```
#ifndef __ARTWORK_CONVERSION_SOFTWARE_COPYRIGHT_QISMNTRC_H
#define __ARTWORK_CONVERSION_SOFTWARE_COPYRIGHT_QISMNTRC_H
#ifdef __cplusplus

#include "qismextension.h"

#ifndef QISMNTRC_LNX_DECLSPEC
  #ifdef _WIN32
    #define QISMNTRC_LNX_DECLSPEC
  #elif __GNUC__ >= 4
    #ifdef QISMNTRC_EXPORTS
      #define QISMNTRC_LNX_DECLSPEC __attribute__ ((visibility ("default")))
    #else
      #define QISMNTRC_LNX_DECLSPEC
    #endif
  #else
    #define QISMNTRC_LNX_DECLSPEC
  #endif
#endif

namespace NsQisMLib {
class QisMFile; /* qismfile.h */
}

namespace NsQisMNtrc {

/*****************************************************************************
QisMNtrc Error Codes
-----------------------------------------------------------------------------*/
struct QISMNTRC_LNX_DECLSPEC QisMNtrcError
{
  /***************************************************************************
  Error codes for QisMNtrc::Open_trace(..)
  -------------------------------------------------------------------------*/
  enum Open_trace_codes {
    OTE_INV_FILEDB=-1 /* Invalid (null) file db handle */
   ,OTE_INV_STACKUP=-2 /* Invalid (null) stackup db handle */
   ,OTE_NO_LSYNTH=-3 /* Failed to locate the layer synthesizer extension */
   ,OTE_NO_BOOLX=-4 /* Failed to locate the boolean (X) extension */
   ,OTE_SETUP=-5 /* Failed to get layer expression for order# <order>, <why> */
   ,OTE_LICENSE=-6 /* Failed to acquire QisMNtrc license, <why> */
  };
  /***************************************************************************
  Error codes for QisMNtrcTracer::Point_trace(..) or
  QisMNtrcTracer::Region_trace(..)
  -------------------------------------------------------------------------*/
  enum Point_Region_trace_codes {
    PRTE_INV_ARGS=-1 /* Invalid trace parameters, <why> */
   ,PRTE_SETUP=-2 /* Failed to setup trace, <why> */
   ,PRTE_TRACE=-3 /* Failed to trace net, <why> */
   ,PRTE_INTR=-4 /* Execution interrupted by client [code <code>] */
  };
};
/*****************************************************************************/

/*****************************************************************************
Notification (Callback) API for a client requesting Point, Region tracing
-----------------------------------------------------------------------------*/
class QISMNTRC_LNX_DECLSPEC QisMNtrcNotify
{
public:
  /***************************************************************************
  Notification at the start of receiving polygons for a net
  -------------------------------------------------------------------------*/
  virtual int Begin_net(const int netID) { return 0; }
  /*-------------------------------------------------------------------------
  PARAMETERS
  + netID: A number (>0) that uniquely identifies a net within an instance of
    QisMNtrcTracer
  -------------------------------------------------------------------------
  RETURN
  + The return code from this notification is ignored
  -------------------------------------------------------------------------
  DETAILS
  + At the moment, this notification originates from the same thread from which
    the tracing was invoked
  *****************************************************************************/

```

```
  /*****************************************************************************
  Notification to receive a single polygon for a net
  -----------------------------------------------------------------------------*/
  virtual int Net_boundary(
    const int* xy, const int numVert, const unsigned short order,
    const double grid
    ) { return 0; }
  /*-----------------------------------------------------------------------------
  PARAMETERS
  + xy: Polygon co-ordinates (in database-units)
  + numVert: Number of integer pairs in 'xy' (xy is an array of numVert vertices
    or (numVert*2) integers)
  + order: Stackup order (>= 1) to which the polygon belongs
  + grid: Resolution of the data (same as QisMFile::Grid()). To convert a
    co-ordinate from database-unit (int) to file-unit (double), multiply by
    the grid
  -----------------------------------------------------------------------------
  RETURN
  + The return code from this notification is ignored
  -----------------------------------------------------------------------------
  DETAILS
  + The array of xy-coordinates obtained via this notification is only valid
    during this notification. The client should not store this address/pointer
    for later use. Instead, the client should copy this array to another buffer
    during this notification
  + The polygon in question may have been synthesized from one or more layers
    and therefore may not correspond to a specific layer:datatype in the
    CAD file
  + At the moment, this notification originates from the same thread from which
    the tracing was invoked
  *****************************************************************************/

  /*****************************************************************************
  Notification at the end of receiving polygons for a net
  -----------------------------------------------------------------------------*/
  virtual void End_net(const int netID, const bool truncated) { }
  /*-----------------------------------------------------------------------------
  PARAMETERS
  + netID: A number (>0) that uniquely identifies a net within an instance of
    QisMNtrcTracer
  + truncated: If true, the net in question was truncated (not completely traced
    beacuse of client-specified tracing options)
  -----------------------------------------------------------------------------
  DETAILS
  + If TraceOptions::Ignore_truncated_nets(..) option is ON, the client will
    not receive any truncated nets and therefore 'truncated' will only be false
