



Core.h

Collections

Any variable: container of any type for one element
.Create<type>(): creates the element of type
bool .Is<type>(): returns true if element of type
Array<type> variable
.Add(value): adds a value to the array
BiVector<type> variable: a bidirectional vector
.AddHead(value): adds at top position
.AddTail(value): adds a bottom position
.DropHead(): drops top value
Index<type> variable
.Add(value): adds a value to the index collection
int .Find(value): shows position of value
.FindAdd(value): if value not found add it
.Insert(int, value): inserts value at position int
bool .IsUnlinked(int): is value at position int tagged unfindable with unlink method
.Put(value): adds a value to the index
.Remove(int): removes value at position int
.Set(int, value): set value at position int
.Sweep(): remove all unlinked values from index
.Unlink(int): sets value at position int not findable
InArray<type> variable: fast insert&remove ops
InVector<type> variable: fast insert&remove ops
One<type> variable: same as std::unique_ptr
Operator <<: adds values to the collection
Sort(variable): sorts the collection type comparable
SortedArrayMap<type> variable: keeps sorted
SortedIndex<type> variable: keeps index sorted
SortedVectorMap<type> variable: keeps sorted
Tuple<type1, type2> variable: different types tuple
.MakeTuple(value, value): insert values variable
.Tie(value, value): extract individual values from a tuple variable
Vector<type> variable
.Add(value)
.Append({value, value}): adds a collection variable
clone(variable): variable copy to variable
.Get(int, value): gets the value at position int, if non existing returns value parameter
int .GetCount(): gets number of elements in a collection
.Insert(int, value): adds value at position int
variable pick(variable): variable move to variable
.Remove(int): removes value at position int
VectorMap<type, type> variable = { {value, value}, {value, value}}
.Add(value, value): adds values to the map
value .Find(value): finds a key
value .FindNext(value): finds the next key
value .Get(value): gets the value for a key
value .GetKeys(): shows the keys of the map
collection .GetValues(): shows all the values of the map
.SetKey(value, value): sets the key for a value

CombineHash

uint **CombineHash(variable, variable)**: returns a hashing of both variables

Command Line

CommandLine() variable: define a command line

variable
int .GetCount(): gets number of command line parameters
String Operator[int]: gets parameter int
SetExitCode(int): returns an exit code int

Comparables

CombineCompare(variable, variable)
(variable, variable)... : compares all variables combined, use with struct and Comparable template

```
struct foo: Comparable<foo>
{
    String a;
    int b;
    int c;
    int Compare(const foo& x) const {return
        CombineCompare(a, x.a) (b, x.b) (c, x.c);}
};
```

int SgnCompare(int, int): compares the sign, 0 if equal

Date and Time(rs)

Date variable
int .year(): gets the year part of the date
Date GetSysDate(): get current date
Time GetSysTime(): get current time
KillTimeCallback(int=0): remove callback on queue
SetTimeCallback(int, THISBACK(function), int=0): puts a callback on the timer queue with delay int ms (periodic if int is negative) with optional id **int**
Time variable
int .hour(): gets the hour part of the time

Dump to logfiles

DUMPC(variable): dumps collection type
DUMPHex(variable): hex dump to TheIDE logfile
DUMPM(variable): dumps map type
LOG(String): log to TheIDE logfile

Functions and Lambdas

[variable=variable] (type variable) {commands};
[] part gets variables from current thread
=: takes all variables by value
&: takes all variables by reference
&variable: gets a variable by reference
() part sets return values from the lambda
{} part contains code
Event<> name = [] {commands};: make an event
Function<type (type)> name = [] (type variable) {commands};: makes a function with type and name. All types must be the same.
name.Clear(): clears the function assignment
Gate<type> name = [] (type variable) {commands}; return boolean;: a gate always returns a boolean
Operator <<: assigns new lambda to function

