The QisMLib system

> A system of C++ APIs for processing GDSII/OASIS data

- The complete API details can be found in the respective C++ header files (.h)
- All the relevant header files are placed in the **include** folder of every QisMLib installation
- QisMLib on the web
- (c) Artwork Conversion Software Inc. (www.artwork.com)
- **Current as of** : 2019-07-12

The QisMLib system
QisMLib
QisMFile
QisMExploder
QisMBool
QisMClipper
QisMExplCounter
QisMLog
QisMFileOut
QisMLayerSynth
QisMClipExtract
QisMRaster
QisMNtrc
QisMBoolFiles
QisMCorrX
QisMHExtract
QisMRTCR (GDSII only, WINDOWS only)
QisMScript (`beta`)

QisMLib

> Gateway to the QisM system

Header : qismlib.h

Binary : qismlib64.dll / libqis64.so

License : 1 x (11003) per process

- check-out -- `QisMLib_initialize_once`
- check-in -- `QisMLib_close_once`

Key features

- Work with multiple GDSII/OASIS databases at the same time
- Plug-n-play architecture gives the QisMLib system virtually unlimited extensibility without bloating the library core
- ONE library, many interfaces -- All the plug-in management happens behind-the-scenes so that the application only needs to link/load only one library -- qismlib64.dll / libqism64.so
- QisMLib can be dynamically loaded at run-time (on-demand) or linked at build time

- All APIs are designed to be feature extensible without breaking source and binary compatibility both forwards (applications can work with new versions of the lib without rebuilding) and backwards (applications will work with older versions of the lib with ability to detect missing features)
- Designed to be used in multi-threaded environments as much as possible
- With rare exceptions, all APIs work with GDSII and OASIS data alike on both 64 bit Windows and Linux
- Learn more about QisMLib

Key classes & operations ([class] operation -- description)

- `QisMLib_initialize_once` -- Initialize the QisM system **once** at the start of the program to get a handle to the QisMLib API
- `QisMLib_close_once` -- Close the QisM system **once** at the end of the program
- [`QisMLib`] `Load_file` -- Create a new database from a GDSII/OASIS/DBLOAD file
- [`QisMLib`] `Unload_file` -- Destroy a database
- [`QisMLib`] `Get_extension_api` -- Get access to other APIs in the system

QisMFile

> API to work with a database created from a GDSII/OASIS file

Header : qismfile.h

Binary : qismlib64.dll / libqis64.so

License : -

Key features

- Layers and texts can be filtered during database creation
- Data can be loaded to memory (for faster operation) or referenced on disk (for smaller memory footprint)
- Multiple databases can co-exist independently
- Multiple spatial query objects (exploders) can co-exist and work per database and for multiple databases to make best use of the available computing power (threads)

Key classes & operations ([class] operation -- description)

- [`QisMFileLoadCtrl`] `Set_layer_map` -- Filter/re-map one or more layers during database creation
- [`QisMFileLoadCtrl`] `Set_ignore_texts` -- Drop TEXT data during database creation
- [`QisMFile`] `Get_default_top_cell` -- Get the name of the (default) root cell with the deepest hierarchy and the largest extents
- [`QisMFile`] `Get_cell_list` -- Get a list of cell names in the database
- [`QisMFile`] `Get_top_cell_list` -- Get a list of root cell names in the database
- [`QisMFile`] `Get_cell_children_list` -- Get a list of cell names referenced directly by the given cell
- [`QisMFile`] `Get_cell_extents` -- Get the extents of the given cell (lower-left, upper-right)
- [`QisMFile`] `Get_layer_list` -- Get a list of layers and datatypes in the database
- [`QisMFile`] `Grid` -- Get the size of 1 grid expressed in the file units (um, mm, inch etc.) e.g for a um file, Grid() == 0.001 implies nm resolution
- [`QisMFile`] `Units` -- Get the size of 1 grid expressed in meters e.g for a um file with Grid() == 0.001, Units() == 1e-9 (nm)

- [`QisMFile`] `Create_exploder` -- Create one instance of the spatial query object (exploder). Requires 1 x (11027) license per call

