/\* CarnegieMellonGraphics Library Version 2.1.4a  
 $Id: CarnegieMellonGraphics.h,v 1.11 2003/01/07 20:24:01 jsaks Exp $  
  
 Copyright (c) 1999-2000 by Geoffrey Alan Washburn <washburn@acm.org>  
 Some portions may be copyright by others. See AUTHORS file.  
  
 If this library is distributed in a precompiled binary form:  
 Portions of the supplied binary may be copyrighted by other individuals  
 and as such subject to their own license. If modifications were  
 necessary to their code, these changes will be made available upon  
 request or in a separate source release. Currently, this may include   
 some subset of the following:  
  
 FreeType : Copyright (c) 1996-1998 by David Turner, Rober Wilhem, and  
 Werner Lemberg (Use approved under FreeType custom license)  
  
 GLTT : Copyright (c) 1998-1999 by Stephane Rehel (LGPL License)  
  
 JPEGlib : Copyright (c) 1991-1998 by Thomas G. Lane  
 (Use approved under IJG custom license)  
  
 libPNG : Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.  
 Copyright (c) 1996, 1997 Andreas Dilger  
 Copyright (c) 1998, 1999 Glenn Randers-Pehrson  
 Also copyright by "Contributing Authors":  
 John Bowler  
 Kevin Bracey  
 Sam Bushell  
 Andreas Dilger  
 Magnus Holmgren  
 Tom Lane  
 Dave Martindale  
 Glenn Randers-Pehrson  
 Greg Roelofs  
 Guy Eric Schalnat  
 Paul Schmidt  
 Tom Tanner  
 Willem van Schaik  
 Tim Wegner  
 (Use approved under custom license)  
  
 Pthreads-Win32 : Copyright (c) 1998 by FSF? (LGPL License)  
  
 Redistribution and use of this library in source and binary form, with or  
 without modification, are permitted provided that the following conditions  
 are met:  
  
 1. Redistribution of source code must retain the above copyright  
 notices, this list of conditions and the following disclaimer.  
 2. Redistribution in binary form must reproduce the above copyright  
 notices, this list of conditions and the following disclaimer in the  
 documentation and/or other materials provided with the distribution.  
 3. When redistributing a modified version of this library it must carry  
 prominent notices stating the name of the individuals(s) that altered  
 the files, the nature of the modifications, and the date they were  
 performed.  
 4. None of the above copyright holders may be used to endore or promote  
 products derived from this software without specific prior written  
 notice.  
 5. No fee is charge for redistribution or use without specific   
 prior written permission of the copyright holders.  
  
 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
 ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  
 TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  
 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE  
 LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
 SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
 INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
 ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  
 POSSIBILITY OF SUCH DAMAGE.  
  
 This code is not yet certified for mission critical applications,  
 such as navigation and/or control of aircraft.  
  
\*/  
  
  
#ifndef \_\_CARNEGIEMELLONGRAPHICS\_H\_\_  
#define \_\_CARNEGIEMELLONGRAPHICS\_H\_\_  
  
#ifdef \_MSC\_VER  
#pragma warning (disable : 4786)  
#endif  
  
#include <queue>  
#include <list>  
#include <map>  
#include <string>  
#include <vector>  
  
// if your compiler understands namespaces, this is all good  
//#define string std::string  
//#define vector std::vector  
//#define map std::map  
  
//using namespace std;  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The KeyModifiers objects act as a "set" whose domain is that of the   
 \* modifier keys that are available on a given system (usually control, shift,  
 \* and alt).   
 \* <!--  
 \* This is not as intutitive as I would like, but this is the best I have come  
 \* up with so far. However, the more complicated functionality associated with  
 \* this is currently only expected to be employed by more advanced users, so   
 \* it is not too problematic.   
 \* -->  
 \* <P>Users can only work with the named sets provided, and only operate on them   
 \* through the overloaded operators : | = Union, & = Intersection, and   
 \* ~ = Compliment. Equality and stream output support are also provided.  
 \*  
 \* <P>So for example, the set of modifers associated with holding down the control  
 \* key would simply be KeyModifiers::CONTROL. If you were to hold down both   
 \* the shift and alt keys, the modifier set would be equal to   
 \*  
 \* <pre> KeyModifiers::SHIFT | KeyModifiers::ALT </pre>  
 \*  
 \* <p>(union of the two singletons) or  
 \*  
 \* <pre> KeyModifiers::SHIFT\_AND\_ALT </pre>  
 \*  
 \* or  
 \*   
 \* <pre> KeyModifiers::ALT\_AND\_SHIFT </pre>  
 \*  
 \* <p>Predefined names have been provided for all subsets (the powerset!) of the  
 \* entire domain, since it is a small set this is reasonable.  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class KeyModifiers {  
 public:  
 KeyModifiers();  
  
 ////// Pre-defined modifier sets //////  
  
 /\*\*  
 \* Matches if there are no active modifiers   
 \*/  
 static const KeyModifiers NO\_MODIFIERS;  
  
 /\*\*   
 \* Matches "control" key  
 \*/  
 static const KeyModifiers CONTROL;  
  
 /\*\*  
 \* Matches the "shift" key  
 \*/  
 static const KeyModifiers SHIFT;  
  
 /\*\*  
 \* Matches the "alt" key  
 \*/  
 static const KeyModifiers ALT;  
  
 /\*\*  
 \* Matches if both "control" and "alt" are active  
 \*/  
 static const KeyModifiers CONTROL\_AND\_ALT;  
  
 /\*\*  
 \* Matches if both "control" and "alt" are active  
 \*/  
 static const KeyModifiers ALT\_AND\_CONTROL;  
  
 /\*\*  
 \* Matches if both "control" and "shift" are active  
 \*/  
 static const KeyModifiers CONTROL\_AND\_SHIFT;  
  
 /\*\*  
 \* Matches if both "control" and "shift" are active  
 \*/  
 static const KeyModifiers SHIFT\_AND\_CONTROL;  
  
 /\*\*  
 \* Matches if both "shift" and "alt" are active  
 \*/  
 static const KeyModifiers SHIFT\_AND\_ALT;  
  
 /\*\*  
 \* Matches if both "shift" and "alt" are active  
 \*/  
 static const KeyModifiers ALT\_AND\_SHIFT;  
  
 /\*\*  
 \* Matches when all modifiers are active  
 \*/  
 static const KeyModifiers ALL\_MODIFIERS;  
  