  + At the moment, this notification originates from the same thread from which
    the tracing was invoked
  *****************************************************************************/

  /*****************************************************************************
  Reserved for future use
  -----------------------------------------------------------------------------*/
  virtual void Net_marker(
    const char* txt, const int x, const int y, const unsigned short o
    ) { }
  /*****************************************************************************/

  /*****************************************************************************
  Progress/Information regarding a particular trace
  -----------------------------------------------------------------------------*/
  virtual void Net_info(const char* pre, const char* msg, const char* post) { }
  /*-----------------------------------------------------------------------------
  PARAMETERS
  + pre, post: Strings to be used as prefix, suffux to msg for the purpose of
    formatting
  + msg: String containing information
  -----------------------------------------------------------------------------
  DETAILS
  + Can be used to report progress updates while the tracing is in progress
  *****************************************************************************/

  /*****************************************************************************
  A periodic refresh while the tracing is in progress so that the client can
  process essential GUI events etc.
  -----------------------------------------------------------------------------*/
  virtual int Net_refresh_mt() { return 0; }
  /*-----------------------------------------------------------------------------
```

```
  DETAILS
  + This notification can originate from one of the many trace threads. The
    client may also receive more that one of these notifications at the same
    time from different threads. The client must apply appropriate protections
    to guarantee mutual exlusion to whichever resources are accessed during
    this notification
  ******************************************************************************/
};
/******************************************************************************/

/******************************************************************************
A base class for various QisMNtrc data storage objects
------------------------------------------------------------------------------*/
class QISMNTRC_LNX_DECLSPEC QisMNtrcObject
{
public:
  /****************************************************************************
  Get the implentation name and version
  ------------------------------------------------------------------------------*/
  virtual const char* Object_name() const = 0;
  /****************************************************************************/

  /****************************************************************************
  Destroy any derivative of QisMNtrcObject using the 'delete' operator
  ------------------------------------------------------------------------------*/
  virtual ~QisMNtrcObject() {}
  /****************************************************************************/
};
/******************************************************************************/

/******************************************************************************
A data structure to store Point-tracing parameters

Create an instance using QisMNtrc::New_object("PointTraceArgs") and destroy
using the delete operator
------------------------------------------------------------------------------*/
class QISMNTRC_LNX_DECLSPEC PointTraceArgs: public QisMNtrcObject
{
public:
  /****************************************************************************
  Set/Get the view cell (to trace nets on)
  ------------------------------------------------------------------------------*/
  virtual void View_cell(const char* cellname) = 0;
  virtual const char* View_cell() const = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + cellname: Name of a valid cell
  ******************************************************************************/

  /****************************************************************************
  Set/Get the layer (metal) of the seed point from where tracing begins
  ------------------------------------------------------------------------------*/
  virtual void Seed_layer(
    const unsigned short layer, const unsigned short dttp
    ) = 0;
  virtual unsigned short Seed_layer() const = 0;
  virtual unsigned short Seed_datatype() const = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + layer, dttp: Layer and datatype for the seed point
  ----------------------------------------------------------------------------
  DETAILS
  + The seed layer:dttp MUST corresponds to/participates in a METAL order in the
    stackup
  ******************************************************************************/

  /****************************************************************************
  Set/Get the co-ordinates of the point where the tracing begins
  ------------------------------------------------------------------------------*/
  virtual void Seed_Point(const double x, const double y) = 0;
  virtual double Seed_point_x() const = 0;
  virtual double Seed_point_y() const = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + x, y: xy co-ordinates in file units
  ----------------------------------------------------------------------------
  DETAILS
  + The seed point MUST lie on a polygons that corresponds to/participates in
```

```
235     a METAL order in the stackup
236   ********************************************************************************/
237 };
238 /********************************************************************************/
