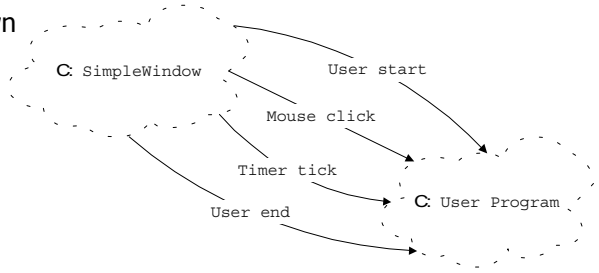
EzWindows API

A Graphical Application Programmer Interface

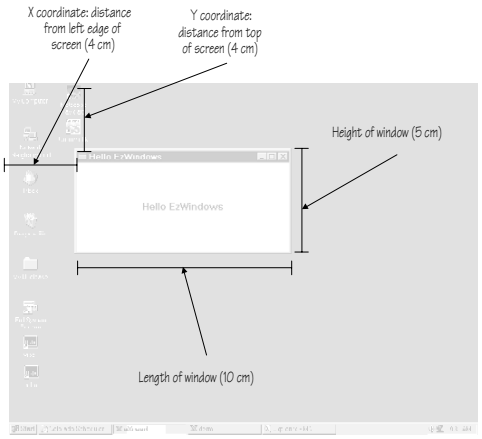
Event-based Programming

- Messages are sent to your program by the operating system
  - Mouse down
  - Mouse up
  - Key down
  - Key up
  - Refresh
  - Quit
  - Timer
- Handle messages by registering a call back



EzWindows Coordinate System

- Use centimeters
  - Metric
  - Simpler to understand than pixels
  - Device independent
  - Helps introduce notion of information hiding or encapsulation



Class Position

- For earlier objects, the position was specified by given both an x-coordinate and a y-coordinate
- We can now introduce a new object called Position and use it

Position

```
class Position {
  public:
      Position(float x = 0.0, float y = 0.0);
      float GetXDistance() const;
      float GetYDistance() const;
      Position Add(const Position &p) const;
  protected:
      void SetXDistance(float x);
      void SetYDistance(float y);
  private:
      float XDistance;
      float YDistance;
};

Position operator+(const Position &x, const Position &y);
```

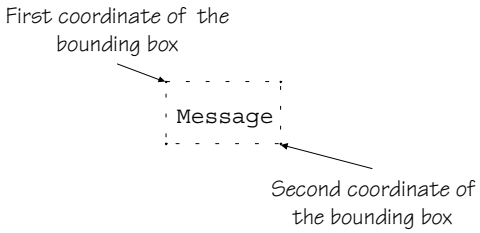
EzWindows Auxiliary Functions

- `long int GetMilliseconds()`
  - Returns the value of a timer that is ticking continuously. The resolution of the timer is milliseconds.
- `void Terminate()`
  - Sends a terminate message to the EzWindows window manager.

Class SimpleWindow

- Writing text in a window

```
void SimpleWindow::RenderText(const Position
  &UpperLeft, const Position &LowerRight,
  const string &Msg = "Message",
  const color &TextColor = Black,
  const color &BackGroundColor = White)
```



Hello EzWindows

```
#include <assert.h>
#include "ezwin.h"

// Create a 10 x 4 window
SimpleWindow HelloWindow("Hello EzWindows",
 10.0, 4.0, Position(5.0, 6.0));

// ApiMain(): create a window and display greeting
int ApiMain() {
  HelloWindow.Open();
  assert(HelloWindow.GetStatus() == WindowOpen);

  // Get Center of Window
  Position Center = HelloWindow.GetCenter();
```

Hello EzWindows

```
  // Create bounding box for text
  Position UpperLeft = Center + Position(-1.0, -1.0);
  Position LowerRight = Center + Position(1.0,  1.0);

  // Display the text
  HelloWindow.RenderText(UpperLeft, LowerRight,
   "Hello EzWindows", Black, White);

  return 0;
}
```

Hello EzWindows

```
// ApiEnd(): shutdown the window
int ApiEnd() {
  HelloWindow.Close();

  return 0;
}
```

Class SimpleWindow

- Simple Window constructor

```
SimpleWindow::SimpleWindow(
   const string &t =
   "Untitled"
   float w = 8,
   float h = 8,
   const Position &p = Position(0,0)
)
```

Bitmaps

Class BitMap

- Uses `BitMapStatus`

```
enum BitMapStatus {
    NoBitMap, BitMapOkay, NoWindow
};
```

Class BitMap

- Class BitMap can display .bmp files in a SimpleWindow window
- BitMap's constructor is
  `BitMap::BitMap(SimpleWindow &w)`
- Additional key member functions are

```
BitMapStatus BitMap::Load(string Filename)
BitMapStatus BitMap::GetStatus() const
void BitMap::SetPosition(const Position &p)
int BitMap::Draw()
int BitMap::Erase()
int BitMap::IsInside(const Position &p) const
```