 /\*\*  
 \* Matches any set of modifiers  
 \*/  
 static const KeyModifiers ANY\_MODIFIERS;  
  
 ////// Overloaded operators //////  
  
 /\*\*  
 \* ~ Compliment operator on KeyModifiers  
 \*/  
 KeyModifiers& operator~();  
  
 /\*\*  
 \* |= Union-Assignment operator on KeyModifiers  
 \*/  
 KeyModifiers& operator|=(const KeyModifiers& rhs);  
  
 /\*\*  
 \* &= Intersection-Assignment operator on KeyModifiers  
 \*/  
 KeyModifiers& operator&=(const KeyModifiers& rhs);  
   
 /\*\*  
 \* == Equality tests on KeyModifiers  
 \*/  
 bool operator==(const KeyModifiers& rhs) const;  
  
 /\*\*  
 \* != Inequality tests on KeyModifiers  
 \*/  
 bool operator!=(const KeyModifiers& rhs) const;  
  
 /\*\*  
 \* | Union operator on KeyModifiers  
 \*/  
 friend KeyModifiers operator|(const KeyModifiers& lhs, const KeyModifiers& rhs);  
  
 /\*\*  
 \* & Intersection operator on KeyModifiers  
 \*/  
 friend KeyModifiers operator&(const KeyModifiers& lhs, const KeyModifiers& rhs);  
  
 private:  
 // Ignore these  
 int value;   
 KeyModifiers(int v);  
};  
  
// Declare the existance of operator<< on modifiers  
std::ostream& operator<<(std::ostream& s, const KeyModifiers& mod);  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* MouseEvents are generated whenever the user performs some action with the   
 \* mouse. The types of MouseEvents that can occur are listed below as   
 \* part of the MouseEvent::Event enumeration.   
 \*   
 \* <P>MouseEvents are designed to work in a manner so that a user can easily  
 \* "match" and filter out events in which they have interest. Basically   
 \* the user constructs a new event to match against the event they currently  
 \* have. The MouseEvent constructor has several default parameters   
 \* so the user doesn't need to specify more detail about the event than they  
 \* need.  
 \*  
 \* <P>It is also possible to simply obtain pertinent information about an  
 \* event through the accessor member functions.  
 \*  
 \* <P>For example, if the user has just removed a mouse event from the mouse input  
 \* queue, and they want to see if the mouse was clicked they can do the   
 \* following:  
 \*  
 \* <pre>  
 \* if(MouseEvent(MouseEvent::BUTTON\_CLICK\_EVENT) == an\_event)   
 \* // do something  
 \* </pre>  
 \*  
 \* <p>Or a more complicated example would be if the user wanted to check to  
 \* see if the mouse has was moved over position (x, y) while holding down the   
 \* left button they would do this:  
 \*  
 \* <pre>  
 \* if(MouseEvent(MouseEvent::MOUSE\_MOVE\_EVENT, MouseEvent::LEFT\_BUTTON, x, y) == an\_event)   
 \* // do something else  
 \* </pre>  
 \*  
 \* <p>Finally, if you wanted to check to see if the right mouse buton has been   
 \* pressed down while holding down the shift key, you would do the following:  
 \*   
 \* <pre>  
 \* if(MouseEvent(MouseEvent::ButtonDown, MouseEvent::RIGHT\_BUTTON, -1, -1,   
 \* KeyModifiers::SHIFT) == an\_event)   
 \* // do more of something else   
 \* </pre>  
 \*  
 \* <p>In this case the coordinate of (-1, -1) indicates the location at which  
 \* the button press occurred does not matter.  
 \*  
 \* <p>Note that on some platforms, certain mouse and modifier combinations will be  
 \* trapped by the operating system/window manager and will not be reported   
 \* as having taken place.  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class MouseEvent {  
 public:  
  
 /\*\*  
 \* These are the buttons that CMG recognizes. A mouse event only ever contains one button.  
 \* To find out if two buttons are down at the same time, use {@link Window#isButtonDown}.  
 \* <table>  
 \* <tr><td>NO\_BUTTON</td> <td>Matches if no button was pressed</td></tr>  
 \* <tr><td>LEFT\_BUTTON</td> <td>Matches if the left button was pressed</td></tr>   
 \* <tr><td>RIGHT\_BUTTON</td> <td>Matches if the right button was pressed</td></tr>   
 \* <tr><td>MIDDLE\_BUTTON</td> <td>Matches if the middle button was pressed</td></tr>  
 \* <tr><td>ANY\_BUTTON</td> <td>Matches if any button was pressed</td></tr>  
 \* </table>  
 \* @see Window#isButtonDown  
 \*/  
 typedef enum {  
 NO\_BUTTON, // Matches if no button was pressed  
 LEFT\_BUTTON, // Matches if the left button was pressed   
 RIGHT\_BUTTON, // Matches if the right button was pressed   
 MIDDLE\_BUTTON, // Matches if the middle button was pressed  
 ANY\_BUTTON // Matches if any button was pressed  
 } Button;  
  
 /\*\*  
 \* These are the events that CMG reports.  
 \* <table>  
 \* <tr><td>NULL\_EVENT</td> <td>For creating "NULL" mouse events</td></tr>  
 \* <tr><td>MOUSE\_MOVE\_EVENT</td> <td>Event generated if the mouse was moved</td></tr>  
 \* <tr><td>ENTER\_WINDOW\_EVENT</td> <td>Event generated if the mouse has entered the window</td></tr>  
 \* <tr><td>EXIT\_WINDOW\_EVENT</td> <td>Event generated if the mouse has exited the window</td></tr>  
 \* <tr><td>BUTTON\_CLICK\_EVENT</td> <td>Event generated if a mouse button has been clicked (Will always follow a ButtonUp event)</td></tr>  
 \* <tr><td>BUTTON\_DOWN\_EVENT</td> <td>Event generated when a mouse button is pressed</td></tr>  
 \* <tr><td>BUTTON\_UP\_EVENT</td> <td>Event generated when a mouse button is released</td></tr>  
 \* </table>  
 \*/  
 typedef enum {  
 NULL\_EVENT, // For creating "NULL" mouse events  
 MOUSE\_MOVE\_EVENT, // Event generated if the mouse was moved  
 ENTER\_WINDOW\_EVENT, // Event generated if the mouse has entered the window  
 EXIT\_WINDOW\_EVENT, // Event generated if the mouse has exited the window  
 BUTTON\_CLICK\_EVENT, // Event generated if a mouse button has been clicked  
 // (Will always follow a ButtonUp event)  
 BUTTON\_DOWN\_EVENT, // Event generated when a mouse button is pressed  
 BUTTON\_UP\_EVENT // Event generated when a mouse button is released  
 } Event;  
  