239
240 /********************************************************************************
241 A data structure to store Region-tracing (hot-spot tracing) parameters
242
243 Create an instance using QisMNtrc::New_object("RegionTraceArgs") and destroy
244 using the delete operator
245 --------------------------------------------------------------------------------*/
246 class QISMNTRC_LNX_DECLSPEC RegionTraceArgs: public QisMNtrcObject
247 {
248 public:
249   /********************************************************************************
250   Set/Get the view cell (to trace nets on)
251   --------------------------------------------------------------------------------*/
252   virtual void View_cell(const char* cellname) = 0;
253   virtual const char* View_cell() const = 0;
254   /*-------------------------------------------------------------------------------
255   PARAMETERS
256   + cellname: Name of a valid cell
257   ********************************************************************************/
258
259   /********************************************************************************
260   Set/Get the list of layer and datatype numbers (metal) to be used to find
261   the seed points for tracing within the specified regions (hot-spots)
262   --------------------------------------------------------------------------------*/
263   virtual void Seed_layers(
264     const int nLayers, const unsigned short* layerList,
265     const unsigned short* dttpList
266     ) = 0;
267   virtual int Seed_layer_count() const = 0;
268   virtual const unsigned short* Seed_layer_numbers() const = 0;
269   virtual const unsigned short* Seed_dttp_numbers() const = 0;
270   /*-------------------------------------------------------------------------------
271   PARAMETERS
272   + nLayers: Number of items (layers, dttps) in the list
273   + layerList, dttpList: List of layer/dttp numbers
274   -------------------------------------------------------------------------------
275   DETAILS
276   + The seed layer:dttp(s) MUST corresponds to/participates in a METAL order in
277     the stackup
278   + Atleast one such layer:dttp MUST be specified to do region tracing
279   ********************************************************************************/
280
281   /********************************************************************************
282   Clear the list of regions (hot-spots)
283   --------------------------------------------------------------------------------*/
284   virtual void Clear_regions() = 0;
285   /********************************************************************************/
286
287   /********************************************************************************
288   Add a region (hot-spot) to the list
289   --------------------------------------------------------------------------------*/
290   virtual void Add_region_llur(
291     const double minX, const double minY, const double maxX, const double maxY
292     ) = 0;
293   virtual void Add_region_llwh(
294     const double minX, const double minY, const double width, const double height
295     ) = 0;
296   virtual void Add_region_cwh(
297     const double cx, const double cy, const double width, const double height
298     ) = 0;
299   /*-------------------------------------------------------------------------------
300   PARAMETERS
301   + minX, minY: Co-ordinates of the lower left point of the rectangular region
302     in file units
303   + maxX, maxY: Co-ordinates of the upper right point of the rectangular region
304     in file units
305   + width, height: Width and height of the rectangular region in file units
306   + cx, cy: Co-ordinates of the center-point of the rectangular region in file
307     units
308   + Atleast one region MUST be specified to do region tracing
309   ********************************************************************************/
310
311   /********************************************************************************
312   Get a region (hot-spot) from the list
```

```
  ----------------------------------------------------------------------------*/
  virtual int Region_count() const = 0;
  virtual bool Get_region(
    const int index, double& minX, double& minY, double& maxX, double& maxY
    ) const = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + minX, minY: Co-ordinates of the lower left point of the rectangular region
    in file units
  + maxX, maxY: Co-ordinates of the upper right point of the rectangular region
    in file units
  + index: Index of the region in the list. 0 <= index < Region_count()
  ****************************************************************************/
};

/*****************************************************************************
Data structure to store trace options that remain the same for a given view cell
+ stackup (and therefore QisMNtrcTracer). These options determine how the input
data set is computed and organized. The input data set, once computed, remains
persistant through the life of the QisMNtrcTracer object and can be used over
and over again for multiple traces

Create an instance using QisMNtrc::New_object("PreTraceOptions") and destroy
using the delete operator
----------------------------------------------------------------------------*/
class QISMNTRC_LNX_DECLSPEC PreTraceOptions: public QisMNtrcObject
{
public:
  /***************************************************************************
  Create a new copy
  ----------------------------------------------------------------------------*/
  virtual PreTraceOptions* Clone() const = 0;
  /***************************************************************************/

  /***************************************************************************
