NextGenRaster API Reference

(c) Artwork Conversion Software Inc.
www.artwork.com
Document Version: 1.222

3/25/2016

Index

```
#ifndef _ARTWORK_CONVERSION_NEXTGENRASTER_H
#define _ARTWORK_CONVERSION_NEXTGENRASTER_H
#ifdef __cplusplus

#define _NEXTGENRASTER_EXPORT
#define _NEXTGENRASTER_LNX_EXPORT
#define _CBDECL
#if(defined(WIN32) || defined(WIN64))
  #undef _CBDECL
  #define _CBDECL __stdcall
  #if(defined(NEXTGENRASTER_EXPORTS))
    #undef _NEXTGENRASTER_EXPORT
    #define _NEXTGENRASTER_EXPORT __declspec( dllexport )
  #endif
#elif __GNUC__ >= 4
  #undef _NEXTGENRASTER_EXPORT
  #define _NEXTGENRASTER_EXPORT __attribute__ ((visibility ("default")))
  #undef _NEXTGENRASTER_LNX_EXPORT
  #define _NEXTGENRASTER_LNX_EXPORT __attribute__ ((visibility ("default")))
#endif

/* Header RCS $Revision: 1.222 $ $Date: 2016/03/25 23:13:21Z $ */

namespace NNextGenRaster {

/* A data structure to store the extents of a rectangular window using the
   co-ordinates of the lower-left point, width and height; in file units
 */
struct _NEXTGENRASTER_LNX_EXPORT sBOXLLWH
{
  double llx /* lower-left/min x */, lly /* lower-left/min y */, width, height;

  /* Constructors */
  sBOXLLWH(): llx(0.0), lly(0.0), width(0.0), height(0.0) {}
  sBOXLLWH(const sBOXLLWH& o_): llx(o_.llx), lly(o_.lly), width(o_.width),
    height(o_.height) {}

  /* Reset */
  void clear() { llx=lly=width=height; }
  /* Assignment */
  sBOXLLWH& assign(
    const double llx_, const double lly_, const double width_, const double height_
    ) { llx=llx_; lly=lly_; width=width_; height=height_;  return *this; }
};

/* A data structure to store the extents of a rectangular window using the
   co-ordinates of the center point, width and height; in file units
 */
struct _NEXTGENRASTER_LNX_EXPORT sBOXCTWH
{
  double cx /* center x */, cy /* center y */, width, height;

  /* Constructors */
  sBOXCTWH(): cx(0.0), cy(0.0), width(0.0), height(0.0) {}
  sBOXCTWH(const sBOXCTWH& o_): cx(o_.cx), cy(o_.cy), width(o_.width),
    height(o_.height) {}

  /* Reset */
  void clear() { cx=cy=width=height; }
  /* Assignment */
  sBOXCTWH& assign(
    const double cx_, const double cy_, const double width_, const double height_
    ) { cx=cx_; cy=cy_; width=width_; height=height_; return *this; }
};

/* A data structure to store the extents of a rectangular window using the
   co-ordinates of the lower-left and upper-right points; in file units
 */
struct _NEXTGENRASTER_LNX_EXPORT sBOXLLUR
{
  double llx /* lower-left/min x */, lly /* lower-left/min y */;
  double urx /* upper-right/max x */, ury /* upper-right/max y */;

  /* Constructors */
  sBOXLLUR(): llx(0.0), lly(0.0), urx(0.0), ury(0.0) {}
  sBOXLLUR(const sBOXLLUR& o_): llx(o_.llx), lly(o_.lly), urx(o_.urx), ury(o_.ury) {}

  /* Reset */
  void clear() { llx=lly=urx=ury; }
  /* Assignment */
  sBOXLLUR& assign(
    const double llx_, const double lly_, const double urx_, const double ury_
    ) { llx=llx_; lly=lly_; urx=urx_; ury=ury_; return *this; }
  sBOXLLUR& assign(const sBOXLLWH& o_) {
    llx=o_.llx; lly=o_.lly; urx=llx+o_.width; ury=lly+o_.height; return *this;
  }
  sBOXLLUR& assign(const sBOXCTWH& o_) {
    llx=o_.cx-(o_.width/2); lly=o_.cy-(o_.height/2);
    urx=llx+o_.width; ury=lly+o_.height; return *this;
  }
};

/* Opaque handle to the API that stores settings to control the opening of
   GDSII/OASIS/DbLoad-Cache files
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HFileOpenSettings
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

/* API to store settings to control the opening of
   GDSII/OASIS/DbLoad-Cache files
 */
class _NEXTGENRASTER_LNX_EXPORT IFileOpenSettings: public HFileOpenSettings
{
public:
  /* Specify either a list or a map of layers to be loaded. Any layer not
     mentioned implicitly or explicitly in this specification will be filtered
     and all the data belonging to that layer will be dropped from the
     database. If a map is specified, the layer, datatype values will be
     mapped to the new values before being added to the database and any
     knowledge of the original values will be lost
   *//* If not used, all layers and corresponding datatypes are loaded into the
     database *//* Syntax for a list of layers is:
     <layer_string> := <layer>[:<datatype>][,<layer>[:<datatype>]]*
     If datatype is omitted, all datatypes will be loaded for that layer
   *//* Syntax for a map of layers is:
     <layer_string> := "Off" | <map>[,<map>]*
     <map> := <layer_in>-<layer_out>
     <layer_in> := "All" | <layer>[:<datatype>]
     <layer_out> := "NULL" | <layer>[:<datatype>
     "All" indicates all layers and datatypes in the file
     "NULL" indicates that the specified <layer>[:<datatype>] is to be dropped
     If <layer_in> != <layer_out>, then <layer_in> is loaded as/mapped to <layer_out>
     If multiple <layer_in> are mapped to the same <layer_out>, they are
     effectively aggregated
   *//* Returns false if an error occured, true otherwise
     Use getLastErrorMsg/getLastErrorCode to get error info
   */
  virtual bool setLayersOfInterest(
    const char* csvLayerList_ /* Non-null, non-empty specification string */
    ) = 0;
  /* Control the tradeoff between memory usage and performance associated with
     loading of GDSII/OASIS/DbLoad-Cache files
     If true, keep the file and database in memory. This results in larger
     memory footprint and somewhat slower load but faster queries. Good option
     for file that are much smaller than the amount of available memory and need
     to be rasterized multiple times
     If false, the file remains on disk, the database is in memory. This has a
     smaller memory footprint (good for very large files) and relatively faster
     loading, but may result slightly slower queries
   *//* If not used, the file is NOT loaded to memory
   */
  virtual void setLoadToMemory(const bool yes_ = true) = 0;
  /* Filter cells by name. Syntax is:
     <list> := <cellname>[,<cellname>]*
   *//* Filtered cells will not appear in the database and all references to them
     will also be dropped
   *//* By default, all non-empty cells in the file are loaded into the database
   *//* Returns false if an error occured, true otherwise
     Use getLastErrorMsg/getLastErrorCode to get error info
   */
  virtual bool filterCellsByName(const char* csvListOfNames_) = 0;
  /* Filter cells by regular expression. Syntax is:
     <list> := <expression>[,<expression>]*
     <expression> is either a ms dos-like expression or a unix-like expression
     depending on the dosLike_ flag
     Any cell that matches any expression in the list will be filtered
   *//* Filtered cells will not appear in the database and all references to them
     will also be dropped
   *//* Returns false if an error occured, true otherwise
     Use getLastErrorMsg/getLastErrorCode to get error info
   *//* By default, all non-empty cells are loaded into the database
   */
  virtual bool filterCellsByRegularExpression(
    const bool dosLike_, const char* csvListOfExpressions_
    ) = 0;
  /* Reset settings to default values */
  virtual void reset() = 0;
};

/* Signature of a file open progress client */
typedef int (_CBDECL *FOnFileOpenProgress_t)(
  const char* message_, /* progress message */
  void* clientHandle_ /* client-specific handle passed on to each call */
  );

/* Opaque handle to the API that stores settings to control the file view
   once it has been loaded
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HFileViewSettings
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

/* API that stores settings to control the file view once it has been loaded
 */
class _NEXTGENRASTER_LNX_EXPORT IFileViewSettings: public HFileViewSettings
{
public:
  /* Set specific layers ON. Syntax:
     <layer_string> := "All" | <layer>[:<datatype>][,layer[:<datatype>]]*
     If <datatype> is omitted, all datatypes corresponding to <layer> are
     turned ON
   *//*
   Changes to this setting are cumulative. By default, all loaded layers are ON
   */
  virtual void setLayersOn(const char* layerString_) = 0;
  /* Set specific layers OFF. Syntax:
     <layer_string> := "All" | <layer>[:<datatype>][,layer[:<datatype>]]*
     If <datatype> is omitted, all datatypes corresponding to <layer> are
     turned OFF
   *//*
   Changes to this setting are cumulative
   */
  virtual void setLayersOff(const char* layerString_) = 0;
  /* Set the current view cell. By default, the view cell is the deepest top
     cell in the file
   */
  virtual void setViewCell(const char* cellName_) = 0;
  /* Reset settings to default values */
  virtual void reset() = 0;
};

/* Opaque handle to the API that stores the extents of one or more raster data
   windows
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HRasterWindowSet
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