 /\*\*  
 \* Constructor for a mouse event from a set of parameters.   
 \*/  
 MouseEvent(const Event e, // Default values  
 const Button b = ANY\_BUTTON,   
 const int x = -1, // Location not applicable  
 const int y = -1, // Location not applicable  
 const KeyModifiers km = KeyModifiers::ANY\_MODIFIERS);  
  
 /\*\*  
 \* Copy constructor  
 \*/  
 MouseEvent(const MouseEvent &me);  
  
 /\*\*   
 \* Default constructor  
 \*/  
 MouseEvent();  
  
 /\*\*  
 \* Comparison operator for equality on events.  
 \*/  
 bool operator==(const MouseEvent& rhs) const;  
 /\*\*  
 \* Comparison operator for inequality on events.  
 \*/  
 bool operator!=(const MouseEvent& rhs) const;  
  
 ////// Accessors //////  
  
 /\*\*  
 \* @return The actual event information for the mouse event.  
 \*/  
 const Event getEvent() const;  
  
 /\*\*  
 \* @return Which button(s), if any, were involved with this event.  
 \*/  
 const Button getButton() const;  
  
 /\*\*  
 \* @return The X coordinate that this event occurred, if relevant, otherwise returns -1.  
 \*/  
 const int getX() const;  
  
 /\*\*  
 \* @return The Y coordinate that this event occurred, if relevant, otherwise returns -1  
 \*/  
 const int getY() const;   
  
 /\*\*  
 \* @return set of modifiers active at the time of the event, if relevant.  
 \*/  
 const KeyModifiers getModifiers() const;  
  
 private:  
 // Ignore these  
 Event event;  
 Button button;  
 int ex, ey;  
 KeyModifiers modifiers;  
};  
  
// Declare the existance of operator<< on mouse events  
std::ostream& operator<<(std::ostream& s, const MouseEvent& event);  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* KeyboardEvents are generated whenever the user presses a key.  
 \* KeyboardEvents operate in a manner very similar to that of {@link .MouseEvent}. So  
 \* in the interest of brevity, I won't repeat much of the similar information.  
 \* As before, they use default parameters, so the user only need specify as   
 \* much detail as they have interest in matching.  
 \*  
 \* <p>As before it is also, possible to simply extract the pertinent information  
 \* from an event using the accessor membor fuctions.  
 \*  
 \* <p>So for example if the user simply wants to see if the "G" key has been  
 \* pressed they can do the following:  
 \*  
 \* <pre>  
 \* if(KeyboardEvent('g') == some\_event)  
 \* // do something  
 \* </pre>   
 \*  
 \* <p>Note, that if they wanted a capital G they should do the following:  
 \*   
 \* <pre>  
 \* if(KeyboardEvent('G') == some\_event)  
 \* // do something  
 \* </pre>  
 \*  
 \* <p>rather than  
 \*  
 \* <pre>  
 \* if(KeyboardEvent('g', KeyModifiers::SHIFT) == some\_event)  
 \* // do something  
 \* </pre>  
 \*  
 \* <p>One final example would be if the user wants to detect if the Alt and   
 \* Escape keys have been pressed simultaneously they would do:  
 \*   
 \* <pre>  
 \* if(KeyboardEvent(NamedKey::ESCAPE, KeyModifiers::ALT) == some\_event)  
 \* // do something else  
 \* </pre>  
 \*  
 \* <p>Note that on some platforms, certain keyboard and modifier combinations   
 \* will a) be trapped by the operating system/window manager and will not be   
 \* reported as having taken place or b) correspond to another ASCII value  
 \* for example Control-A under Unix/X11 will not map to A with a control   
 \* modifier  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class KeyboardEvent {  
  
public:  
 /\*\*  
 \* Constructor for a keyboard event from a pair of parameters.  
 \*/  
 KeyboardEvent(const int key, // Default value  
 const KeyModifiers km = KeyModifiers::ANY\_MODIFIERS);  
  
 KeyboardEvent(const KeyboardEvent &ke);  
 KeyboardEvent();  
  
 /\*\*  
 \* Comparison operators for equality on keyboard events  
 \*/  
 bool operator==(const KeyboardEvent& rhs) const;  
  
 /\*\*  
 \* Comparison operators for inequality on keyboard events  
 \*/  
 bool operator!=(const KeyboardEvent& rhs) const;  
  
 ////// Accessors //////  
  
 /\*\*   
 \* Obtain the integer value cooresponding to the event  
 \*/  
 const int getValue() const;  
  
 /\*\*  
 \* Obtain the set of modifers active at the time of the event  
 \*/  
 const KeyModifiers getModifiers() const;  
  
 private:  
 // Ignore these  
 int value;  
 KeyModifiers modifiers;  
};  
  
// Declare the existance of operator<< on keyboard events  
std::ostream& operator<<(std::ostream& s, const KeyboardEvent& event);  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The NamedKey class acts as a namespace for constants that   
 \* correspond to keys that do not have a simple ASCII representation.  
 \*  
 \* <p>So for example, if you wanted the value that corresponds to the   
 \* letter A you would use 'A', since it can be represented using a displable  
 \* ASCII character. If you wanted the value that corresponds to the left   
 \* arrow key, you would use NamedKey::LEFTARROW.   
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class NamedKey {  
 public:  
  
 ////// Non-displayable ASCII characters //////  
 static const int BACKSPACE;  
 static const int ENTER;  
 static const int ESCAPE;  
 static const int DELETE\_KEY;  
  
 ////// Function keys //////  
 static const int FUNCTION\_1;  
 static const int FUNCTION\_2;  
 static const int FUNCTION\_3;  
 static const int FUNCTION\_4;  
 static const int FUNCTION\_5;  
 static const int FUNCTION\_6;  
 static const int FUNCTION\_7;  
 static const int FUNCTION\_8;  
 static const int FUNCTION\_9;  
 static const int FUNCTION\_10;  
 static const int FUNCTION\_11;  
 static const int FUNCTION\_12;  
   
 ////// Arrow keys //////  
 static const int LEFT\_ARROW;  
 static const int UP\_ARROW;  
 static const int RIGHT\_ARROW;  
 static const int DOWN\_ARROW;  
  