Loops

For (type variable : collection) {}: loops through collection

Mailing POP3

#include <Core/POP3/POP3.h>
InetMessage variable: define a mail format string
String [int][“content-type”]: gets the content type from attachment indexed int
int [int].Decode().GetLength(): returns the length

of attachment indexed int
String [“date”]: returns the date field
String [“from”]: returns the sender field
String [“subject”]: returns the subject field
int .GetCount(): the number of attachments
bool .Read(messagestring): returns true if the getmessage string is a valid string
Pop3 variable: define a pop3 connection variable
String .GetLastError(): gets the last error
String .GetMessage(int): gets the indexed mail
int .GetMessageCount(): number of unread mails
.Host(String): defines the pop3 mail server
bool .Login(): returns true if successful
.Port(int): defines the pop3 tcp port
.SSL(): enable a SSL connection
.Trace(): enables pop3 logging
.User(String, String): username and password

Mailing SMTP

#include <Core/SMTP/SMTP.h>
SmtP variable: define a smtp connection variable
.Attach(String, String): attach a file named String with content **String**
.AttachFile(GetDataFile(String)): attach file String
.Auth(String, String): authenticates user String with password **String**
.Body(String): defines the body of the mail
String .GetError(): gets the error if the mail send method was not successful
.Host(String): set the smtp server
bool .Send(): returns true if mail successfully send
.SSL(): activates ssl for the connection
.Subject(String): set the mail subject
.To(String): sends mail to address

Multithreading

auto variable = **Async(lambda|function, value)**: executes function in parallel in current thread with value as parameter for this function or lambda
value variable.Get(): gives return value when ready
CoDo(lambda): parallel processing where the code does the scheduling

```
Vector<String> sdata;
for (int i=0; i<100; i++) sdata.Add(AsString(1.0/i));
double dsum=0;
std::atomic<int> ii=0; //atomic type for thread races
CoDo([&] {
    double m=0;
    for (int i=ii++; i<data.GetCount(); i=ii++)
        m += data[i];
    CoWork::FinLock();
    dsum += m; });
```

CoFindIndex(collection, value): parallel FindIndex
CoPartition(collection, lambda): parallel processing of collections using a subrange

```
int isum=0; Vector<int> vdata;
for (int i=0; i<10000; i++) vdata.Add(i);
CoPartition(vdata, [isum] (const auto& subrange) {
    int partial_sum=0;
    for (const auto& x : subrange) partial_sum += x;
    CoWork::FinLock(); //CoPartition inherits CoWork
    isum += partial_sum; });
```

ConditionVariable variable: control thread flow
.Signal(): signals variable, awakens thread
.Wait(mutex): wait for condition linked to mutex
CoSort(collection): parallel sort
CoWork variable: worker threads over all cores
CoWork variable & lambda|function: starts a new worker thread
.Cancel(): cancel all worker threads, running ones will execute until ended



.Finish(): wait for worker threads to finish
bool CoWork::IsCanceled(): check if all threads are cancelled

Mutex variable: defines a mutex

.Enter(): locks the mutex

.Leave(): unlocks the mutex

Mutex::Lock__(variable): lock until end of scope

Thread variable: defines a thread variable

.Run (lambda | function): starts an async thread

.Wait(): waits for thread to finish

Randomize function

int **Random(int):** gets a random int between 0 and int

Ranges and algorithms

collection **ConstRange(int, int):** returns a collection of int number of values int

int **Count(collection, value):** counts value presence
 int **FindIndex(collection, value):** gets the position of the value in the collection

int **FindMax(collection):** position of max value

int **FindMin(collection):** position of min value

collection **FilterRange(variable, lambda):** filters the collection using a lambda function

```
DUMP(FilterRange(x, [] (int x){return x>30;}));
```

collection **GetSortOrder(collection):** gets collection of int representing the order of values as sorted value

Max(collection): gets maximum value

value **Min(collection):** gets minimum value

collection **ReverseRange(collection):** reverse order

collection **SortedRange(collection):** sorts collection

collection **SubRange(collection, int, int):** trims collection from position int to int

value **Sum(collection):** summates all values

Sockets

HttpRequest variable: defines a http(s) request. If SSL needed add #include<Core/SSL/SSL.h>;

.Add(): create a new http request

.Do(): run the request, see inprogress

String **.GetContent():** read requested content

String **.GetErrorDesc():** gets error description

String **.GetPhaseName():** gets the phase name of the current request (when inprogress)