/* Flags to specify the order in which the next raster tile is determined for
   row, column based tiling
 */
struct _NEXTGENRASTER_LNX_EXPORT ncTILEORDER
{
  enum Direction {
    UP=0, /* After the current tile, move along +Y */
    RIGHT=1, /* After the current tile, move along -X */
    DOWN=2, /* After the current tile, move along -Y */
    LEFT=3 /* After the current tile, move along +X */
  };
};

/* Flags to specify the location of the first tile for row, column based tiling
 */
struct _NEXTGENRASTER_LNX_EXPORT ncFIRSTTILE
{
  enum Position {
    LL=0, /* Start tiling from the lower-left */
    LR=1, /* Start tiling from the lower-right */
    UR=2, /* Start tiling from the upper-right */
    UL=3  /* Start tiling from the upper-left */
  };
};

/* API that stores the extents of one or more raster data windows
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterWindowSet: public HRasterWindowSet
{
public:
  /* Add a window by specifying the lower-left and upper-left points
   */
  virtual bool addWindow(const sBOXLLUR& extents_) = 0;
  /* Add rows and columns of same sized windows over a specific rectangular
     region
   */
  virtual bool addTilesNXY(
    const sBOXLLUR& window_, /* Region to be tiled */
    const int nX_, const int nY_, /* Column and Row counts */
    const ncFIRSTTILE::Position first_, /* Position of the first window */
    const ncTILEORDER::Direction next_ /* Direction to determine the next window */
    ) = 0;
  /* Add same sized windows over a specific rectangular region by specifying the
     window width and height
   */
  virtual bool addTilesWH(
    const sBOXLLUR& window_, /* Region to be tiled */
    const double width_, const double height_, /* in file units */
    const ncFIRSTTILE::Position first_, /* Position of the first window */
    const ncTILEORDER::Direction next_ /* Direction to determine the next window */
    ) = 0;
  /* Clear all windows */
  virtual void clear() = 0;
  /* Return the number of rectangular windows in the set */
  virtual int count() const = 0;
};

/* Flags indicating the CAD file format
 */
struct _NEXTGENRASTER_LNX_EXPORT ncCADFORMAT
{
  enum Type { UNKNOWN=0 /* No file OR error */, GDSII=1, OASIS=2, DBLOAD=3 };
};

/* Opaque handle to the API to get information about the file currently
   loaded into the database
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HFileInfo
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

/* API to get information about the file currently loaded into the database
 */
class _NEXTGENRASTER_LNX_EXPORT IFileInfo: public HFileInfo
{
public:
  /* Get the path of file on disk */
  virtual const char* filePath() = 0;
  /* Get list of cell names in the file */
  /* The function returns the number of cells in the list and listPtr_
     will point to an array of cell-name strings
   *//* Example:
     const char* const* list = NULL;
     int nCells = fileInfoHandle->listOfCells(&list);
   *//* The list of names should NOT be freed as it is allocated and managed
     internally
   */
  virtual int listOfCells(const char* const** listPtr_) = 0;
  /* Get list of top cell names in the file */
  /* The function returns the number of top cells in the list and listPtr_
     will point to an array of cell-name strings
   *//* Example:
     const char* const* list = NULL;
     int nCells = fileInfoHandle->listOfTopCells(&list);
   *//* The list of names should NOT be freed as it is allocated and managed
     internally
   */
  virtual int listOfTopCells(const char* const** listPtr_) = 0;
  /* Get list of layers in the file */
  /* The function returns the number of items in the layer/datatype list and
```

```
     layers_, datatypes_ will point to array of numbers. Therefore
     for 0 <= i < return value, layers_[i] and datatypes_[i] form a unique
     layer:datatype pair
   *//* Example:
     const unsigned short* layerList = NULL;
     const unsigned short* datatypeList = NULL;
     int nCells = fileInfoHandle->listOfLayers(&layerList, &datatypeList);
   *//* The list of layers/datatypes should NOT be freed as it is allocated and
     managed internally
   */
  virtual int listOfLayers(
    const unsigned short** layers_, const unsigned short** datatypes_
    ) = 0;
  /* Get the extents of a cell. Returns false if no such cell has been
     loaded
   */
  virtual bool getCellExtents(const char* cell_, sBOXLLUR& extents_) = 0;
  /* Returns the size of the grid in meters */