 ////// Other keys //////  
 static const int PAGE\_UP;  
 static const int PAGE\_DOWN;  
 static const int HOME;  
 static const int END;  
 static const int INSERT;  
  
 private:  
 //// Ignore ////  
 static const int BASE;  
 static const int LAST;  
 // Should not be able to create an instance of NamedKey  
 NamedKey() { };  
 friend class KeyboardEvent;  
};  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The Color class is used by the system to represent color information   
 \* (of all things). The default color object is simply black, but it is also  
 \* possible to construct a color by specifying RGB (red, green, blue) and   
 \* RGBA (red, green, blue, alpha) component values. The alpha channel   
 \* information is use to carry information about the opacity of the color.   
 \* It is possible that your renderer may not support this capability, in   
 \* which case it is simply ignored.  
 \*  
 \* <P>Currently, the internal representation of colors is unsigned characters,  
 \* and the external representation uses integers. This is not completely   
 \* satisfactory, because the user can create colors using invalid components.  
 \* Currently, component values are clamped between 0 and 255 and issues a  
 \* warning message when an out of bounds color is supplied. In future releases  
 \* it may be desirable to use an internal and external representation that  
 \* will more closely match. The internal representation can be changed to  
 \* use integers, but then the end user would have to use negative numbers to  
 \* specify some values, which is not entirely intutitive. Floating point  
 \* values have the same sort of problem.  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   
class Color {  
public:  
  
 /\*\*  
 \* Construct an empty color (which is by default, fully opaque black)  
 \*/  
 Color();  
  
 /\*\*  
 \* Construct a completely opaque color from three component hues   
 \*/  
 Color(int red, int green, int blue);  
   
 /\*\*  
 \* Construct a color from the three component hues and an alpha channel.  
 \* This may be used to provide opacity information to the renderer.  
 \*/  
 Color(int red, int green, int blue, int alpha);  
  
 /\*\*  
 \* Comparison operator for equality on color objects  
 \*/  
 bool operator==(const Color& rhs) const;  
 /\*\*  
 \* Comparison operator for inequality on color objects  
 \*/  
 bool operator!=(const Color& rhs) const;  
  
 ////// Accessors //////  
  
 // Obtain the specified component value  
 int getRed() const { return static\_cast<int>(red); }  
 int getGreen() const { return static\_cast<int>(green); }  
 int getBlue() const { return static\_cast<int>(blue); }  
 int getAlpha() const { return static\_cast<int>(alpha); }  
  
 ////// Mutators //////  
  
 // Modify the specified component value  
 void setRed (int value) { red = checkValue(value, \_\_RED); }  
 void setGreen(int value) { green = checkValue(value, \_\_GREEN); }  
 void setBlue (int value) { blue = checkValue(value, \_\_BLUE); }  
 void setAlpha(int value) { alpha = checkValue(value, \_\_ALPHA); }  
  
 ////// Some predefined colors ////////  
 static const Color BLACK;  
 static const Color WHITE;  
 static const Color RED;  
 static const Color GREEN;  
 static const Color BLUE;  
  
/\* CPPDOC\_BEGIN\_EXCLUDE \*/  
private:  
 // Ignore these  
 unsigned char red, green, blue, alpha;  
 typedef enum { \_\_RED, \_\_GREEN, \_\_BLUE, \_\_ALPHA } ComponentName;  
 std::string ComponentNameToString(ComponentName cn);  
 unsigned char checkValue(int value, ComponentName cn);  
/\* CPPDOC\_END\_EXCLUDE \*/  
};  
  
// Declare the existance of operator<< on Color objects  
std::ostream& operator<<(std::ostream& s, const Color& color);  
  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The Image class   
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class Image {  
public:  
 /\*\*  
 \* Image types currently supported  
 \*  
 \* <table>  
 \* <tr><td>JPEG</td> <td>JPEG Images</td></tr>  
 \* <tr><td>PNG</td> <td>PNG Images</td></tr>  
 \* <tr><td>SCREEN</td><td>An image created from a region in a window</td></tr>  
 \* </table>  
 \*/  
 typedef enum {  
 JPEG, // JPEG images  
 PNG, // PNG images  
 SCREEN, // An image created from region in a window  
 NO\_TYPE,  
 } Type;  
  
 Image(const std::string &file, const Type type);  
 Image();  
 ~Image();  
  
 /\*\*  
 \* Copy constructor   
 \*/   
 Image(const Image& image);  
  
 /\*\*  
 \* Assignment operator  
 \*/  
 Image& operator=(const Image &rhs);  
  
 /\*\*  
 \* Comparison operator for equality on images  
 \*/  
 bool operator==(const Image& rhs) const;  
 /\*\*  
 \* Comparison operator for inequality on images  
 \*/  
 bool operator!=(const Image& rhs) const;  
  
 /\*\*  
 \* Save this image to a file of the specified type.  
 \*/  
 void save(const std::string &file, const Type type) const;  
  
 ////// Accessors //////  
  
 /\*\*   
 \* Get the width of the image  
 \*/  
 int getWidth() const;  
  
 /\*\*  
 \* Get the height of the image  
 \*/  
 int getHeight() const;  
  
 /\*\*   
 \* Get the type of the image  
 \*/  
 Type getType() const;  
  
 /\*\*  
 \* Class function for checking to see whether an Image exists   
 \*/  
 static bool checkImage(const std::string &file, const Type type);  
  
 /\*\*   
 \* Copy out a sub-portion of the image and return it.  
 \* This is very useful for creating an animation in a single image and then   
 \* chopping it into an array of sprites in your program.  
 \*/  
 Image subImage(int x,int y,int w,int h);  
  
protected:  
/\* CPPDOC\_BEGIN\_EXCLUDE \*/  
 //// Ignore these ////  
 int width, height;  
 int imagehandle;  
 Type type;  
  
 Image(int handle, int width, int height, Type type);  
  
 friend class DaemonImp;  
 friend class RENDERER;  
 // this should be unnecessary  
 friend class EditableImage;  
/\* CPPDOC\_END\_EXCLUDE \*/  
};  
  
// Declare the existance of operator<< on keyboard events  
std::ostream& operator<<(std::ostream& s, const Image& image);  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Images that can be manipulated at the pixel level. EditableImages should  
 \* be used when you want to either do lots and lots of pixel manipulation quickly  
 \* or be able to modify other images. You can use EditableImages just as you would  
 \* Images, just that you can manipulate pixels.  
 \* @see .Window#drawImage  
 \* @since 2.1.4a  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class EditableImage : public Image {  
public:  
 /\*\*  
 \* Copy constructor  
 \*/  
 EditableImage(const EditableImage &image);  
 /\*\*  
 \* Cast Image object up to an EditableImage  
 \*/  
 EditableImage(const Image &image);  
 /\*\*  
 \* Constructor to make an image that's width x height.   
 \*/  
 EditableImage(int width, int height);  
  