String **.GetReasonPhrase():** gets http reason phrase

String **.GetStatusCode():** gets the req status code

bool **.InProgress():** returns true if request busy

bool **.IsError():** returns true if request error

bool **.IsSuccess():** returns true if request ended ok

.Timeout(int): defines request timeout in ms. If int=0 then calls in asynchronous mode

.Url(String): defines the url of the request

.UserAgent(String): defines the user agent callsign

SocketWaitEvent variable: wait for sockets to be available to read from or to write to

.Add(socket): adds a socket (eg HttpRequest var)

.Wait(int): wait at most int ms (eg 10ms)

TcpSocket variable: defines a tcp socket variable

bool **.Accept(serversocketvariable):** accepts a connection from serversocket variable in a socket stack

```
TcpSocket server;
bool success=server.Listen(1234,5);
for(;;){
  TcpSocket s;
  if(s.Accept(server)){
    String ws.GetLine(); //gets command
```

```
s.Put("ack from:" + s.GetPeerAddr());
}}
bool .Connect(String,int): connects to host/ip address String on tcp port int
String .GetLine(): get answer from socket stack
String .GetPeerAddr(): returns the peer address
bool .Listen(int,int): returns true if server socket on port int is initialized with a listen queue of int
.Put(String): sends string data to the socket stack
```

Streams

CompareStream variable(variable): compares stream variable with variable

bool **.IsEqual():** check if streams are equal

.Put(object): adds object to the stream

FileAppend variable(String): appends to String file

.Close(): close the stream

FileIn variable(String): opens a file stream with filename String

.Close(): close the stream

String **.Get(long):** get long bytes from the stream

String **.GetLine():** gets the full line from the stream

byte **.Peek():** peeks at the byte at the pointer location

.Seek(long): puts the pointer at location long

FileOut variable(String): creates a file out stream with filename String

.Close(): close the stream

String **GetHomeDirFile(String):** returns the user home directory appended with file name string

stream **LoadFile(String):** loads entire file stream

Operator <<: adds objects to the stream

Operator %: serialization

```
StringStream ss3;
int x=123; Color h=White();
ss3 % x % h; // serialize the variables
StringStream ss4(ss3.GetResult());
int x2; Color h2;
ss4 % x2 % h2; // x2 and h2 are deserialized
```

OutFilterStream variable: output filter stream

SizeStream variable: stream to get the size

int **.GetSize():** gets the size in bytes of the stream

StringStream variable: creates a stream of Strings

stream **.GetResult():** Get the resulting stream

.Put32le(0x12345678): little endian stream store

.Put32be(0x12345678): big endian stream store

TeeStream variable(variable, variable): a stream that sends to both streams variable, variable

String

String **Format(String, values):** returns a String of a formatted String for the specific values

```
Format("%010d",value); //decimal 10 chars leading 0
Format("%c",value); //character value
Format(">10<d",value); //width 10 chars, left align
Format(">10<d",value); //width 10 chars, center align
Format("%d",value); //decimal value
Format("%i",value); //integer value
Format("%s",value); //string value
```

String variable

.Clear(): clears the value of the string

int **.Find(chars):** get the position of chars

int **.GetLength():** gets the length of the string

.Insert(int, chars): inserts chars at position int

.Mid(int, int): int chars from position int

.Remove(int, int): removes int chars at position int

int **.ReverseFind(chars):** get the position of chars in reverse order, from end to begin

bool **.StartsWith(chars):** does string starts with chars?

.ToString(): converts to wide string

.Trim(int): trims string to int chars

Operator <<: add string, number values

StringBuffer

StringBuffer variable: *char API call compatibility

.SetLength(int): defines buffer length

.StrLen(): adjust length to buffer values

strcpy(variable, variable): byte copy variable into variable

Values

type variable = **Null:** sets null value to variable

Value variable = value: self type defining variable

bool **.Is<type>:** returns true if type is corresponding

ValueArray variable: self type defining valuearray

.Add(value): adds a value to the array

.Insert(int, value): inserts value at position int

.Remove(int, int): removes int values from position int

.Set(int, value): sets value at position int

ValueMap variable: self type defining map

.Add(value, value): adds a key value with value

collection **.GetKeys():** get all key values

.Set(value, value): sets the key value to value

.SetKey(int, value): sets position int to key value

WString

WString variable: double byte string Unicode

.cat(int): adds a Unicode character at the end

.ToString(): converts to String

ZIP

#include <plugin/zip/zip.h>

FileUnZip variable(String): define a variable to unzip a file with filename string

int **.GetLength():** gets the length of the file

String **.GetPath():** returns the path

Time **.GetTime():** returns the time

bool **.IsError():** is there a unzip error?

bool **.IsEof():** is end of file reached?

bool **.IsFolder():** is the object a folder?