  Control the manner in which the data space is divided (tiling) for faster and
  parallel computation
  ----------------------------------------------------------------------------*/
  virtual void Optimal_tile(const int nx, const int ny) = 0;
  virtual int Optimal_tile_nx() const = 0;
  virtual int Optimal_tile_ny() const = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + nx, ny: Number of tiles along X,Y (>= 1)
  ----------------------------------------------------------------------------
  DETAILS
  + By default, QisMNtrcTracer auto-computes the number of tiles along X and Y
    based on the estimated number of vertices in the data space. This
    computation is only applied over the layers in the stackup and is
    multi-threaded. Auto-tiling will be a one-time computation for a given view
    cell and stackup
  + If the optimal tiling parameters are already known to the client, this
    computation can be avoided
  ***************************************************************************/

  /***************************************************************************
  Specify the maximum size of a rectangle to be treated as a via
  ----------------------------------------------------------------------------*/
  virtual void Max_via_size(const double maxDxDy) = 0;
  virtual double Max_via_size() const = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + maxDxDy: Max. Dx or Dy in file units for a rectangle to be treated as a via
  ----------------------------------------------------------------------------
  DETAILS
  + Any polygon on a VIA order that is not a simple rectangle and has a dx,dy
    larger than the specified limit will be dropped from the list of vias
  + By default (0.0), there is no limit to the size of what should be treated as
    a via
  + A via is determined to be connected to a neighboring metal polygon ONLY if
    it's centroid lies on/inside that metal polygon
  ***************************************************************************/

  /***************************************************************************
  Limit the range of order numbers involved in computation of nets
  ----------------------------------------------------------------------------*/
  virtual void Order_range(
    const unsigned short start, const unsigned short stop
```

```
391     ) = 0;
392   virtual unsigned short Start_order() const = 0;
393   virtual unsigned short Stop_order() const = 0;
394   /*-------------------------------------------------------------------------
395   PARAMETERS
396   + start, stop: Order numbers indicating the lower and upper bounds for tracing
397   -------------------------------------------------------------------------
398   DETAILS
399   + start, stop orders MUST correspond to METAL. 1 <= start <= stop <= 65535
400   + By default, start == 1, stop == 65535
401   *************************************************************************/
402 };
403 /*************************************************************************/
404
405 /*************************************************************************
406 Data structure to store trace options that can change between traces
407
408 Create an instance using QisMNtrc::New_object("TraceOptions") and destroy
409 using the delete operator
410 -------------------------------------------------------------------------*/
411 class QISMNTRC_LNX_DECLSPEC TraceOptions: public QisMNtrcObject
412 {
413 public:
414   /*************************************************************************
415   Set/Get number of threads to be used for tracing
416   -------------------------------------------------------------------------*/
417   virtual void Thread_num(const int thrnum) = 0;
418   virtual int Thread_num() const = 0;
419   /*-------------------------------------------------------------------------
420   PARAMETERS
421   + thrnum: No. of threads. If 0, QisMNtrcTracer auto-computes the number of
422     threads based on the number of processors available
423   *************************************************************************/
424
425   /*************************************************************************
426   Set/Get Maximum number of vertices after which tracing of a net is to be
427   truncated
428   -------------------------------------------------------------------------*/
429   virtual void Max_net_vertices(const long long numVert) = 0;
430   virtual long long Max_net_vertices() const = 0;
431   /*-------------------------------------------------------------------------
432   PARAMETERS
433   + numVert: Max. number of vertices
434   -------------------------------------------------------------------------
435   DETAILS
436   + If the Ignore_truncated_nets(..) option is set to true, truncated net will
437     not be returned to the user via QisMNtrcNotify. Otherwise, all nets will be
438     returned and the 'truncated' flag in QisMNtrcNotify::End_net(..) will
439     indicate if a net is truncated
440   *************************************************************************/
441
442   /*************************************************************************
443   Set/Get flag to ignore truncated nets
444   -------------------------------------------------------------------------*/
445   virtual void Ignore_truncated_nets(const bool yesNo) = 0;
446   virtual bool Ignore_truncated_nets() const = 0;
447   /*-------------------------------------------------------------------------
448   PARAMETERS
449   + yesNo: If true, truncated net will not be returned to the user via
450     QisMNtrcNotify. Otherwise, all nets will be returned and the 'truncated'
451     flag in QisMNtrcNotify::End_net(..) will indicate if a net is truncated