---

QisMExploder

> API to run spatial queries and collect data vectors (boundary, path, text, reference) crossing a view of the QisMFile database (cell, layers, window, nesting level)

Header : qismview.h, qismexploder.h

Binary : qismlib64.dll / libqis64.so

License : 1 x (11027) per exploder object

- check-out -- `QisMFile::Create_exploder`
- check-in -- `QisMFile::Destroy_exploder`

Key features

- Multiple spatial query objects (exploders) can co-exist and work per database and for multiple databases to make best use of the available computing power (threads)
- Rich set of controls (view cell, layers, nesting level, window, texts, paths, arrays etc.) to control query results
- Spatial queries can be used to process both geometric and hierarchical information from the design
- Send the results to QisMBool , QisMCorrX for further processing or QisMRaster , QisMFileOut for writing files on disk

Key classes & operations ([class] operation -- description)

- [`QisMView`] `Set_view_cell` -- Set the view cell (name)
- [`QisMView`] `Set_layers_on`, `Set_layers_off` -- Show/hide one or more layers
- [`QisMView`] `Set_text_on` -- Show/hide TEXT items
- [`QisMView`] `Set_exact_window` -- Set a rectangular view window
- [`QisMView`] `Set_nesting_level` -- Show ALL or a specific nesting level
- [`QisMExploder`] `Set_convert_paths_to_boundaries` -- Convert paths to boundaries
- [`QisMExploder`] `Get_vector_data` -- Collect data vectors from the set view (run a spatial query) as a series of callbacks (1 per vector)
- [`QisMExploderV2`] `Get_boundaries` -- Get only polygons from the set view in form of a container
- [`QisMExploderV2`] `Get_cell_references` -- Get only reference vectors to a given cell from the set view
- [`QisMExploderV2`] `Get_cell_tree` -- Get the entire sub-tree of a cell in form of reference vectors
- [`QisMNotify`] `On_qismt_vector` -- Notification/callback to receive a data vector

---

QisMBool

> API to perform operations (union, intersection, xor, difference, clipping, sizing etc.) on large sets of polygons

Header : qismbool.h

Binary : qismlib64.dll / libqis64.so

License : 1 x (11047) per boolean object

- check-out -- `QisMBool::Create_instance`
- check-in -- `QisMBool::Destroy_instance`

Key features

- Optimized and time tested for processing very large data sets making use of the available computing power (threads) whenever appropriate
- Refined to handle and repair complex polygons and generate clean geometries in the output
- Get polygons with holes as leonov (represented as a set of distinct outer and inner polygons), cutlines (co-incidental edges connecting outer and inner boundaries) or butting (broken up into distinct boundaries with co-in-siding edges)
- Control the max. number of vertices in output polygons
- Other output options such as clipping, sizing etc.
- Send the results to QisMCorrX for further processing or QisMRaster , QisMFileOut for writing files on disk
- Learn about Artwork's boolean engine
- Learn more about QisMBool

Key classes & operations ([class] operation -- description)

- [`QisMBool`] `Create_instance` -- Create an instance of the boolean object
- [`QisMBool`] `New_settings` -- Create an instance of the settings object
- [`QisMBoolInst`] `BooleanST` -- Perform binary operations between two sets of polygons using one thread
- [`QisMBoolInst`] `UnionMT` -- Unionize a set of polygons using multiple threads
- [`QisMBoolInst`] `BinaryMT` -- Perform binary operations between two sets of polygons using multiple threads
- [`QisMBoolX`] `Find_connected_sets` -- Group a set of polygons into multiple sets based on connectivity (touching or intersection)
- [`QisMBoolSettings`] `Set_clip_window` -- Set a rectangular clipping window
- [`QisMBoolSettings`] `Set_max_points` -- Set the max. number of points for a polygon in the output (break the ones larger then this)

QisMClipper

> API clip a set of paths/boundaries against a set of rectangular or polygonal windows

Header : qismclipper.h

Binary : qismlib64.dll / libqis64.so

License : 1 x (11047) per clipper object

- check-out -- `QisMClipper::Create_poly_instance`, `Create_box_instance`
- check-in -- `QisMClipper::Destroy_instance`