String **.ReadFile():** gets the content of the file and moves the pointer to the next file or folder

.SkipFile(): skips the current file and moves the pointer to the next file or folder

FileZip variable(String): define a zip variable with filename string

.BeginFile(String): opens file string to write to

.BeginFile(OutFilterStream,String): uses a output filter stream to write to file string

```
FileZip zip(GetHomeDirFile("test.zip"));
{
  OutFilterStream oz;
  zip.BeginFile(oz,"file2.txt");
  oz << "Some Content";
} //OutFilterStream destructor calls EndFile
```

.EndFile(): closes a beginfile file

bool **.Finish():** returns true if zip created successfully

.Put(String): puts string on a beginfile file

.WriteFile(String,String): writes the contents of string to filename string

.WriteFolder(String,time): makes a folder named string on time (see getsystime)

CtrlLib.h



ArrayCtrl

ArrayCtrl variable: creates a ArrayControl object
.Add(String, ...): adds a row of text to the control
.AddColumn(String): adds a column with a title
.Clear(): clears the control
int .Find(String): find line of String in control
.HeaderTab(int).SetText(String): Update list header with column int
.Remove(int): removes line int
.Set(line, column, String): sets a string at position
.WhenCursor()=lambda: when cursor enters ctrl
.WhenLeftDouble()=lambda: when left double click

Drawing

Draw& variable: gets a drawing context
.DrawArc(RectC(x1,y1,x2,y2),Point(x3,y3),Point(x4,y4),width,color): draws arc in rectangle from point 3 to point 4
.DrawDrawing(x1,y1,x2,y2,drawing): paint the drawing variable to any drawing context rectangle
.DrawEllipse(x1,y1,x2,y2,fillcolor,width,color): draws an ellipse in the rectangle
.DrawImage(x,y,image): paint the image bitmap to a position in a drawing context
.Drawimage(x1,y1,x2,y2,image,fillcolor): puts an image in a rectangle with color refill

```
w.DrawImage(10,10,100,100,CtrlImg::save(),Blue());
```


.DrawLine(x1,y1,x2,y2,width,color): draws a line
.DrawPolyLine(pointcollection,width,color): draws a polyline using a collection of Points
.DrawPolygon(pointcollection,fillcolor): polygon
.DrawRect(GetSize(), fillcolor): colored (enum Color) rectangle that fills the drawing context
.DrawRect(x1,x2,y1,y2,fillcolor): filled rectangle
.DrawText(x1,y1,y2,String,Font,color): places colored text rotated inside y2-y1

```
w.DrawText(10,10,20,"Test",Courier(100).Underline());
```


Drawing variable=drawingdrawvariable: set of vector drawing operations defined by DrawingDraw
DrawingDraw variable(x,y): vector drawing context with size x*y, to be used with Draw* commands
.Draw*(params): all draw methods available
Image variable=imagedrawvariable: set to an ImageDraw bitmap
ImageDraw variable(x,y): image bitmap context with size x*y, to be used with Draw& commands
.Draw*(params): all draw methods available
.Alpha().Drawcommand(params,GrayColor(byte)): draws a drawcommand with alpha layer Graycolor (255 = non-transparent)

External applications (clipboard, ...)