  /* e.g For a um file with a grid of 0.001 micron (nm), unitsInM() == 1e-9 */
  /* 1 file unit in meters = unitsInM()/gridInUnits() */
  virtual double unitsInM() = 0;
  /* Returns the ratio of the grid to 1 file unit (e.g micron) */
  /* e.g For a um file with a grid of 0.001 micron (nm), gridInUnits() == 0.001 */
  /* 1 file unit in meters = unitsInM()/gridInUnits() */
  virtual double gridInUnits() = 0;
  /* Get file format */
  virtual ncCADFORMAT::Type fileType() = 0;
};

/* Opaque handle to the API to get information about the current view
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HViewInfo
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

/* API to get information about the current view */
class _NEXTGENRASTER_LNX_EXPORT IViewInfo: public HViewInfo
{
public:
  /* Get the name of the current view cell */
  virtual const char* currentViewCell() = 0;
  /* Get list of layers that are currently ON (visible) */
  /* The function returns the number of items in the layer/datatype list and
     layers_, datatypes_ will point to array of numbers. Therefore
     for 0 <= i < return value, layers_[i] and datatypes_[i] form a unique
     layer:datatype pair
   *//* Example:
     const unsigned short* layerList = NULL;
     const unsigned short* datatypeList = NULL;
     int nCells = fileInfoHandle->layersOn(&layerList, &datatypeList);
   *//* The list of layers/datatypes should NOT be freed as it is allocated and
     managed internally
   */
  virtual int layersOn(
    const unsigned short** layers_, const unsigned short** datatypes_
    ) = 0;
  /* Get the extents of the current home view (extents of the view cell) */
  virtual void homeViewExtents(sBOXLLUR& extents_) = 0;
};

/* Opaque handle to the API to store rasterizer control settings
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HRasterSettings
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

#if(defined(WIN32) || defined(WIN64))
typedef __int64 VERTCNT_t;
typedef __int64 BSIZE_t;
#else
typedef long long VERTCNT_t;
typedef long long BSIZE_t;
#endif

/* Flags to specify the manner in which individual polygons are rasterized */
/* Image polarity (inversion) applies to all of these options */
struct _NEXTGENRASTER_LNX_EXPORT ncFILL
{
  enum Type {
    /* Boundaries are completely filled (subject to the dither value).
       Paths are converted to boundaries
     */
    SOLID=0,
    /* Only the edges of the boundaries are rasterized. Dithering is not
       applicable here. Paths are converted to boundaries
     */
    OUTLINE=1,
    /* This mode is typically used for file containing only path data.
       Boundaries if present are rasterized the same way as SOLID. For paths,
       only a line joining the vertices is drawn. Dithering is not
       applicable here
     */
    PATHLINE=2,
    /* This mode is typically used for file containing only path data.
       Boundaries if present are rasterized the same way as SOLID. For paths,
       only the vertex points are plotted as 1-pixel dots. Dithering is not
       applicable here
     */
    PATHPTS=3
  };
};

/* Flags to specify the raster image format */
struct _NEXTGENRASTER_LNX_EXPORT ncIMGFORMAT
{
  enum Type {
    /* Rasterize the data, but do not format it and/or write it to disk */
    NONE=0,
    /* Write the raster image to disk as a monochrome (1-bit-pixel) TIFF file
       with packbits compression
     */
    TIFF=1,
    /* Write the raster image to disk as a monochrome BMP file */
    BMP=2,
    /* Write the raster image as-is to a binary file with a small header *//*
       Header is defined as follows:
       First seven bytes identify the file type ('L''G''R''A''W''0''0')
       Four bytes for image width in pixels
       Four bytes for image height in pixels
       Eight bytes for the image data (as present in the raster buffer) in bytes
     */
    RAW=3,
    /* The raster buffer holds packbits-compressed raster image
       (very simillar to TIFF) and is written to disk as-is. *//*
       Format syntax:
       Format Tag - JDKPBFMT- 8Bytes
       Version Tag - v100 - 4Bytes
       Reserved - Anything - 256 Bytes
       Image Width - CntVal - 4 Bytes
       Image Height - CntVal - 4 Bytes
       Row Offset & Row Length - 8 & 4 Bytes
       Row Offset & Row Length - 8 & 4 Bytes
       ...
       Row Offset & Row Length - 8 & 4 Bytes
       Row Offset & Row Length - 8 & 4 Bytes
       Row Data 1
       Row Data 2
       ...