Fun with Pictures

```
// Display a bit map image of the authors in the
// center of a window
#include <assert.h>
#include "bitmap.h"

// Open a window to display photograph
SimpleWindow PhotoWindow("The Authors", 10.0, 7.0,
 Position(5.0, 3.0));

// ApiMain(): display a bitmap photo
int ApiMain() {
  PhotoWindow.Open();
  assert(PhotoWindow.GetStatus() == WindowOpen);
  const Position WindowCenter =
  PhotoWindow.GetCenter();
```

Fun with Pictures

```
  // Create a bitmap
  BitMap Photo(PhotoWindow);

  // Load the image
  Photo.Load("photo.bmp");
  assert(Photo.GetStatus() == BitMapOkay);

  // Compute position of logo so it is centered
  Position PhotoPosition = WindowCenter +
   Position(-.5 * Photo.GetWidth(), -.5 *
  Photo.GetHeight());
  Photo.SetPosition(PhotoPosition);

  // Draw bitmap and we're done
  Photo.Draw();

  return 0;
}
```

Fun with Pictures

Mouse Events

- Before we can react to a mouse event in a **`SimpleWindow`**
  - Must tell window what function to call when an event occurs
    - Registering a callback
- To register a callback use the SimpleWindow member function **`SetMouseClickCallback`**.

```
W1.SetMouseClickCallback(f);
```

  - Says if the mouse is clicked in window **`W1`**, call function **`f()`**
    - **`f()`** is passed a **`Position`** that is the coordinate of the location of the mouse when the button was clicked

Mouse Events

```
int ApiMain() {
   // Open the window
   W1.Open();
   assert(W1.GetStatus() == WindowOpen);

   // Load the image
   B.Load("wizards.bmp");
   assert(B.GetStatus() == BitMapOkay);

   // Display the bit maps at a starting position
   B.SetPosition(Position(1.0, 1.0));
   B.Draw();

   // Register the callbacks for each window
   W1.SetMouseClickCallback(ReceiveMouseClick);

   return 0;
}
```

Mouse Events

```
#include <assert.h>
#include "bitmap.h"
SimpleWindow W1("Window One", 10.0, 7.0, Position(1.0,
   1.0));

BitMap B(W1); // Define a bitmap

// Mouse callback function
int ReceiveMouseClick(const Position &p) {
   // Erase the bitmap
   B.Erase();

   // Set its new position and display it
   B.SetPosition(p);
   B.Draw();

   return 1;
}
```

Timer Events

- The **`SimpleWindow`** class supports a timer mechanism
  - You can set a timer to go off periodically
  - When the timer goes off, a call back is made to the function specified by the user

Timer Functions

- **`void SimpleWindow::SetTimerCallback( TimerTickCallbackFunction f)`**
  - Registers a callback for a timer tick
  - Function `f()` will be called when a timer tick occurs.
  - The function `f()` must be declared to take no parameters, and it should return an `int`
  - The return value of `f()` indicates whether the event was handled successfully
  - A value of 1 is to indicate success
  - A value of 0 is to indicate an error occurred

Timer Functions

- `int SimpleWindow::StartTimer(int Interval)`
  - Starts timer running
  - Parameter Interval is the number of milliseconds between timer events
  - The return value indicates whether the timer was successfully started
  - A return value of 1 indicates success
  - A return value of 0 indicates the timer could not be set up
- `void SimpleWindow::StopTimer()`
  - Turns timer off

Example

```
#include <assert.h>
#include "bitmap.h"

SimpleWindow W1("Fun", 15.0, 9.0, Position(1.0, 1.0));

BitMap B(W1); // Define a bitmap

// W1TimerEvent(): move bitmap to a new location
int W1TimerEvent() {
  // Erase the bitmap
  B.Erase();
  // Compute a new position and display it
  // Make sure the bitmap is completely in the window
  int XCoord = Uniform(1, W1.GetWidth());
  if (XCoord + B.GetWidth() > W1.GetWidth())
      XCoord = XCoord - B.GetWidth();
  int YCoord = Uniform(1, W1.GetHeight());
  if (YCoord + B.GetHeight() > W1.GetHeight())
      YCoord = YCoord - B.GetHeight();
  B.SetPosition(Position(XCoord, YCoord));
  B.Draw();
}
```

Example

```
int ApiMain() {
  W1.Open(); // Open the window
  assert(W1.GetStatus() == WindowOpen);
  B.Load("davidson.bmp"); // Load the image
  assert(B.GetStatus() == BitMapOkay);

  // Display the bit maps at a starting position
  B.SetPosition(Position(1.0, 1.0));
  B.Draw();

  // Register the callbacks for each window
  // and start the timers to go off every 500 ms
  W1.SetTimerCallback(W1TimerEvent);
  W1.StartTimer(500);
  return 0;
}
```

Example

```
int ApiEnd() {
  // Stop the timers and close the windows
  W1.StopTimer();
  W1.Close();
  return 0;
}
```