Key features

- Use rectangular or polygonal areas as clipping windows
- Optimized for clipping large sets of polygons with ability to employ multiple threads for clipping
- Ability to clip PATHs

- Send the results to QisMBool , QisMCorrX for further processing or QisMRaster , QisMFileOut for writing files on disk

Key classes & operations ([class] operation -- description)

- [`QisMClipper`] `Create_poly_instance` -- Create a clipping object for polygonal clipping windows
- [`QisMClipper`] `Create_box_instance` -- Create a clipping object for rectangular clipping windows
- [`QisMClipperObj`] `Clip_boundaries` -- Clip a set of boundaries using multiple threads
- [`QisMClipperObj`] `Clip_path` -- Clip one path (output is one or more boundaries)
- [`QisMClipperObj`] `Boundary_interaction` -- Test the interaction of a boundary with the specified windows
- [`QisMClipperObj`] `Path_interaction` -- Test the interaction of a path with the specified windows
- [`QisMClipperObj`] `Point_interaction` -- Test the interaction of a point with the specified windows

QisMExplCounter

> API to get exploded (flat) vertex/polygon counts for a given set of layers of a cell of a QisMFile database

Header : qismexplcounter.h

Binary : qismlib64.dll / libqis64.so

License : -

Key features

- Knowing the size of the data set upfront can be very useful in balancing the load correctly amongst multiple worker threads in an application
- Use brute force method using multiple threads or a smart hierarchical approach to compute counts
- Compute exploded counts for any cell, set of layers, window or nesting level

Key classes & operations ([class] operation -- description)

- [`QisMExplCounter`] `Get_counts_hierarchical` -- Compute counts using a hierarchy traversal algorithm for fast results
- [`QisMExplCounter`] `Get_counts_full` -- Compute counts using multiple threads with the option for specifying a window
- [`QisMExplCounterV2`] `Get_counts_nl` -- Get_counts_full with control for nesting level

QisMLog

> API for thread-safe logging from both inside and outside the QisM system. Requires logging to be enabled/setup during QisMLib_initialize_once

Header : qismlog.h

Binary : qismlib64.dll / libqis64.so

License : -

Key features

- Ability to centralize the logging for all components connected to the QisMLib system
- Ability to specify multiple targets for logging -- standard out (console), file, callback etc.
- Multi-thread safe
- Variety of options for the application to control logging -- create a new file, append to an existing file, specify a `FILE*`, receive callbacks etc.
- Use the environment variable `ACS_VERBOSE=ON` or `ACS_VERBOSE=OFF` to enable/disable detailed logging

Key classes & operations ([class] operation -- description)

- [`QisMLog`] `Log_msg` -- Send a message to all log targets
- [`QisMLog`] `Screen_msg` -- Send a message to stdout/stderr only
- [`QisMLog`] `Verbose_msg` -- Send a message depending on the state of the `ACS_VERBOSE` environment variable
- [`QisMLogCb`] `On_qismlog_msg` -- Callback/notification when a log message have been transmitted anywhere in the QisMLib system

---

QisMFileOut

> API for writing polygons to a GDSII/OASIS/TIFF/BMP/RAW file

Header : qismfileout.h

Binary : qismlib64.dll / libqis64.so

- May require QisMRaster to be installed for certain operations

License : 1 x (14827) per call required only for writing image files

- check-out -- `QisMFileOut::Open_image_writer`
- check-in -- `QisMFileOut::Close_writer`

Key features

- Single interface to generate output files in multiple formats, vector (GDSII/OASIS) or raster (TIFF/BMP/RAW)
- Multi-thread safe -- write boundaries to the same file simultaneously from multiple threads

Key classes & operations ([class] operation -- description)

- [`QisMFileOut`] `Open_vector_writer` -- Create a GDSII/OASIS writer
- [`QisMFileOut`] `Open_image_writer` -- Create a TIFF/BMP/RAW writer (requires the QisMRaster extension)
- [`QisMBndryWriter`] `Boundary`, `Boundaries` -- Write one or more boundaries