       Row Data N-1
       Row Data N
     */
    PBMEM=4
  };
};

/* Reserved for future use
 */
struct _NEXTGENRASTER_LNX_EXPORT ncTHREADMODE
{
  enum Type { N_THR_PER_WIN=0, N_WIN_PER_THR=1 };
};

/* API to store rasterizer control settings
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterSettings: public HRasterSettings
{
public:
  /* Set the image resolution by specifying the size of a pixel in file units
     (a.k.a DPU, dots per file unit)
     Default value is 1.0 file units. Resolution is identical in X and Y
     (square pixel)
   */
  virtual void setPixelSize(const double sizeInFileUnits_) = 0;
  /* Specify the number of threads to be used for rasterization.
     By default, this number is automatically determined based on the number of
     CPU cores
   */
  virtual void setNumberOfThreads(const int thrnum_) = 0;
  /* Specify image polarity:
     true: Draw black data over a white background (default)
     false: Draw white data over a black background
     This control impacts dithering as well as fill mode
   */
  virtual void invertPolarity(const bool whiteOnBlack_ = true) = 0;
  /* If enabled (default), the rasterizer detects repetitions using cell
     hierarchy to improve speed by reducing computations for repeating
     data
   */
  virtual void recognizePatterns(const bool yes_ = true) = 0;
  /* Specify a threshold (in terms of number of veritces) for polygon buffering
     to efficiently manage memory allocations and therefore boost both speed and
     memory usage performance
     Default is set to 1,000,000 veritces
   *//*
     If number of threads == 1 and vertexCount_ == 0, polygon buffering is
     disabled and each polygon is rasterized on the fly. This option is
     better used for small data windows where the client application wishes to
     manage it's own multi-threading by running multiple rasterizers on tiny
     windows in separate threads to rip multiple tiny windows in parallel
   */
  virtual void setMaxPolygonBufferCount(const VERTCNT_t vertexCount_) = 0;
  /* Specify if the raster image is to be written to disk in a specific file
     format. Default: Only rasterization, No formatting/writing to disk
   */
  virtual void setImageFormat(const ncIMGFORMAT::Type format_) = 0;
  /* Reserved for future use */
  virtual void setThreadingMode(const ncTHREADMODE::Type mode_) = 0;
  /* Specify the directory + file name (without extension) where the image will
     be written if a file format is specified. The extension will be added
     automatically based on the file format (.tif, .bmp, .raw, .pbmem)
   *//*
     In case of tiling, the row and column number will also appended to the
     base path
   */
  virtual void setOutputFilePathBase(const char* filePathBase_) = 0;
  /* Reset settings to default values */
  virtual void reset() = 0;
};

/* Extension of the raster settings API to enable raster image overlay
   and dithering
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterSettingsDither: public IRasterSettings
{
public:
  /* If enabled (true), current raster image drawn without clearing the
     raster buffer. Previous raster image will be preserved in areas in the
     current image where there is no data. Used for multi-layered rasterization.
     Default is false. The raster buffer is cleared before a new image is drawn
   */
  virtual void setOverlayMode(const bool enable_ = true) = 0;
  /* Specify a dither value that control the density of pixels in the shaded
     areas of the raster image using a 8x8 Bayer matrix. 1.0 implies 100% pixel
     density, 0.0 implies 0% pixel density. This setting is only used with
     SOLID fill. Default is 1.0 */
  virtual void setGreyLevel(const double dither_) = 0; /* 0.0 to 1.0 */
};

/* Extension of the raster settings API to control the manner in which
   individual polygons are rasterized (fill mode)
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterSettingsOutline: public IRasterSettingsDither
{
public:
  /* Specify the manner in which each polygon is rasterized. Default is SOLID
     fill. Refer to ncFILL for different modes
   */
  virtual void setOutlineMode(const ncFILL::Type mode_) = 0;
};

/* Extension of the raster settings API to control the direction in which row
   pixels are set (right-to-left or left-to-right)
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterSettingsReverseRow: public IRasterSettingsOutline
{
public:
  /* If true, raster row pixels are set into the raster buffer from right-to-left.
     The raster image will appear as though it has been mirrored about a Y-axis
     running through the center of the image when compared to the CAD data.
     If false (default), the pixels are set from left-to-right so that the image
     resembles the CAD data
   */
  virtual void setReverseRasterRow(const bool yesNo_) = 0;
};

/* Extension of the raster settings API to enable dual resolution (along X and
   Y) rasterization
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterSettingsXYres: public IRasterSettingsReverseRow
{
public:
  /* Set dual resolution by specifying the size of a pixel in file units
     (a.k.a DPU, dots per file unit) along X and Y
     Default value is 1.0 file units along X and Y.