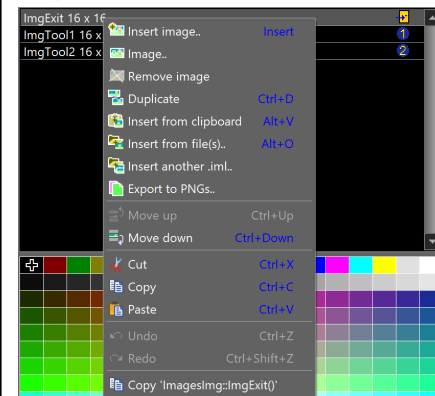
LaunchWebBrowser(String): launches the default web browser with the url string
WriteClipboardText(String): writes to clipboard

Fonts

Font variable(name, size): select a font object
int .GetAscent(): gets the distance from the baseline to the top of the font
int .GetDescent(): gets the distance from the baseline to the bottom of the font
int .GetHeight(): gets the height of the font

int Font::GetFaceCount(): gets the number of fonts present in the OS
String Font::GetFaceName(int): gets the font name
Operator[]: gets the individual letter width

Images



#define IMAGECLASS name: define the imageclass for future macros. The name is visible in the image editor context menu, last item (see screenshot)
#define IMAGEFILE <folder\name>: enter the folder and file name of the iml file
#include <Draw/iml.h>: enables use of iml files
Imagevariable=Imagebuffervariable: copies the buffered image to a visible image and clears buffer
image imageclass::Get(int): returns the image bitmap with index int
String imageclass::GetId(int): returns the image name with index int
int imageclass::GetCount(): returns number of images present in the imageclass definition
Image variable: defines a bitmap image
ImageBuffer variable(x,y): defines a bitmap image buffer of specified dimensions in pixels
Premultiply(imagebuffervariable): premultiplies the alpha channel with the rgb channels
Imagevariable=StreamRaster::LoadFileAny (~fileselvariable): loads a image from a FileSel standard dialog filename property

Images—cached

struct cachedclass:Imagemaker: Cached image class to be inherited from imagemaker
{type variable: defines the cached parameters
virtual String Key() const: key is a unique string
virtual Image Make() const: make is the image to be cached
String cachedclass::Key() const {commands; return String}: make a function that returns a unique string value from the cached parameter variables
Image cachedclass::Make() const {commands; return Image}: make a function that returns the image to be cached (cache size is limited by OS)
cachedclass variable: instances an cached object
.parameter=value: defines the cached parameters
MakeImage(variable): returns the cached image

JPEG Encoder

#include <plugin/jpg/jpg.h>

JPGEncoder variable(quality): define a jpg object with a compression quality between 0 and 100
.Create(size): creates a raster in memory

```
One<StreamRaster> raster=StreamRaster::OpenAny(file);
JPGEncoder jpg(20);
Jpg.Create(raster->GetSize());
```


.SetStream(fileout): define a fileout variable for the output encoded jpeg stream
.WriteLine(fileout): writes one line to the encoder

```
RasterLine l=raster->GetLine(1); //gets line 1
Buffer<RGBA> out(raster->GetWidth());
for(int j=0;j<raster->GetWidth();j++) {
    out[j].g=out[j].b=out[j].r=1[j].g
    out[j].a=255; }
jpg.WriteLine(out); //writes 1 line to RasterEncoder
```

Layouts

#define LAYOUTFILE <folder\name>: enter the folder and file name of the lay file
#include <CtrlCore/lay.h>
buttonvariable.Cancel() << Rejector(IDCANCEL): set return value for destructor of dialog window and add a default Cancel behavior
buttonvariable.Ok() << Acceptor(IDOK): defines a button with dialog OK handling

```
struct MyApp:public WithDlgLayout<TopWindow> {
    MyApp() {
        CtrlLayout(*this,"My Dialog");
        ok.Ok()<<Acceptor(IDOK); }
    GUI_APP_MAIN {
        MyApp app;
        switch(app.run()) {
            Case IDOK: PromptOK("OK pressed"); break;}
```


CtrlLayout(*this,String): sets up the dialog window titled String using the LAYOUTFILE specifications
WithDlgLayout<TopWindow> variable: defines a new model dialog layout based window
.layoutvariable.ClearModify(): set flag back to unmodified property
.layoutvariable.Disable(): disables editing value
.layoutvariable.GetData(): get the common display data for the specific control
Operator ~: same behavior as GetData method
bool .layoutvariable.IsModified(): returns true if the control has been modified
.layoutvariable.SetData(value): sets display data
Operator <<<: same behavior as SetData method
.layoutvariable.SetReadOnly(): makes the control read-only