QisMLayerSynth

> API for synthesizing new layers of polygons based on an expression involving operations (union, or, intersection, xor, difference) between existing layers of a QisMFile database

Header : qismlayersynth.h

Binary : qismlayersynth64.dll / qismlayersynth64.so (plug-in/extension to qismlib64.dll/libqism64.so)

License : 1 x (11069) per synthesizer object

- check-out : `QisMLayerSynth::New_synthesizer`
- check-in : `QisMLayerSynth::Delete_synthesizer`

Key features

- A variety of boolean operators -- union `+` , intersection `&` , difference `-` , xor `^` , or/aggregation (without union) `|` , assignment `=`
- Use an linear notation for simple expressions e.g `"0:0=1-2:2&3"`
  - aggregate all datatypes of `1`
  - subtract `2:2` from it
  - compute intersection of the results with all datatypes of `3`
- Use a postfix notation (preceded by `@`) for complex expressions e.g `"@1 2:2 - 3 4:4 ^ & 0:0 ~"`
  - aggregate all datatypes of `1`
  - subtract `2:2` from it and save the result -- `A`
  - aggregate all datatypes of `3`
  - xor it with `4:4` and save the result -- `B`
  - compute intersection between `A` and `B`
  - transmit the result as `0:0`
  - i.e `0:0 = (1-2:2) & (3^4:4)`
- Multi-threaded computations for faster results
- Send the results to QisMBool , QisMCorrX for additional processing or QisMRaster for generating images, QisMFileOut for writing files on disk
- Learn more about QisMLayerSynth

Key classes & operations ([class] operation -- description)

- [`QisMLayerSynth`] `New_synthesizer` -- Create new instance of the synthesizer object
- [`QisMLayerSynthObject`] `Synthesize_layers` -- Execute layer synthesis based on a layer expression
- [`LSynthNotify`] Synthesized_polygon -- Notification/callback to receive a synthesized polygon

---

QisMClipExtract

> API for extracting lots of tiny clips of polygons in parallel threads as GDSII/OASIS/TIFF/BMP/RAW files on disk or a collection/raster image in memory from a QisMFile database

Header : qismclipextract.h

Binary : qismclipextract64.dll / qismclipextract64.so (plug-in/extension to qismlib64.dll/libqism64.so)

- May require QisMLayerSynth to be installed for certain operations

License : N x (31209) where N = no. clips to extract in parallel (no. extraction threads)

- check-out & check-in : `QisMClipExtract::Extract_image` , `QisMClipExtract::Extract_polygons`

Key features

- N x M threading architecture where N = no. extraction threads processing one clip at a time and M = no. threads per clip for processing the clip data (for union or layer synthesis)
- Variety of options for image output -- dithering, polarity inversion, right-to-left rasterization
- Variety of options for polygonal output -- per-layer union, clipping, max. vertices per polygon, butting with overlap or cutlines option for polygons with holes
- Perform layer synthesis using QisMLayerSynth within each clip
- Send the results of polygonal extraction to QisMBool or QisMCorrX for additional processing
- Learn more about QisMClipExtract

Key classes & operations ([class] operation -- description)

- [`QisMClipExtract`] Extract_image -- Extract the specified set of windows (clips) as raster images
- [`QisMClipExtract`] `Extract_polygons` -- Extract the specified set of windows (clips) as polygons
- [`QisMClipExtractV2`] `Extract_synthesized_polygons` -- Extract the specified set of windows (clips) as polygons synthesized from a layer expression (requires the QisMLayerSynth extension and license)
- [`QisMClipExtractNotify`] `On_clipextract_image` -- Notification/callback to receive a clip as image
- [`QisMClipExtractNotify`] `On_clipextract_polygons` -- Notification/callback to receive a clip as a set of polygons

---

QisMRaster

> API to generate high resolution monochrome raster images (1 bit/pixel) from a QisMFile database

Header : qismraster.h

Binary : qismraster64.dll / qismraster64.so (plug-in/extension to qismlib64.dll/libqism64.so)