452   *************************************************************************/
453 };
454 /*************************************************************************/
455
456 /*************************************************************************
457 API to perform point and region (hot-spot) tracing
458 -------------------------------------------------------------------------*/
459 class QISMNTRC_LNX_DECLSPEC QisMNtrcTracer
460 {
461 public:
462   /*************************************************************************
463   Implementation name and version
464   -------------------------------------------------------------------------*/
465   virtual const char* Object_name() const = 0;
466   /*************************************************************************/
467
468   /*************************************************************************
```

```
  Get error message and code corresponding to the last error condition
  ----------------------------------------------------------------------------*/
  virtual const char* Get_last_error_msg() const = 0;
  virtual int Get_last_error_code() const = 0;
  /*****************************************************************************/

  /*****************************************************************************
  Trace a net from a single point on metal
  ----------------------------------------------------------------------------*/
  virtual bool Point_trace(
    const PointTraceArgs* args, QisMNtrcNotify* client,
    const TraceOptions* opts = 0
    ) = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + args: Point tracing parameters
  + client: Callback handler to receive net polygons and trace information
  + opts: Optional tracing parameters
  ------------------------------------------------------------------------------
  RETURN
  + Success: true
  + Failure: false. Call Get_last_error_msg() or Get_last_error_code() to get
    more information about the error
  ------------------------------------------------------------------------------
  ERROR CODES: One of the values from QisMNtrc::Point_Region_trace_codes
  ------------------------------------------------------------------------------
  DETAILS
  + Once the tracing begins, the client receives a series of
    QisMNtrcNotify::Net_info(..) and QisMNtrcNotify::Net_refresh_mt()
    notifications for progress updates and client refresh respectively
  + Once the tracing is complete, the client receives a
    QisMNtrcNotify::Begin_net(..) notification followed by a series of
    QisMNtrcNotify::Net_boundary(..) and finally a QisMNtrcNotify::End_net(..)
    notification
  + The entire data space is divided into equally sized tiles and input data
    for tracing is computed for a tile only once for a given cell when that
    tile is visited for the first time. This data persists for the life of
    QisMNtrcTracer unless the view cell changes and is reused for subsequent
    traces
  + Net polygons that cross multiple tiles are clipped at the tile boundary
    with a tiny amount of overlap
  + See PointTraceArgs and TraceOptions for more information about the
    trace parameters
  *****************************************************************************/

  /*****************************************************************************
  Trace nets that cross one or more regions (hot-spots)
  ----------------------------------------------------------------------------*/
  virtual bool Region_trace(
    const RegionTraceArgs* args, QisMNtrcNotify* client,
    const TraceOptions* opts = 0
    ) = 0;
  /*----------------------------------------------------------------------------
  PARAMETERS
  + args: Region tracing parameters
  + client: Callback handler to receive net polygons and trace information
  + opts: Optional tracing parameters
  ------------------------------------------------------------------------------
  RETURN
  + Success: true
  + Failure: false. Call Get_last_error_msg() or Get_last_error_code() to get
    more information about the error
  ------------------------------------------------------------------------------
  ERROR CODES: One of the values from QisMNtrc::Point_Region_trace_codes
  ------------------------------------------------------------------------------
  DETAILS
  + Once the tracing begins, the client receives a series of
    QisMNtrcNotify::Net_info(..) and QisMNtrcNotify::Net_refresh_mt()
    notifications for progress updates and client refresh respectively
  + For each net encountered, the client receives a
    QisMNtrcNotify::Begin_net(..) notification followed by a series of
    QisMNtrcNotify::Net_boundary(..) and finally a QisMNtrcNotify::End_net(..)
    notification
  + The entire data space is divided into equally sized tiles and input data
    for tracing is computed for a tile only once for a given cell when that
    tile is visited for the first time. This data persists for the life of
    QisMNtrcTracer unless the view cell changes and is reused for subsequent
    traces
```

```
  + Net polygons that cross multiple tiles are clipped at the tile boundary
    with a tiny amount of overlap
  + See RegionTraceArgs and TraceOptions for more information about the
    trace parameters
  ****************************************************************************/

  /****************************************************************************
  Get a list of regions that a specified net crosses
  ----------------------------------------------------------------------------*/
  virtual int Get_regions_crossing_net(
    const int netID, const int*& regionIDs
    ) = 0;
  /*--------------------------------------------------------------------------
  PARAMETERS
  + netID: Net identifier as obtained via QisMNtrcNotify::Begin_net(..)
