| |  | | --- | | |
| --- | --- | --- |
| |  | | --- | |  | |  | |  | |  | |  | | Version 1.5 Documentation:  | Overview | Programming |  | | --- | --- | --- | | * [Setup Notes](http://docs.google.com/setup.html) * [Notes/Caveats](http://docs.google.com/notes.html) * [FAQs](http://docs.google.com/faqs.html) | * [Basics](http://docs.google.com/basics.html) * [Objects](http://docs.google.com/objects.html) * [Functions](http://docs.google.com/functions.html) * [Constants](http://docs.google.com/constants.html) * [Source](http://docs.google.com/source.html) |  |    FunctionsGlobal Functions Some auxiliary functions for handling time  void Pause(const unsigned long ulTime);  Pause for iTime milliseconds  unsigned long CurrentTime();  Return the number of milliseconds since Windows was started  bool ElapsedTime(const unsigned long ulInterval);  Return true if iInterval milliseconds have elapsed since the last time ElapsedTime was called. Returns false on the first call, and the clock will overflow after too long of an interval. window window(const int iWindWidth, const int iWindHeight, const int iWindXPos, const int iWindYPos);  Create a new window! iWindWidth and iWindHeight will change the size of the window created. iWindXPos and iWindYPos is the position the window will appear on the screen  bool SetWaitClose(const bool bSetting) const;  If this window is the last window on screen, the default behavior is that in the window destructor the program waits for a mouse click before closing the window. This function will allow you to change this behavior. The default value is true. It will return whatever the previous state was.  bool SetBuffering(const bool bSetting);  Turn double buffering on or off. The way things currently work the background color of the double buffer defaults to black, whereas the normal screen defaults to white. Returns the old state.  void UpdateBuffer();  When double buffering is turned on this will copy the offscreen buffer to the screen.  void ChangeTitle(const char \*cpNewTitle);  void ChangeTitle(const std::string strNewTitle);  Allows the user to change what the title of the window. Accepts string or C-strings for input.  int GetWidth() const;  int GetHeight() const;  void GetWindowSize(int &iX, int &iY) const;  Functions to get information about the size of the window.  buttonstate GetButtonState(const button btMouse, int &iX, int &iY);  Get information on the current state of the mouse buttons and it's position  void GetMouseCoord(int &iX, int &iY);  Gets the current mouse coordinates, returns results in the passed parameters.  clicktype GetMouseClick(int &iX, int &iY);  Removes the next mouse click event from the queue iX and iY will contain the location of the click and the clicktype return willed allow the user to determine which mouse button was clicked, and if the click was a double click (see the clicktype enum in mousequeue.h).  clicktype WaitMouseClick(int &iX, int &iY);  WaitMouseClick works the same as above but will only return if there is a mouse click event in the queue, otherwise it will wait for one.  keytype GetKeyPress(char &cKey);  Removes the next keyboard event from the queue stuffs the key value into ucKey, and returns the type of key that was pressed (see the keytype enum in keyqueue.h).  keytype WaitKeyPress(char &cKey);  Same as above, but won't return unless there is a key event in the queue. As a result it will wait until there is one.  void FlushKeyQueue();  void FlushMouseQueue();  These two functions flush all waiting input out of the mouse and keyboard input queues. This is necessary because if someone accidentally hits a key or clicks the mouse that event will stay in the buffer until it is removed, which may be at a time that input would not be wanted by the user.  color SetBrush(const double dRed, const double dGreen, const double dBlue);  Sets the color of the current brush, which is used when drawing with the FILLED style. Returns the last selected color.  color SetBrush(const unsigned char ucRed, const unsigned char ucGreen, const unsigned char ucBlue);  Same as above, but allows you to use characters instead of doubles, which is faster. Returns the last selected color.  color SetBrush(const color &colBrush);  Set the brush color using a color object. Returns the last selected color.  color SetPen(const double dRed, const double dGreen, const double dBlue, const int iWidth = ciDefBrushSize);  Set the color and size (if you specify the size) of the current pen, which is used with drawing with the FRAME style and outlines entities drawing in the FILLED style. Returns the last selected color.  color SetPen(const unsigned char ucRed, const unsigned char ucGreen, const unsigned char ucBlue, const int iWidth = ciDefBrushSize);  Same as above, just allowing the use of characters instead of doubles for speed. Returns the last selected color.  color SetPen(const color &colPen, const int iWidth = ciDefBrushSize);  Set the pen color and size using a color object. Returns the last selected color.  