- May require QisMLayerSynth to be installed for certain operations

License : 1 x (14827) per rasterizer object

- check-out : `QisMRaster::Create_rasterizer` , `QisMRasterV2::Open_file_writer`
- check-in : `QisMRaster::Destroy_rasterizer` , `QisMRasterV2::Close_file_writer`

Key features

- Employ multiple threads to quickly rasterize one large bitmap
- Employ multiple rasterizer objects in parallel to quickly rasterize multiple smaller bitmaps
- Use a N x M threading architecture where N = no. of rasterizer threads and M = no. threads per rasterizer to achieve optimal performance
- Use cellular pattern recognition for much faster results (especially for arrays)
- Use non-uniform resolution along X and Y (rectangular pixels)
- Variety of options for image outputs -- dithering, polarity inversion, right-to-left rasterization, solid fill or outline only etc.
- Generate images from a set of polygons created by the application (not existing in the database) or result of layer synthesis using QisMLayerSynth
- API to format a given image buffer to disk as TIFF/BMP/RAW files
- Create a new image or superimpose over a previous one
- Overlay a set of polygons on a raster image in paint, scratch or dither mode
- Create a TIFF/BMP/RAW image writer for writing one boundary at a time

- Learn more about QisMRaster

Key classes & operations ([class] operation -- description)

- [`QisMRaster`] `Create_rasterizer` -- Create an instance of the rasterizer object
- [`QisMRaster`] `Create_formatter` -- Create an instance of the formatter object
- [`QisMRasterV2`] `Open_file_writer` -- Open a TIFF/BMP/RAW
- [`QisMRasterWriter`] `Boundary` -- Write a polygon to the TIFF/BMP/RAW file
- [`QisMFormatter`] `Write_tiff` , `Write_bmp` , `Write_raw` -- Write the contents of an image buffer to a TIFF/BMP/RAW file
- [`QisMRasterizer`] `Rasterize_window` -- Generate a raster image (buffer) from a view (cell, layers, window) of the database
- [`QisMRasterizer`] `Rasterize_window_synthesized` -- Generate a raster image (buffer) from the result of a layer expression over a window of the database (requires the QisMLayerSynth extension and license)
- [`QisMRasterizer`] `Rasterize_polygon_set` -- Generate a raster image (buffer) from a set of polygons
- [`QisMRasterizerV2`] `Overlay_polygon_set` -- Overlay a set of polygons on an existing raster image (buffer)

QisMNtrc

> API to trace nets of connected METAL (conductors) and VIA (dielectric) polygons based on a pre-defined stackup for a given QisMFile database

Header : qismntrc.h, qismstackupdb.h

Binary : qismntrc64.dll / qismntrc64.so (plug-in/extension to qismlib64.dll/libqism64.so)

- also requires QisMLayerSynth to be installed (license **not** required)

License : 1 x (11059) per tracer object

- check-out - `QisMNtrc::Open_trace`
- check-out - `QisMNtrc::Close_trace`

Key features

- Compute nets from a single point on a METAL layer, or multiple nets crossing a specific region of the layout (hot-spot)
- API to define and query a stackup with controls to associate a stackup layer with the results of boolean operations between select database layers (e.g METAL1 = 1- 2:2)
- Multiple net-tracers can be active simultaneously and can be used in parallel for managing large number of extractions
- Send the results to [QisMFileOut][#QisMFileOut] to for generating files on disk
- Learn more about Net tracing

Key classes & operations ([class] operation -- description)

- [`QisMNtrc`] `Open_trace` -- Create an instance of a net tracer object
- [`QisMNtrc`] `New_stackup` -- Create a new stackup definition
- [`QisMNtrcTracer`] `Point_trace` -- Trace a net from a single point on a METAL layer
- [`QisMNtrcTracer`] `Region_trace` -- Trace one or more nets that cross a region of the layout (hot spot)
- [`QisMNtrcNotify`] `Begin_net` , `Net_boundary` , `End_net` -- Notifications/callbacks to receive a net

QisMBoolFiles

> API to perform boolean operations (union, or, difference, intersection, xor) over a set of windows between two QisMFile databases

Header : qismboolfiles.h

Binary : qismboolfilesextn64.dll / qismboolfiles64.so (plug-in/extension to qismlib64.dll/libqism64.so)

