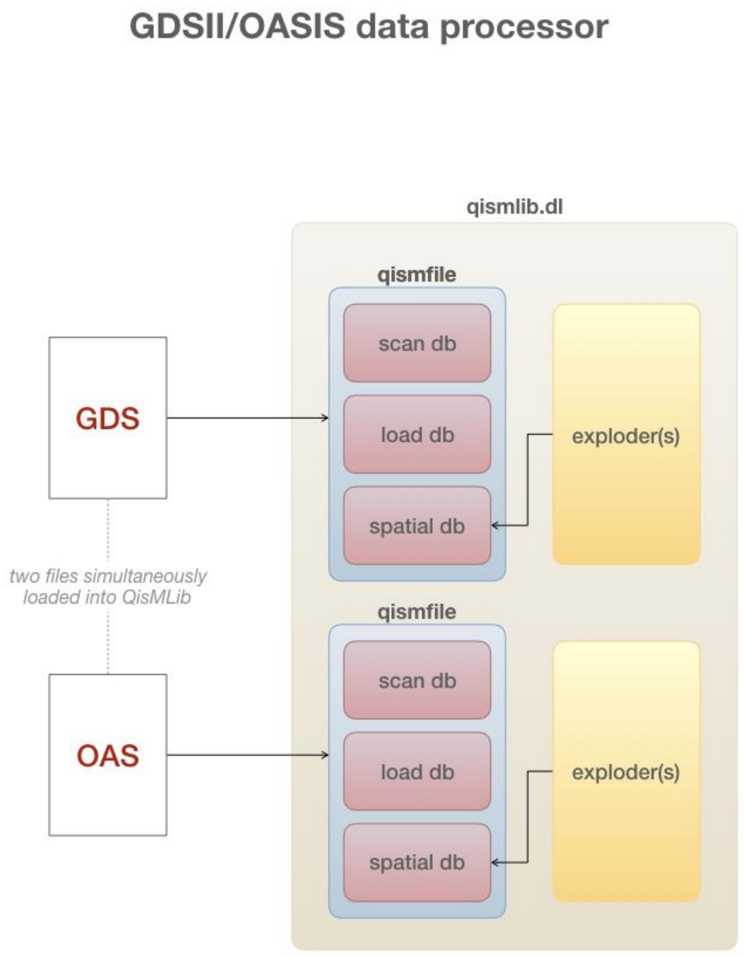


What is a GDSII/OASIS data processor (qismfile object) ?

The GDSII/OASIS data processor (a.k.a qismfile object) that sits at the heart of QisMLib is the collective state of a GDSII/OASIS file loaded into QisMLib. Each qismfile object has the following key components:

- **Scan DB** : A database that contains information about the file structure such as cells, cell hierarchy, layers etc.
- **Load DB** : The actual data that corresponds to records in the original file. If the file is loaded into memory, the Load DB is independent from the file on disk. Otherwise, the file on disk itself serves as the Load DB

- **Spatial DB** : A set of specialized quad trees that index the data present in the Load DB spatially for executing fast window based queries (explosions)



What are QisMLib extensions ?

QisMLib extensions are special DLs (plug-ins) that are automatically loaded by QisMLib during initialization at the start of the client program and automatically unloaded during close at the end of the client program. QisMLib determines which extension to load and in what order based on an entry in a Config file (default: qismlib.cfg) that says 'EXTENSION=<dl-path>', one per extension to be loaded. As the name suggest, these extensions 'extend' the QisMLib API by providing specialized data processing functions such as layer synthesis, net tracing, rasterization, hierarchical extraction etc.

QisMLib extensions can also make use of the APIs provided by other extensions to achieve their functional objectives. Each extension API is individually licensed and the number of licenses required depends on the design of that particular extension. Because they are plug-ins, extensions can be made available at installation time and while their presence enhances the capabilities of QisMLib, their absence does not inhibit the core QisMLib operations. Some of the extensions available today are:

- **QisMLayerSynth**: API to synthesize new layers based on expressions involving boolean operations between existing layers
- **QisMClipExtract**: API to extract lots of tiny clips of data in parallel as GDSII/OASIS/TIFF/BMP files on disk or image/polygon buffer in memory
- **QisMNtrc**: API to trace nets of connected METAL and VIA polygons over a pre-defined stackup
- **QisMRaster**: API to generate large high resolution monochrome bitmaps using multi-threaded rasterization and pattern recognition
- **QisMBoolFiles**: API to perform boolean operations between data from two different files
- **QisMHExtract**: API to extract hierarchical data to GDSII/OASIS file with various options for clipping and filtering

All extensions get their source data from qismfile objects already loaded into QisMLib by the client application. The extension APIs allow the client to specify which qismfile object to use for a particular operation

What platforms does QisMLib support ?

All QisMLib APIs and extensions are available on Windows 64 (built with VC++ 2008) and Linux 64 (built with GCC 4.11)

How does one learn to use the QisMLib API ?

Follow these steps to get started:

1. Contact Artwork (info@artwork.com) for assistance in understanding what's suitable for your needs and to get an installation package on the platform of your choice (64bit Windows or Linux) with the appropriate trial license codes
2. Once installed, find the following:
    a. Sample code (.cpp) for various use cases in ${install}/samplecode
    b. Fully documented API header files in ${install}/include
    c. Demo command-line application for performance and correctness evaluation in ${install}/artwork
    d. Command-line usage details and other information in ${install}/reference