 ~EditableImage();  
  
 /\*\*  
 \* Assignment operator. As always, assignment is done via references,  
 \* so this is a "shallow" copy of the image. Changes to either the left-hand-side or  
 \* right-hand-side will affect the other.   
 \* @see #clone  
 \*/  
 EditableImage &operator=(const EditableImage &image);  
   
 /\*\*  
 \* gets the color of the pixel at (x,y)  
 \*/  
 Color getPixel(int x, int y);  
 /\*\*  
 \* sets the color of the pixel at (x,y)  
 \*/  
 void setPixel(int x, int y, const Color &c);  
 /\*\*  
 \* sets the color of the pixel at (x,y)  
 \*/  
 void setPixel(int x, int y, unsigned char r, unsigned char g, unsigned char b, unsigned char a = 255);  
 /\*\*  
 \* return a "deep" copy of this image  
 \*/  
 EditableImage clone();  
protected:  
 /\* CPPDOC\_BEGIN\_EXCLUDE \*/  
 EditableImage clone(int imagehandle, int width, int height);  
 EditableImage(int imagehandle, int width, int height);  
 unsigned char \*data;  
 bool editable;  
 int dataheight, datawidth;  
 void \*imageinstance;  
 /\* CPPDOC\_END\_EXCLUDE \*/  
};  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Class for representing fonts. A new font object can be created given   
 \* the location of a TrueType font file, and a point size.  
 \*   
 \* <p>For example to create a font for 20 point Arial, the user would do the   
 \* following:  
 \*   
 \* <pre>  
 \* Font myArial("/where/I/keep/my/fonts/arial.ttf", 20);  
 \* </pre>  
 \*  
 \* <P>The string for specifing the font location is dependent upon the platform  
 \* used.  
 \*   
 \* <P>The user may also use the mutators to alter the font after creation.  
 \* <P><B>Update:</B> You can now also create a font using one of the built-in fonts. See {@link #Font(int,int)}.  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class Font {  
public:  
 /\*\*  
 \* Constructor for creating fonts from a TrueType font file. <B>Warning:</B> Not supported on MacOS.  
 \*/  
 Font(const std::string &face, const int pointsize);  
 /\*\*  
 \* Constructor for creating fonts from one of the builtin fonts.  
 \* @see #ROMAN  
 \* @see #MONO\_ROMAN  
 \* @see #HELVETICA  
 \* @see #TIMES  
 \* @since 2.10  
 \*/  
 Font(int fontID, int pointsize);  
 ~Font();  
  
 /\*\*  
 \* Copy constructor  
 \*/  
 Font(const Font& font);  
 /\*\*  
 \* Assignment operator  
 \*/  
 Font& operator=(const Font &rhs);  
  
 /\*\*  
 \* Comparison operator for equality on fonts  
 \*/  
 bool operator==(const Font& rhs) const;  
 /\*\*  
 \* Comparison operator for inequality on fonts  
 \*/  
 bool operator!=(const Font& rhs) const;  
  
 ////// Accessors //////  
  
 std::string getFace() const;  
 int getPointSize() const;  
  
 ////// Mutators //////  
   
 void setFace(const std::string &face);  
 void setPointSize(const int pointsize);  
  
 /\*\*  
 \* Class function for checking to see whether a font exists   
 \*/  
 static bool checkFont(const std::string &face, const int pointsize);  
   
 /\*\*  
 \* Get the dimensions of the specified text. Put the results in width and height.  
 \* @param text [in] Text to measure  
 \* @param width [out] Width of the text.  
 \* @param height [out] Height of the text.  
 \* @since 2.1.2   
 \*/  
 void getStringSize(const std::string &text, int &width, int &height);  
  
 /\*\*  
 \* stroked font, can be used at any resolution  
 \*/  
 static const int ROMAN;  
 /\*\*  
 \* stroked font, can be used at any resolution  
 \*/  
 static const int MONO\_ROMAN;  
 /\*\*  
 \* bitmapped fonts, must be used at specific sizes;  
 \* helvetica sizes: 10,12, 18.  
 \*/  
 static const int HELVETICA;  
 /\*\*  
 \* bitmapped fonts, must be used at specific sizes;  
 \* times sizes: 10, 24.  
 \*/  
 static const int TIMES;  
  
 private:  
 ////// Ignore //////  
 std::string name;  
 int size;  
 int fonthandle;  
  
 friend class DaemonImp;  
  
};  
  
// Declare the existance of operator<< on keyboard events  
std::ostream& operator<<(std::ostream& s, const Font& font);  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Style objects are used to encapsulate several pieces of style information  
 \* to be passed to drawing calls. At the most basic level styles store color  
 \* information, but they also contain information about line-widths,   
 \* the type of raster operation to use when rendering, and what pattern to use  
 \* when drawing lines and outline primatives.  
 \*  
 \* <P>So if you want to draw a green line you could create a style like so:  
 \*   
 \* <pre>   
 \* Style mystyle = Style(Color::GREEN);   
 \* </pre>  
 \*  
 \* <P>or  
 \*  
 \* <pre>  
 \* Style mystyle = Style(Color(0,0,255));  
 \* </pre>  
 \*  
 \* <P>However, when you draw your line it will only be one pixel thick. If you  
 \* want a thicker green line (3 pixels wide) you can do the following:  
 \*  
 \* <pre>  
 \* Style mystyle = Style(Color::GREEN, 3);  
 \* </pre>  
 \*   
 \* <P>In another situation you might want to create an rubber-banding box by  
 \* XORing and outline rectangle, so you would create the following style:  
 \*  
 \* <pre>  
 \* Style mystyle = Style(Color::BLUE, 2, Style::XOR\_OP);  
 \* </pre>  
 \*  
 \* <P>Patterns for lines, arcs, and curves are represented using 16 bits.  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class Style {  
private:  
 // Ignore   
 static std::vector<bool> defaultpattern;  
   
public:  
 /\*\*  
 \* Possible raster operations   
 \* <table>  
 \* <tr><td>COPY\_OP</td> <td>Simply draw the color, default behavior</td></tr>  
 \* <tr><td>INVERT\_OP</td> <td>Invert the drawing surface's color</td></tr>  
 \* <tr><td>XOR\_OP</td> <td>XOR drawing surface with color</td></tr>  
 \* <tr><td>AND\_OP</td> <td>AND drawing surface with color</td></tr>  
 \* <tr><td>OR\_OP</td> <td>OR drawing surface with color</td></tr>  
 \* </table>  
 \*/  
 typedef enum {  
 COPY\_OP, // Simply draw the color, default behavior  
 INVERT\_OP, // Invert the drawing surface's color  
 XOR\_OP, // XOR drawing surface with color  
 AND\_OP, // AND drawing surface with color  
 OR\_OP // OR drawing surface with color  
 } RasterOp;  
  