  + regionIDs: A buffer to store the addeess of a list of region identifiers
  ----------------------------------------------------------------------------
  RETURN
  + The number of regions in the list (pointed to by regionIDs)
  ----------------------------------------------------------------------------
  DETAILS
  + A region-id identifies the location of a region in the list specified by
    the user (RegionTraceArgs). 1 <= region-id <= RegionTraceArgs::Region_count()
  + To retreive the co-ordinates of a region-id 'i', call
    RegionTraceArgs::Get_region(..) with index = i+1
  ****************************************************************************/

  /****************************************************************************
  Get a list of nets that cross a specified region
  ----------------------------------------------------------------------------*/
  virtual int Get_nets_crossing_region(
    const int regionID, const int*& netIDs
    ) = 0;
  /*--------------------------------------------------------------------------
  PARAMETERS
  + regionID: 1 <= region-id <= RegionTraceArgs::Region_count()
  + netIDs: A buffer to store the net identifiers
  ----------------------------------------------------------------------------
  RETURN
  + The number of nets in the list (pointed to by netIDs)
  ----------------------------------------------------------------------------
  DETAILS
  + A region-id identifies the location of a region in the list specified by
    the user (RegionTraceArgs). 1 <= region-id <= RegionTraceArgs::Region_count()
  + To retreive the co-ordinates of a region-id 'i', call
    RegionTraceArgs::Get_region(..) with index = i+1
  ****************************************************************************/
};
/****************************************************************************/

#define QISMEXTENSION_QISMNTRC "QisMNtrc" /* QisMNtrc Extension Name */
#define QISMCODE_QISMNTRC 11059 /* QisMNtrc Extension Product Number */

/****************************************************************************
The QisMLib Extension API that serves as a gateway to tracing nets
----------------------------------------------------------------------------*/
class QISMNTRC_LNX_DECLSPEC QisMNtrc: public NsQisMLib::QisMExtensionAPI
{
public:
  /****************************************************************************
  Get error message and code corresponding to the last error condition
  ----------------------------------------------------------------------------*/
  virtual const char* Get_last_error_msg() const = 0;
  virtual int Get_last_error_code() const = 0;
  /****************************************************************************/

  /****************************************************************************
  Create an instance of any one of the various QisMNtrc data structures
  ----------------------------------------------------------------------------*/
  virtual QisMNtrcObject* New_object(const char* className) = 0;
  /*--------------------------------------------------------------------------
  PARAMETERS
  + className: Name of the class whose instance is to be created
  ----------------------------------------------------------------------------
  RETURN
  + A handle to an instance of the newly created object. It must be
    down-casted to the appropriate class using dynamic_cast
  ----------------------------------------------------------------------------
```

```
  DETAILS
  + Use this method of creation for object of the following classes:
    * PointTraceArgs
    * RegionTraceArgs
    * PreTraceOptions
    * TraceOptions
  + To avoid memory leak, every object created using this method must be
    destroyed using the c++ delete operator
  ******************************************************************************/

  /******************************************************************************
  Create an instance of the net tracer
  ------------------------------------------------------------------------------*/
  virtual QisMNtrcTracer* Open_trace(
    NsQisMLib::QisMFile* fileDb, const char* stackup, const PreTraceOptions* opts,
    const int argC = 0, const char* const* argT = 0, void* const* argV = 0
    ) = 0;
  /*------------------------------------------------------------------------------
  PARAMETERS
  + fileDb: Handle to a QisMFile database obtained when a GDSII/OASIS/DbLoad
    file is loaded using QisMFile::Load_file
  + stackup: Path of the technology file that defines the stackup for tracing
    See 'Technology File Syntax' for details
  + opts: Tracing options that remain consistent for a given view cell and
    stackup
  + argC, argT, argV: Reserved for future use
  ------------------------------------------------------------------------------
  RETURN
  + Success: A non-NULL handle to a new tracer object
  ------------------------------------------------------------------------------
  ERROR CODES: One of the values from QisMNtrc::Open_trace_codes
  ------------------------------------------------------------------------------
  DETAILS
  + Each Open_trace requires 1 QisMNtrc license and that license is held until
    the tracer is destroyed using Close_trace
  ******************************************************************************/

  /******************************************************************************
  Destroy an instance of the net tracer
  ------------------------------------------------------------------------------*/
  virtual void Close_trace(QisMNtrcTracer* handle) = 0;
  /*------------------------------------------------------------------------------
  PARAMETERS
  + handle: Handle to a valid QisMNtrcTracer object to be destroyed
  ------------------------------------------------------------------------------
  DETAILS
  + Every tracer created using QisMNtrc::Open_trace MUST be destroyed using
    this method to avoid memory and license leak
  ******************************************************************************/
};

}
#endif
#endif
```