License : N x (11071) where N = no. windows to be processed in parallel (no. window threads)

- check-out & check-in - `QisMBoolFiles::Booleanize_two_files`

Key features

- Perform boolean operations between a cell, set of layers of two databases over a set of windows
- N x M threading architecture for optimal performance where N = no. window threads (each processing one window at a time) and M = no. threads per window (for boolean operations)
- Use boolean operations -- or/aggregation, union, intersection, xor, difference; and variety of options such as clipping and max. vertices per polygon
- Use dynamic windowing for optimal load balancing (and therefore threading performance) in situations where certain regions of data are much more dense than others
- For each window, receive both the input polygon sets and the output
- Send the results to [QisMRaster][#QisMRaster] for generating raster images or QisMFileOut for writing files on disk
- Learn more about QisMBoolFiles

Key classes & operations ([class] operation -- description)

- [`QisMBoolFiles`] `Booleanize_two_files` -- Execute a boolean operation over a set of windows for the specified databases
- [`QisMBoolFilesClient`] `On_qismboolfiles_window_mt` -- Callback/notification to receive the results of boolean operations on a single window

QisMCorrX

> API to apply corrections (bilinear transformation) on a view of the QisMFile database based on one or more known correction points

Header : qismcorrx.h

Binary : qismcorrx64.dll / qismcorrx64.so (plug-in/extension to qismlib64.dll/libqism64.so)

License : 1 x (11093) per correction object

- check-out - `QisMCorrX::Create_correction_object`
- check-in - `QisMCorrX::Destroy_correction_object`

Key features

- Define a correction space (domains) based on any number of points located anywhere in the data space. Each correction point is defined by `x,y` (it's location in the source data) and `dx,dy` (the know correction at that point expressed as deltas)
- Get corrected polygons from a view of the database (cell, layers, window) or an arbitrary set of polygons generated by the application

- Multiple correction objects can be active simultaneously and can be used in parallel for optimal performance
- Employ multiple threads per correction for faster results
- Get the input/source polygons in addition to the corrected results
- Send the results to [QisMBool][#QisMBool] or QisMRaster for additional processing
- Convenience functions to compute standard transformations such as scaling, rotation, mirror, shear

Key classes & operations ([class] operation -- description)

- [`QisMCorrX`] `Create_correction_object` -- Create a new instance of the correction object
- [`QisMCorrXObj`] `Get_corrected_polys` -- Get corrected polygons from a view (layers, window) of the database
- [`QisMCorrXObj`] `Correct_polygons` -- Correct polygons in the specified set
- [`QisMCorrXNotify`] `On_corrected_polygons` -- Callback/notification to receive corrected polygons

---

QisMHExtract

> API to extract a view of the QisMFile database as a GDSII/OASIS file with hierarchy and clipping

Header : qismhextract.h

Binary : qismhextract64.dll / qismhextract64.so (plug-in/extension to qismlib64.dll/libqism64.so)

License : 1 x (11083) per extractor object

- check-out - `QisMHExtract::Create_extractor`
- check-in - `QisMHExtract::Destroy_extractor`

Key features

- Multiple extractor objects can be active simultaneously and can be used in parallel threads
- Rich set of clipping options -- clip polygons, references, drop partial polygons, drop partial references, clip paths etc.
- Use one or more rectangular, polygonal, circular areas and their complements (negatives) as a clipping mask
- Extract with or without hierarchy
- Generate GDSII or OASIS output on disk
- Filter references to specific cells based on names or regular expressions
- Filter tiny polygons
- Receive extracted data as vectors for further processing/filtering
- Apply basic transformations (scale, angle, mirror) to the output
- Add offsets to output layer and datatypes numbers and prefix/suffix to cell names
- Learn more about QisMHExtract

Key classes & operations ([class] operation -- description)

- [`QisMHExtract`] `Create_extractor` -- Create an instance of the extractor object
- [`QisMHExtractor`] `Create_clipper`, `Create_box_clipper` , `Create_poly_clipper` , `Create_circ_clipper` -- Create a clipping object containing one or more rectangular, polygonal or circular clipping areas
- [`QisMHExtractor`] `Create_file_writer` -- Open a GDSII/OASIS file as a target for extracted cell definitions