   */
  virtual void setPixelSize(const double Xsize_, const double Ysize_) = 0;
```

```
};

/* Opaque handle to the API to retrieve information about the raster image
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HRasterInfo
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
};

/* API to retrieve information about the raster image
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterInfo: public HRasterInfo
{
public:
  /* Get the extents of the CAD data that was used to
     compose the raster image in file units
   */
  virtual void getDataExtents(
    double& llx_ /* lower-left/min X */, double& lly_, /* lower-left/min Y */
    double& urx_ /* upper-right/max X */, double& ury_ /* upper-right/max Y */
    ) const = 0;
  /* Get the extents of the CAD data that was used to
     compose the raster image when translated to pixel space. These are not
     the extents of the raster image itself and therefore may NOT start at
     pixel 0,0
   */
  virtual void getImageExtents(
    int& llx_ /* lower-left/min X */, int& lly_ /* lower-left/min Y */,
    int& urx_ /* upper-right/max X */, int& ury_ /* upper-right/max Y */
    ) const = 0;
  /* Get the image width in pixels */
  virtual int getImageWidth() const = 0;
  /* Get the image height in pixels */
  virtual int getImageHeight() const = 0;
  /* Get the image resolution in DPI (dots per inch) */
  virtual double getDPI() const = 0;
};

/* Opaque handle to the API to retrieve raster buffer used to store the
   raster image (1 bit per pixel)
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HRasterBuffer
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
};

/* API to retrieve raster buffer used to store the raster image (1 bit per pixel)
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterBuffer: public HRasterBuffer
{
public:
  /* Address of the memory block allocated to store the raster image
   */
  virtual const unsigned char* handle() const = 0;
  /* Size of the memory block in bytes
   */
  virtual BSIZE_t size() const = 0;
};

/* Opaque handle to the API to retrieve rasterization statistics
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
```

```
class _NEXTGENRASTER_LNX_EXPORT HRasterStats
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
};

/* API to retrieve rasterization statistics
 */
class _NEXTGENRASTER_LNX_EXPORT IRasterStats: public HRasterStats
{
public:
  /* Get total time (in seconds) to generate a raster image from a
     data window. This includes the time to fetch the data (explosion)
     and perform rasterization */
  virtual double totalTimeInSeconds() const = 0;
  /* Get the total time to perform rasterization (in seconds). This does NOT
     include the time to fetch the data */
  virtual double rasterTimeInSeconds() const = 0;
  /* Get the rasterized polygon count */
  virtual VERTCNT_t polygonCount() const = 0;
  /* Get the rasterized polygon vertex count */
  virtual VERTCNT_t vertexCount() const = 0;
};

/* Opaque handle to the API to write the image in the raster buffer in a
   specific image file format on disk
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HImgFormatter
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

/* API to write the image in the raster buffer in a specific image file format
   on disk
 */
class _NEXTGENRASTER_LNX_EXPORT IImgFormatter: public HImgFormatter
{
public:
  /* Create a monochrome (1 bit per pixel) TIFF file with packbits compression
     Returns false on error. Use getLastErrorMsg(), getLastErrorCode() for
     error info
     */
  virtual bool writeTIFF(
    const char* filePath_, /* Full path of the file on disk */
    const int widthInPixels_, /* Image width in pixels */
    const BSIZE_t sizeInBytes_, /* Image size in bytes */
    const double dpi_, /* Image resolution in dots-per-inch */
    unsigned char* buffer_, /* Address of the raster image in memory */
    const int numThreads_ /* Number of threads to use for formatting */
    ) = 0;
  /* Create a monochrome (1 bit per pixel) uncompressed BMP file
     Returns false on error. Use getLastErrorMsg(), getLastErrorCode() for
     error info
   */
  virtual bool writeBMP(
    const char* filePath_, /* Full path of the file on disk */
    const int widthInPixels_, /* Image width in pixels */
    const BSIZE_t sizeInBytes_, /* Image size in bytes */
    const double dpi_, /* Image resolution in dots-per-inch */
    unsigned char* buffer_, /* Address of the raster image in memory */
    const int numThreads_ /* Number of threads to use for formatting */
    ) = 0;
  /* Create a raw image file
     Returns false on error. Use getLastErrorMsg(), getLastErrorCode() for
     error info
   */
  virtual bool writeRAW(
    const char* filePath_, /* Full path of the file on disk */
    const int widthInPixels_, /* Image width in pixels */
    const int heightInPixels_, /* Image height in pixels */
    const BSIZE_t sizeInBytes_, /* Image size in bytes */
    unsigned char* buffer_ /* Address of the raster image in memory */
    ) = 0;
};

/* API extension to do in-memory image formatting and compression in a format
   simillar to TIFF with packbits. A new buffer is allocated to hold the
   formatted image so that the original image is not modified
   Refer to ncIMGFORMAT::PBMEM for details
 */
class _NEXTGENRASTER_LNX_EXPORT IPbMemImgFormatter: public IImgFormatter
{
public:
  /* Format and compress the image in the raster buffer */
  virtual bool formatPBMEM(
    const int widthInPixels_, /* Image width in pixels */
    const BSIZE_t sizeInBytes_, /* Image size in bytes */
    const double dpi_, /* Image resolution in dots-per-inch */
    unsigned char* buffer_, /* Address of the raster image in memory */
    const int numThreads_, /* Number of threads to use for formatting */
    unsigned char** formattedBuffer_, /* Address of the new buffer holding
    the formatted image*/
    BSIZE_t* formattedImageSizeInBytes_ /* Size of the new buffer in bytes */
    ) = 0;
  /* Release the memory allocated to hold the new formatted raster image.