void SetFont(const int iSize, const unsigned short usStyle, const fontfamily ffFamily, const char\* cpFontName = NULL);  Sets the current font. See the enums at the top of this file for info on the fontstyle and fontfamily types. C-String and String versions  void DrawPixel(const int iX, const int iY);  Draws a pixel at (iX, iY) in the current color  void DrawLine(const int iX1, const int iY1, const int iX2, const int iY2);  Draws a line from (iX1, iY1) to (iX2 - 1, iY2 - 1), because the Win32 API is wierd. This actually sort of makes sense, if you think about it, and I decided not to hardcode in +1's to make up for this fact.  void DrawRectangle(const int iX1, const int iY1, const int iX2, const int iY2, const drawstyle dsStyle = FILLED, const int iWidth = 0, const int iHeight = 0);  Draws a rectangle from with the same sort of off by one as DrawLine. Valid drawstyles are FRAME, FILLED, and INVERTED. Default drawing style is FILLED. If you want a rounded rectangle, use the last two parameters, iWidth and iHeight, to adjust the size of the fillets  void DrawTriangle(const int iX1, const int iY1, const int iX2, const int iY2, const int iX3, const int iY3, const drawstyle dsStyle = FILLED);  Draws a triangle with vertices (iX1, iY1), (iX2, iY2), and (iX3, iY3). Valid drawstyles are FRAME, FILLED, and INVERTED.  void DrawPolygon(const int\* ipX, const int\* ipY, const int iVertices, const drawstyle dsStyle = FILLED);  Works the same as DrawTriangle, but takes a pointer (or an array) of X and Y values. The number of elements in ipX and ipY must be the same, and iVertices must be no greater than the number of elements in the two arrays, or your program may very well crash. There is no way I can really guard against this problem.  void DrawCircle(const int iX, const int iY, const int iRadius, const drawstyle dsStyle = FILLED);  Draws a circle centered at (iX, iY) with a radius of iRadius. Valid drawstyles are FRAME, FILLED, and INVERTED.  void DrawEllipse(const int iX1, const int iY1, const int iX2, const int iY2, const drawstyle dsStyle = FILLED);  Draws an ellipse inside the rectangle bounded by (iX1, iY1) and (iX2, iY2). Valid drawstyles are FRAME, FILLED, and INVERTED.  void DrawArc(const int iX1, const int iY1, const int iX2, const int iY2, double dStartAngle, double dEndAngle, const drawstyle dsStyle = FRAME, const angletype atInfo = DEGREES);  Draws a section of and ellipse inside the rectangle bounded by (iX1, iY1) and (iX2, iY2), from dStartAngle to dEndAngle. atInfo may be set to RADIANS for those who perfer to use that system over DEGREES. Valid drawstyles are FRAME, FILLED, and INVERTED  void DrawBezierSpline(const int iX1, const int iY1, const int iX2, const int iY2, const int iX3, const int iY3, const int iX4, const int iY4);  Draws a Bezier spline. The start and end points are specified by (iX1, iY1) and (iX4, iY4). (iX2,iY2) and (iX3, iY3) specify control points that affect the shape of the curve. The only valid drawstyle currently is FRAME.  void DrawCatmullRomSpline(const int\* ipX, const int\* ipY, const int iVertices);  Draws a Catmull-Rom spline. Takes a pointer (or an array) of X and Y values for specifying the control points.  void DrawString(const int iX, const int iY, const char\* cpText);  void DrawString(const int iX, const int iY, const std::string strText);  Draws a string starting at the location given by (iX, iY) using the font set by SetFont(..).  void DrawInteger(const int iX, const int iY, const long lNumber);  void DrawDouble(const int iX, const int iY, const double dNumber);  Draws an integer or double starting at the location given by (iX, iY) using the font set by SetFont(..).  void GetStringSize(int &iWidth, int &iHeight, const char\* cpText);  void GetStringSize(int &iWidth, int &iHeight, const std::string strText);  Returns the width and height of a string drawn using the current font.  void GetIntegerSize(int &iWidth, int &iHeight, const long lNumber);  void GetDoubleSize(int &iWidth, int &iHeight, const double dNumber);  Returns the width and height of a number drawn using the current font .  void DrawImage(const image \*imgThis, const int iX, const int iY, const int iWidth = -1, const int iHeight = -1);  void DrawImage(const image &imgThis, const int iX, const int iY, const int iWidth = -1, const int iHeight = -1);  Draws an image at the location specified by (iX, iY). The iWidth and iHeight Parameters can be used to scale an image. The alpha channel in PNG images will be ignored if the image is scaled.  color GetColor(const int iX, const int iY);  void GetColor(const int iX, const int iY, double &dRed, double &dGreen, double &dBlue);  double GetRed(const int iX, const int iY);  double GetGreen(const int iX, const int iY);  double GetBlue(const int iX, const int iY);  Get the color of the pixel at location (iX, iY).  void StoreImage(image \*imgThis, const unsigned usX, const unsigned short usY, const unsigned short usWidth, const unsigned short usHeight);  void StoreImage(image &imgThis, const unsigned usX, const unsigned short usY, const unsigned short usWidth, const unsigned short usHeight);  Grabs a section of the section of the screen starting at (iX, iY) iWidth wide and iHeight tall and stores it in an image object . image image();  Make an null image object with no content.  image(const std::string strFileName, const imagetype itThisType);  image(const char \*cpFileName, const imagetype itThisType);  Creates an image and loads the image from the specified file. imagetype can be JPEG or PNG.  bool Valid() const;  Check to see if an image is valid.  void Open(const std::string strFileName, const imagetype itThisType);  void Open(const char \*cpFileName, const imagetype itThisType);  Open an image file. Again, imagetype can be JPEG or PNG.  void Save(const std::string strFileName, const imagetype itThisType) const;  void Save(const char \*cpFileName, const imagetype itThisType) const;  Save an image file.  unsigned short GetWidth() const;  unsigned short GetHeight() const;  Find out the dimensions of our image. color color(unsigned char Red = 0, unsigned char Green = 0, unsigned char Blue = 0);  unsigned char ucRed, ucGreen, ucBlue;  Color components. 0 = no intensity, 255 = full intensity. |