- [`QisMHExtractor`] `Extract` -- Extract a view of the database
- [`QisMHExtractTarget`] `On_extract_begin_cell` , `On_extract_boundary` , `On_extract_path` , `On_extract_text` , `On_extract_sref` , `On_extract_aref` , `On_extract_end_cell` -- Callbacks/notifications to receive the definition for each extracted cell

---

QisMRTCR (GDSII only, WINDOWS only)

> API to generate high resolution monochrome raster images from a view of a QisMFile database with corrections and annotations
>
> **To be used in conjunction with SFGen**

Header : qismrtcr.h

Binary : qismrtcr64.dll (plug-in/extension to qismlib64.dll)

- Also requires QisMRaster to be installed

License :

- 1 x (1303) per RTCR job
  - check-out - `QisMRTCR::Setup_job`
  - check-in - `QisMRTCR::End_job`
- 1 x (14827) per rasterizer object
  - check-out - `QisMRTCRJob::Create_rasterizer`
  - check-in - `QisMRTCRJob::Destroy_rasterizer`

Key features

- Create a new RTCR job by applying corrections and annotations to a source GDSII file
- Multiple RTCR jobs can be active simultaneously and can be used in parallel threads
- Get access to the QisMFile database associated with a job for further info/processing
- Create one or more rasterizers per job for generating multiple images in parallel
- Employ multiple threads per raster image for faster results
- Use this API in conjunction with QisMRaster for more image processing and formatting options
- Additional image processing operations such as polarity inversion, dithering, right-to-left rasterization, pixel-shifting and polygon masks
- Use non-uniform resolution along X and Y (rectangular pixels)
- Learn more about RTCR

Key classes & operations ([class] operation -- description)

- [`QisMRTCR`] `Setup_job` -- Create a new RTCR job (source GDSII + corrections + annotations)
- [`QisMRTCRJob`] `Create_rasterizer` -- Create an instance of the rasterizer object
- [`QisMRTCRJob`] `Get_raster_image` -- Generate a monochrome raster image from the associated job using the associated rasterizer

---

QisMScript (`beta`)

> API to add scripting support to any API/feature/operation within the QisM system. The script engine can be invoked within any QisM client application or using a ready-to-use console application -- **qismscript64.exe**

- **Header** : qismscript.h
- **Binary** : qismlib64.dll / libqism64.so
- **License** : -

Key features

- Run scripts using an available console program (qismscript64.exe) or add scripting support to your application using the QisMScript API
- An application and register it's own commands
- At the moment, script commands only available from QisMRTCR. In time, all APIs in the QisMLib system will support scripting
- Support for script commands to define and use variables that contains strings or object handles
- The scripting system is only loaded if explicitly specified during QisMLib_initialize_once
- The scripting system use QisMLog for a coherent information reporting scheme for the user

Key classes & operations ([class] operation -- description)

- [`QisMScriptRunner`] `Run_script_command`, `Run_script_command_v` -- Execute an available script command with the specified arguments
- [`QisMScriptRunner`] `Run_script_command_file` -- Execute a text file containing script commands
- [`QisMScriptRunner`] `Get_script_help` -- Get a complete list of available commands with usage as a string
- [`QisMScriptRunner`] `Add_var`, `Get_var` -- (From the client application,) define/query a script variable as a **name=string_value** pair that can be referenced by any subsequent script command
- [`QisMScriptRegister`] `Register_command` -- Register a command and command handler so that it is available to be executed by the script engine
- [`QisMScript`] `On_command` -- Callback/notification to the registered script handler to execute the a command associated with it
- [`QisMScriptUtil`] `Add_var` , `Get_var` , `Remove` -- (While executing a command,) define/query/delete a script variable as a **type,name=object_handle** pair that can be referenced by any subsequent script command
- [`QisMScriptUtil`] `Log_msg` -- (While executing a command,) create a log message to be reported back to the user via the QisMLog system

---