Menus, bars and buttons

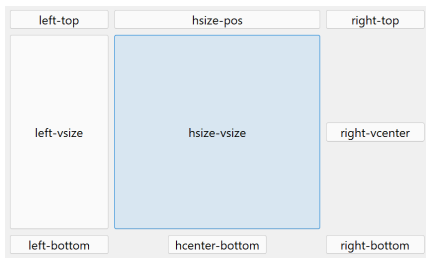
AddFrame(menu): add a top frame with a menu
AddFrame(InsetFrame()): add an inset frame
AddFrame(TopSeparatorFrame()): add a top separator frame
AddFrame(statusbar): adds a bottom statusbar
AddFrame(toolbar): add a top toolbar
Bar& variable: defines a menu bar item
.Add(String, lambda): adds a single menu item
.Add(image, lambda): adds a toolbar item
.Add(String, lambda).Help(String): help status bar
.Add(String, image, lambda): adds menu item with icon or toolbar item with tooltip, image = imageclass:object

```
bar.Add("Exit",ImagesImg::ImgExit(), [=] {Exit();});
```


.Separator(): inserts a separator horizontal line
.Sub(String,lambda): adds a menu heading

```
bar.Sub("File", [=] {bar.add("Exit", [=] {Exit();});});
```


Button variable: creates a click button
.SetLabel(String).horpos.verpos: adds button at a specific position, see hor-ver labels in screenshot



```
Button lb,button;
*this<<lb.SetLabel("OK").HSizePos(220,220).VSizePos(60,60);
*this<<button.SetLabel("Zoomed").LeftPos(Zx(10),Zy(64)).TopPosZ(10,34); //Enables Font-Zooming
```

MenuBar variable: creates a menu

.Set(lambda): executes menu bar design (subs)

```
MenuBar menu;
menu.Set( [=](Bar&bar){Mainmenu(bar);});
```

.WhenHelp=statusbar: help text link to statusbar

MenuBar::Execute(lambda): insert context menu, to be used in RightDown callback function

StatusBar variable: creates a statusbar at bottom

ToolBar variable: creates a toolbar at top

.Set(lambda): executes toolbar design (items)

.WhenHelp=status: help text link to statusbar

Offset and Clipping

Draw& variable

.Clip(x1,y1,x2,y2): clips the screen to rectangle

.ClipOff(x1,y1,x2,y2): combines clipping and offset

.End(): ends the offset or clipping state

.Offset(x,y): offsets the coordinate system

OpenGL drawing

```
#include <GLDraw/GLDraw.h>
```

```
#include <GLCtrl/GLCtrl.h>
```

struct gclass:GLCtrl: define an OpenGL class

{GLDraw variable: defines a GL draw object

.Draw*(params): all Drawing methods work

.Init(size): initializes the GL window

```
Size sz=GetSize();
GLDraw w;
w.Init(sz);
```

{virtual void GLPaint() {commands;}: define the paint event for an OpenGL class

```
GUI_APP_MAIN {
Ctrl::GlobalBackPaint(); //allow gl painting
TopWindow win;
gclass gl;
gl.SetFrame(InsetFrame());
win.Add(gl.HSizePos(10,10).VSizePos(10,10));
win.Open();
win.Run(); }
```

PDF

```
#include <PdfDraw/PdfDraw.h>
```

PdfDraw variable: defines a pdf drawing context

.Draw*(params): all drawing methods are valid

PdfSignatureInfo variable: adds digital signatures

.cert=LoadDataFile(String): certificate pem file

.contact_info=String: define contact information

.location=String: define a location for signing

.name=String: define a name for signing

.pkey=LoadDataFile(String): private key pem file

.reason=String: define a reason for signing

SaveFile(String,pdfvariable.Finish

(&pdfsignaturevariable)): save the pdf to a file

Printing

PrinterJob variable(String): defines a named job

bool .Execute(): returns true if printing executed

Draw& variable = variable.GetDraw(): gets a drawing context pointing to the printjob **variable**. All coordinates are based upon 1/600 of an inch