     This memory if not released will result in a leak and degraded
     application performance
     */
  virtual void releasePBMEM(
    unsigned char** formattedBuffer_ /* Address of the new buffer holding
    the formatted image*/
    ) = 0;
};

/* Signature of the callback every time a raster image is ready
 */
typedef int (_CBDECL *FOnRasterizeWindow_t)(
  HRasterInfo* info_, /* API to get image information */
  HRasterBuffer* buffer_, /* API to access the raster image data */
  HRasterStats* stats_, /* API to get rasterization statistics */
  void* clientHandle_, /* Opaque pointer supplied by the client application
  at the start of rasterization */
  const int windowNumber_, /* Location of window corresponding to this image
  in the raster window set */
  const int totalWindows_ /* Size of the raster window set */
  );

/* Opaque handle to the NextGenRaster API
   Use dynamic_cast to retrieve the underlying API and check if NULL is
   returned since all API extensions may not be available
 */
class _NEXTGENRASTER_LNX_EXPORT HNextGenRaster
{
public:
  /* Reserved for internal use */
  virtual void* core() = 0;
  /* Get error string with details about the last error condition */
  virtual const char* getLastErrorMsg() = 0;
  /* Get numeric code corresponding to the last error condition */
  virtual int getLastErrorCode() = 0;
};

/* NextGenRaster API */
```

```
class _NEXTGENRASTER_LNX_EXPORT INextGenRaster: public HNextGenRaster
{
public:
  /* Create an instance of the open file settings object to control
     how a GDSII/OASIS/DbLoad Cache file is loaded into the database.
     Must be destroyed when not needed
   */
  virtual HFileOpenSettings* createFileOpenSettings() = 0;
  /* Destroy an instance of the open file settings object */
  virtual void destroyFileOpenSettings(HFileOpenSettings* handle_) = 0;
  /* Load a GDSII/OASIS/DbLoad Cache file to database
     Returns false on error. Use getLastErrorMsg()/getLastErrorCode()
     to get error information
   */
  virtual bool openFile(
    const char* filePath_, /* Full path of the CAD file on disk */
    HFileOpenSettings* options_, /* Handle to the open file settings object */
    FOnFileOpenProgress_t progressCallback_, /* Optional callback to recieve
    file open progress updates */
    void* callbackClientHandle_ /* Optional client-specific opaque handle to be passed
    as-is to each call of the progess update callback */
    ) = 0;
    /* Get information about the file currently loaded into the database
     */
    virtual HFileInfo* fileInfo() = 0;
public:
  /* Create an instance of the view settings object to control the current
     view (cell, layers)
     Must be destroyed when not needed
   */
  virtual HFileViewSettings* createFileViewSettings() = 0;
  /* Destroy an instance of the view settings object */
  virtual void destroyFileViewSettings(HFileViewSettings* handle_) = 0;
  /* Set the current view (cell, layers).