 /\*\*  
 \* Construct a style from a set of parameters  
 \*/  
 Style(Color col, // Default values  
 int lwidth = 1,   
 RasterOp rop = COPY\_OP,   
 const std::vector<bool> &newpattern = defaultpattern);  
  
 Style(const Style &s);  
 Style &operator=(const Style& rhs);  
  
 /\*\*  
 \* Comparison operator for equality on style objects  
 \*/  
 bool operator==(const Style& rhs) const;  
 /\*\*  
 \* Comparison operator for inequality on style objects  
 \*/  
 bool operator!=(const Style& rhs) const;  
  
 ////// Accessors //////  
  
 Color getColor() const;  
 int getLineWidth() const;  
 RasterOp getRasterOp() const;  
 std::vector<bool> getPattern() const;  
 unsigned short getPatternAsUShort() const;  
  
 ////// Mutators //////  
  
 void setRasterOp(RasterOp rop);  
 void setColor(Color col);  
 void setLineWidth(int value);  
 void flipPatternBit(int position);  
 void setPattern(const std::vector<bool> &newpattern);  
  
 ////// Some predefined styles //////  
  
 static const Style BLACK;  
 static const Style WHITE;  
 static const Style RED;  
 static const Style GREEN;  
 static const Style BLUE;  
 static const Style INVERTED;  
  
 private:  
 // Ignore these  
 Color color;   
 int linewidth;  
 RasterOp rasterop;  
 unsigned short pattern;  
 unsigned short bvectorToUShort(const std::vector<bool> &pattern);  
   
 friend class RENDERER;  
};  
  
// Declare the existance of operator<< on Style objects  
std::ostream& operator<<(std::ostream& s, const Style& style);  
  
/\*\*  
 \* TimerEvents are used to keep track of events that happen at regular intervals.  
 \* <B>Note:</B> Do not instantiate one directly, use {@link .Window#registerNewTimerEvent} or   
 \* {@link .Window#startTimer} to create one.  
 \*/  
class TimerEvent {  
 public:  
 TimerEvent(int timerid);  
 TimerEvent();  
 TimerEvent(const TimerEvent &te);  
  
 // Comparison for timer events  
 bool operator==(const TimerEvent& rhs) const;  
 bool operator!=(const TimerEvent& rhs) const;   
  
 int getTimerID() const;  
  
  
 private:  
 //// Ignore ////  
 int timerid;  
};  
  
// Declare the existance of operator<< on timer event objects  
std::ostream& operator<<(std::ostream& s, const TimerEvent& timer);  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Transform objects are used to encapsulate image transformations:  
 \* rotations and scales. A transform object can be used to modify how  
 \* an image is displayed. Images displayed transformed might draw  
 \* slower than untransformed images, depending on your system.  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class Transform {  
public:  
 Transform();  
 Transform(double rotation);  
 Transform(double sx,double sy);  
 Transform(double rotation,double sx,double sy);  
  
 void setRotation(double theta);  
 void setScale(double sx,double sy);  
 void setScaleX(double sx);  
 void setScaleY(double sy);  
  
 double getScaleX() const;  
 double getScaleY() const;  
 double getRotation() const;  
  
 static const double SCALE;  
private:  
 double scaleX,scaleY,theta;  
};  
  
std::ostream& operator<<(std::ostream &s, const Transform &transform);  
  
/\* CPPDOC\_BEGIN\_EXCLUDE \*/  
template <class T> class SemaphoreT;  
/\* CPPDOC\_END\_EXCLUDE \*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The Window class is used to represent that actual drawing surfaces.   
 \* Currently on creating a new window object, a new window will open in user's  
 \* operating environment. The background color is black by default.   
 \*  
 \* <P>Two different input paradigms have been provided. One is event based,  
 \* and requires deriving a new Window class and then starting the CarnegieMellonGraphics  
 \* event handler (described below). For this derived class to do something  
 \* interesting, one or more of the "handle" member functions listed below   
 \* must be overridden.   
 \*  
 \* <P>The other method of input works within the standard structured programming  
 \* framework, and involves polling the "isQueueEmpty" functions to see if   
 \* an event has occurred, removing the event from the queue, and processing it  
 \* as desired.  
 \*  
 \* <P>Drawing into the window is then simply a matter of calling the appropriate   
 \* member functions on the window.  
 \*  
 \* <P><B>Updates:</B>   
 \* <UL>  
 \* <LI>A new method of input has been added: see {@link #isKeyDown} and {@link #isButtonDown}.   
 \* <LI>To go into fullscreen mode, specify this as an additional parameter to the constructor: {@link #Window}.  
 \* </UL>  
 \*   
 \* <P>If you're working on a game, you will probably be interested in investigating  
 \* the following functions: {@link #disableAutoPageFlip}, {@link #flipPage}, {@link #isKeyDown},   
 \* {@link #isButtonDown}, {@link #isModifierDown}, and {@link #Window} (fullscreen mode).  
 \*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class Window {  
 public:  
 /\*\*  
 \* Construct a new window with the specified size and title.  
 \* (defaults to 640 by 480, with "CarnegieMellonGraphics Window" as the title)   
 \*  
 \* <p>This is the appropriate way to go into full screen mode. To create a regular  
 \* windowed application, construct a window as follows:  
 \*  
 \* <pre>  
 \* Window myWindow();  
 \* </pre>  
 \*  
 \* <p>or  
 \*  
 \* <pre>  
 \* Window myWindow(640,480,"my window"); // the last parameter defaults to false  
 \* </pre>  
 \*  
 \* <p>To create a fullscreen window, pass true as the fourth parameter:  
 \*  
 \* <pre>  
 \* Window myFullScreenWindow(640,480,"my full screen window",true);  
 \* </pre>  
 \*  
 \* If the dimensions of the window are not a standard dimension or one that is not  
 \* supported by your video card, the window will probably not be constructed and no  
 \* error will be reported. Dimensions such as <tt>320x200, 320x240, 400x300, 640x480,  
 \* 800x600, 1024x768, etc.</tt> are recommended for full screen mode.  
 \*/  
 Window(const int width = 640, const int height = 480, const std::string &title = std::string("CarnegieMellonGraphics Window"), bool fullscreen=false,int bitdepth=32,int refreshrate=60);  
 /\*\*   
 \* An alternate constructor where you get to choose the position of the window as well as   
 \* all of the other usual parameters.  
 \* @since 2.1.5  
 \*/  
 Window(int xpos, int ypos, int width, int height, const std::string &title = std::string("CarnegieMellonGraphics Window"), bool fullscreen=false,int bitdepth=32,int refreshrate=60);  
 virtual ~Window();  
  