.EndPage(): ends a printer page

.StartPage(): starts a new printer page

Standard dialogs

bool EditText(variable,String,String): returns true if OK is pressed in an edit text field with title String and a query **String**, edit field comes in variable

FileSel variable: defines a file selection object

bool .ExecuteOpen(String): opens dialog with title to choose filename. Returns true if valid filename

.Type(String,String): define standard file types

```
String with help name String
fs.Type("Images","*.bmp;*.png;*.tif;*.jpg");
```

StreamRaster and Rasterline

RasterLine variable: defines a scanline object

```
StreamRaster raster=StreamRaster::OpenAny(filename);
RasterLine l=raster->GetLine(1); //Get raster line 1
```

Operator []: returns the RGBA pixel value of the operand in the rasterline variable

StreamRaster variable: defines a raster image stream object (bitmap with scanlines)

::OpenAny(filein): puts an image in the stream

int ->GetHeight(): returns the number of lines in the bitmap

rasterline ->GetLine(line): gets scanline line from the bitmap as a rasterline type

int ->GetSize(): returns the size of the bitmap

int ->GetWidth(): returns width of the bitmap

Tray icons

```
#define IMAGECLASS Tray
```

```
#define IMAGEFILE <folder/file.iml>
```

```
#include <Draw/iml.h>
```

struct trayclass:TrayIcon: new tray icon class

{virtual void LeftDown() {commands...}: commands to execute when clicking on the tray icon

{virtual void Menu(Bar& variable): adds a menu

{Add(String, THISBACK(method)): adds a menu item with text String to the tray icon menu

{Icon(imageclass::icon()): sets the tray icon

{Tip(String): sets the tray icon tip text

Types

Color(r,g,b): defines a color using RGB byte values

Point: defines a point with two coordinates

```
Vector<Point> p;
p << Point(10,10) << Point(20,20) << Point(30,30);
```

RGBA* variable: pointer to RGBA(lpha) values

.a: byte defining alpha (transparency) value

.b: byte defining blue value

.g: byte defining green value

.r: byte defining red value

```
ImageBuffer ib(50,50);
for(int y=0;y<50;y++){
  RGBA* line=ib[y];
  for(int x=0;x<50;x++){line->r=4*y; line++; }
```

Typedef appstruct CLASSNAME: needed for

callback function macros like THISFN

Windowed application (Ctrl)

Break(): exits application

Ctrl::Eventloop(): wait for all windows to be closed

Delete this: delete the current window

(new appstruct)->OpenMain(): open new window

ProcessEvents(): process GUI events

PromptOK(String): show information dialog

PromptOKCancel(String): returns true if OK clicked

Refresh(): refreshes the paint operation

TopWindow variable: defines a modal window

.Close(): closes a non-modal window

.FullScreen(): full screen top-mode

bool .IsOpen(): checks if a non-modal window is open

.KillCaret(): removes the cursor from the window

.Open(this): opens a non-modal window

.Run(): shows the window and execute events

.SetAlpha(byte): set the window transparency

.SetCaret(x,y,cx,cy): set blinking cursor at position x,y with width cx and height cy

.SetPos(x,y): sets the pointer position

.SetRect(x1,y1,x2,y2): set window size

.Sizeable(): the window is sizeable

.Title(String): the window title

.Zoomable(): the window is zoomable

```
struct MyApp:TopWindow {
virtual void Paint(Draws w) override {
w.DrawRect(GetSize(), White()); };
GUI_APP_MAIN
{ MyApp().Sizeable().Run(); }
```

Windowed event functions (Ctrl)

virtual void Activate() override: when the window is toplevel and is activated

void Close() override: when closing window

virtual bool Key(dword key,int count) override: returns true if ctrl accepted the keystroke with keyvalue key and repeat count

virtual void LeftDouble(Point pos, dword flags) override: when double click left mouse button

virtual void LeftDown(Point pos, dword flags) override: when left mouse button down, pos = position, flags = shift,ctrl,alt keyflags

virtual void MouseMove(Point pos, dword flags) override: when mouse moves over window

virtual void MouseWheel(Point pos,int delta,dword flags) override: when mouse wheel rotates, delta is the amount of rotation

virtual void Paint(Draw& w) override: when OS is painting on the window drawing context