     Returns false on error. Use getLastErrorMsg()/getLastErrorCode()
     to get error information
   */
  virtual bool setFileView(HFileViewSettings* handle_) = 0;
  /* Get information about the current view (cell, layers) */
  virtual HViewInfo* viewInfo() = 0;
public:
  /* Create an instance of the object to hold a set of raster windows
     Must be destroyed when not needed
   */
  virtual HRasterWindowSet* createRasterWindowSet() = 0;
  /* Destroy an instance of the raster window set object */
  virtual void destroyRasterWindowSet(HRasterWindowSet* handle_) = 0;
  /* Create an instance of the object to specify raster settings
     Must be destroyed when not needed */
  virtual HRasterSettings* createRasterSettings() = 0;
  /* Destroy an instance of the raster settings object */
  virtual void destroyRasterSettings(HRasterSettings* handle_) = 0;
  /* Rasterize a set of windows
     Returns false on error. Use getLastErrorMsg()/getLastErrorCode()
     to get error information
   */
  virtual bool rasterizeJob(
    HRasterWindowSet* window_, /* Handle to the raster window set */
    HRasterSettings* options_, /* Handle to the raster settings */
    FOnRasterizeWindow_t callback_, /* Optional callback handler to recieve
    and update every time a raster image is ready along with information
    about that image */
    void* clientHandle_ /* Optional client-specific opaque handle to be
    passed as-is to each call of the image ready callback */
    ) = 0;
public:
  /* Create an instance of the image formatter to format and write a raster
     image to disk
     Must be destroyed when not needed
   */
  virtual HImgFormatter* createFormatter() = 0;
  /* Destroy an instance of the image formatter */
  virtual void destroyFormatter(HImgFormatter* handle_) = 0;
};

/* API extension to specify a client-supplied file pointer for writing
   the execution log
 */
class _NEXTGENRASTER_LNX_EXPORT INextGenRasterLog: public INextGenRaster
{
public:
  /* Enable execution log by specifying a file pointer.
     Client application is responsible for opening and closing the file
     associated with this file pointer
     The file should be opened as a text file using the system fopen call
   */
  virtual void setLogFP(FILE* Fp_) = 0;
};

/* API extension to rasterize just one window in a buffer pre-allocated by
   the client program
 */
class _NEXTGENRASTER_LNX_EXPORT INextGenRasterWindow: public INextGenRasterLog
{
public:
  /* Rasterize a window in a pre-allocated buffer */
  virtual bool rasterizeWindow(
    const sBOXLLUR& dataExtents_, /* Extents of the data to be rasterized */
    unsigned char* buffer_, /* Address of a pre-allocated buffer to hold
    the raster image. The raster buffer must be big enought to hold the
    raster image */
    const size_t sizeBytes_, /* Size of the pre-allocated raster buffer in bytes */
    HRasterSettings* options_ /* Handle to the raster settings */,
    FOnRasterizeWindow_t callback_, /* Optional callback handler to recieve
    and update when the raster image is ready along with information
    about that image */
    void* clientHandle_ /* Optional client-specific opaque handle to be
    passed as-is to each call of the image ready callback */
    ) = 0;
};

}

extern "C" {

typedef void* HNextGenRaster_t; /* 'C' alias for HNextGenRaster* */

/* Get NextGenRaster library version string */
_NEXTGENRASTER_EXPORT const char* getNextGenRasterLibraryVersion();

/* Initialize the NextGenRaster library once at the start of the
   program
   Returns a handle (of type HNextGenRaster*) to the NextGenRaster API or
   NULL if an error occurs. In case of error, the errorMsgBuffer_ (if provided)
   contains the error string
 *//* argC_, argT_, and argV_ can be used to pass additional arguments during
   initialization. The size of argT_ and argV_ arrays must be no less than
   argC_ items. for i <= 0 < argC_, argT_[i] represents the argument type
   and argV_[i] is the corresponding value. In cases where argT_[i] contains
   sufficient information, argV_[i] can be NULL
 *//* Arguments currently supported: argT, argV type, argument description
 -log, const char*, path where an execution log will be created
 -log+, const char*, path to an existing log file to which the execution log will be appended
 -log*, FILE*, pointer to an open file for writing the execution log
 -stdout, NULL, print execution log to stdout/stderr

 All of these logging arguments will be overridden if INextGenRasterLog::setLogFP
 is used at any point during the life of the NextGenRaster library
 */
_NEXTGENRASTER_EXPORT HNextGenRaster_t initNextGenRasterLibraryOnce(
  const char* dllPath_, /* Full path of the NextGenRaster library */
  const int argC_, /* Number of initialization parameters */
  const char* const* argT_, /* List of argument type strings (argC_ count) */
  void** argV_, /* List of argument values (argC_ count) */
  char* errorMsgBuffer_, /* Buffer to recieve error message */
  const int errorMsgBufferLength_ /* Size of the error buffer in bytes */
  );

/* Close the NextGenRaster library once at the end of the program */
_NEXTGENRASTER_EXPORT void closeNextGenRasterLibraryOnce();

}
#endif
#endif
```