 /\*\*   
 \* Hide this window.  
 \* @see #isHidden  
 \* @see #show  
 \*/  
 void hide();  
 /\*\*   
 \* Show this window.  
 \* @see #hide  
 \* @see #isHidden  
 \*/  
 void show();  
 /\*\*  
 \* Tell if this window is hidden or not.  
 \* @see #hide  
 \* @see #show  
 \*/  
 bool isHidden();  
  
 /\*\*  
 \* Return the width of the window.  
 \* @see #getHeight  
 \*/  
 int getWidth() const;  
 /\*\*  
 \* Return the height of the window.  
 \* @see #getWidth  
 \*/  
 int getHeight() const;  
  
 /\*\*  
 \* Return the last location of mouse X position.  
 \* @see #getMouseEvent  
 \* @see #getMouseY  
 \* @see #isButtonDown  
 \*/  
 int getMouseX() const;  
 /\*\*  
 \* Return the last location of mouse Y position.  
 \* @see #getMouseEvent  
 \* @see #getMouseX  
 \* @see #isButtonDown  
 \*/  
 int getMouseY() const;  
  
 /\*\*  
 \* Register a timer event to occur after the specified number of milliseconds. Only causes the timer  
 \* to go off once. If you want repeated occurrences, you must re-register when you recieve the event.  
 \* @see #getTimerCount  
 \* @see #getTimerEvent  
 \* @see #setTimerCount  
 \* @see #startTimer  
 \* @see #stopTimer  
 \* @deprecated This has unnecessary overhead. Use {@link #startTimer} instead.  
 \*/  
 TimerEvent registerNewTimerEvent(const int milliseconds);  
  
 /\*\*  
 \* Register a timer event to occur every specified interval until you stop it using {@link #stopTimer}.  
 \* @see #getTimerCount  
 \* @see #getTimerEvent  
 \* @see #isTimerQueueEmpty  
 \* @see #setTimerCount  
 \* @see #stopTimer  
 \* @since 2.1.5  
 \*/  
 TimerEvent startTimer(int milliseconds);  
  
 /\*\*  
 \* Stop a timer that was started with {@link #startTimer}.  
 \* @see #getTimerCount  
 \* @see #getTimerEvent  
 \* @see #isTimerQueueEmpty  
 \* @see #setTimerCount  
 \* @see #startTimer  
 \* @since 2.1.5  
 \*/  
 void stopTimer(const TimerEvent &timer);  
  
 /\*\*  
 \* Check whether there is input sitting in the Keyboard Queue.  
 \* @see #flushKeyboardQueue  
 \* @see #getKeyboardEvent  
 \* @see #waitForKeyboardEvent  
 \*/  
 bool isKeyboardQueueEmpty();  
 /\*\*  
 \* Check whether there is input sitting in the Mouse Queue.  
 \* @see #flushMouseQueue  
 \* @see #getMouseEvent  
 \* @see #waitForMouseEvent  
 \*/  
 bool isMouseQueueEmpty();  
 /\*\*  
 \* Check whether there is input sitting in the Timer Queue.  
 \* @see #flushTimerQueue  
 \* @see #getTimerEvent  
 \* @see #waitForTimerEvent  
 \* @see #registerNewTimerEvent  
 \*/  
 bool isTimerQueueEmpty();  
  
 /\*\*  
 \* Pull events off the front of the Keyboard event queue.   
 \* Use {@link #isKeyDown} if you're interested in less precise input processing (i.e. when you're making a game).  
 \* @see #flushKeyboardQueue  
 \* @see #ignoreKeyRepeat  
 \* @see #isKeyboardQueueEmpty  
 \* @see #isKeyDown  
 \* @see #waitForKeyboardEvent  
 \*/  
 KeyboardEvent getKeyboardEvent();  
 /\*\*  
 \* Pull events off the front of the Mouse event queue  
 \* Use {@link #isButtonDown} if you're interested in less precise input processing (i.e. when you're making a game).  
 \* @see #flushMouseQueue  
 \* @see #isButtonDown  
 \* @see #isMouseQueueEmpty  
 \* @see #waitForMouseEvent  
 \*/  
 MouseEvent getMouseEvent();  
 /\*\*  
 \* Pull events off the front of the Timer event queue  
 \* Use {@link #getTimerCount} to get lower-overhead event processing.  
 \* @see #flushTimerEvent  
 \* @see #isTimerQueueEmpty  
 \* @see #waitForTimerEvent  
 \* @see #registerNewTimerEvent  
 \*/  
 TimerEvent getTimerEvent();  
  
 /\*\*  
 \* Wait for a keyboard event to occur and then return the event. If there are   
 \* already events in the queue the function will return immediately.  
 \* @see #flushKeyboardQueue  
 \* @see #getKeyboardEvent  
 \* @see #ignoreKeyRepeat  
 \* @see #isKeyboardQueueEmpty  
 \* @see #isKeyDown  
 \*/  
 KeyboardEvent waitForKeyboardEvent();  
 /\*\*  
 \* Wait for a mouse event to occur and then return the event. If there are   
 \* already events in the queue the function will return immediately.  
 \* @see #flushMouseQueue  
 \* @see #getMouseEvent  
 \* @see #isButtonDown  
 \* @see #isMouseQueueEmpty  
 \*/  
 MouseEvent waitForMouseEvent();  
 /\*\*  
 \* Wait for a timer event to occur and then return the event. If there are   
 \* already events in the queue the function will return immediately.  
 \* @see #flushTimerQueue  
 \* @see #getTimerCount  
 \* @see #getTimerEvent  
 \* @see #registerNewTimerEvent  
 \* @see #isTimerQueueEmpty  
 \*/  
 TimerEvent waitForTimerEvent();  
  
 /\*\*  
 \* Remove all pending messages in the keyboard event queue.  
 \* @see #autoFlushKeyboardQueue  
 \* @see #getKeyboardEvent  
 \* @see #isKeyboardQueueEmpty  
 \* @see #isKeyDown  
 \* @see #waitForKeyboardEvent  
 \*/  
 void flushKeyboardQueue();  
 /\*\*  
 \* Remove all pending messages in the mouse event queue.  
 \* @see #autoFlushMouseQueue  
 \* @see #getMouseEvent  
 \* @see #isButtonDown  
 \* @see #isMouseQueueEmpty  
 \* @see #waitForMouseEvent  
 \*/  
 void flushMouseQueue();  
 /\*\*  
 \* Remove all pending messages in the timer event queue.  
 \* @see #autoFlushTimerQueue  
 \* @see #getTimerCount  
 \* @see #getTimerEvent  
 \* @see #isTimerQueueEmpty  
 \* @see #registerNewTimerEvent  
 \* @see #waitForTimerEvent  
 \*/  
 void flushTimerQueue();  
  
 /\*\*  
 \* Tells whether the specified key is down. <tt>key</tt> can either be an ascii character,  
 \* like 'a' or a {@link NamedKey}, like {@link NamedKey#ESCAPE}. <tt>isKeyDown</tt> will   
 \* always return the same value for lowercase and uppercase version of the same letter.   
 \* This is because there are ways to have a key be pressed in its lowercase form and then be  
 \* raised in its uppercase form (e.g. press 'a', then press shift, then release 'a'). You can  
 \* use {@link #isModifierDown} to test if a modifier was pressed at the time that the character was.  
 \* @since 2.1.2  
 \* @see #autoFlushKeyboardQueue  
 \* @see #flushKeyboardQueue  
 \* @see #getKeyboardEvent  
 \* @see #ignoreKeyRepeat  
 \* @see #isKeyboardQueueEmpty  
 \* @see #isKeyDown  
 \* @see #isModifierDown  
 \* @see #waitForKeyboardEvent  
 \*/  
 bool isKeyDown(int key);  
 /\*\*  
 \* Tells whether a specified mouse button is down. You can check for "chording" of mouse buttons  
 \* in this fashion. For example:  
 \* <pre>  
 \* if (isButtonDown(MouseEvent::LEFT\_BUTTON) && isButtonDown(MouseEvent::RIGHT\_BUTTON))   
 \* // do something if both buttons are down  
 \* </pre>  
 \* <p>This is the recommended method of input if you are writing a game. If you need absolute   
 \* precision, then do not use this function. There is no guarantee that if a button is pressed  
 \* checking it in this fashion will ever show that it has been pressed.  
 \* <p>This should probably be used in conjuction with {@link #autoFlushMouseQueue}.  
 \* @since 2.1.2  
 \* @see #autoFlushMouseQueue  
 \* @see #flushMouseQueue  
 \* @see #getMouseEvent  
 \* @see #isButtonDown  
 \* @see #isModifierDown  
 \* @see #isMouseQueueEmpty  
 \* @see #waitForMouseEvent  
 \*/  
 bool isButtonDown(MouseEvent::Button b);  
 /\*\*  
 \* Tells whether the specified modifier was pressed during the last recorded mouse or keyboard  
 \* event. This will be accurate if there is a lot of keyboard/mouse input going on, but will probably  
 \* be innaccurate in other instances. It is recommended that if you need precision to   
 \* use the eventqueues. This is an approximate way of getting whether a modifier is down.  
 \* @since 2.1.2  
 \* @see #isButtonDown  
 \* @see #isKeyDown  
 \*/  
 bool isModifierDown(KeyModifiers km);  
  
 /\*\*  
 \* Ignore repeated keystrokes sent to this window. If this is disabled, keystrokes will  
 \* occur repeatedly which might be unwanted for games using the keyboard.  
 \* @since 2.1.2  
 \* @see #autoFlushKeyboardQueue  
 \* @see #getKeyboardEvent  
 \* @see #isKeyDown  
 \*/  
 void ignoreKeyRepeat(bool ignore);  
  
 /\*\*  
 \* Get the number of times this timer event has occurred. Generally used in   
 \* conjunction with {@link @setTimerCount}, this is very good for making sure  
 \* that your program keeps a time (like maintaining a framerate).   
 \* @see #getTimerEvent  
 \* @see #registerNewTimerEvent  
 \* @see #setTimerCount  
 \* @since 2.1.2  
 \*/  
 int getTimerCount(const TimerEvent &te);  
 /\*\*  
 \* Changes the count for a timer. This is useful for resetting a timer's count  
 \* once you've checked it.  
 \* @see #getTimerCount  
 \* @see #getTimerEvent  
 \* @see #registerNewTimerEvent  
 \* @since 2.1.2  
 \*/  
 void setTimerCount(const TimerEvent &te, int count = 0);  
  
 /\*\*  
 \* Automatically lets you ignore the events in the keyboard message queue. You should use  
 \* this if you want to ignore the event queue altogether. Enabling/disabling this will not  
 \* affect {@link #isKeyDown}. It is possible that the queue will at some point contain   
 \* messages, but it will be flushed eventually.  
 \* @see #autoFlushMouseQueue  
 \* @see #autoFlushTimerQueue  
 \* @see #flushKeyboardQueue  
 \* @see #getKeyboardEvent  
 \* @see #isKeyboardQueueEmpty  
 \* @see #isKeyDown   
 \* @see #isModifierDown  
 \* @since 2.1.2  
 \*/  
 void autoFlushKeyboardQueue(bool enable);  
 /\*\*  
 \* Automatically lets you ignore the events in the mouse message queue. You should use  
 \* this if you want to ignore the event queue altogether. Enabling/disabling this will not  
 \* affect {@link #isButtonDown}. It is possible that the queue will at some point contain   
 \* messages, but it will be flushed eventually.  
 \* @see #autoFlushKeyboardQueue  
 \* @see #autoFlushTimerQueue  
 \* @see #flushMouseQueue  
 \* @see #getMouseEvent  
 \* @see #isButtonDown  
 \* @see #isModifierDown  
 \* @see #isMouseQueueEmpty  
 \* @since 2.1.2  
 \*/  
 void autoFlushMouseQueue(bool enable);  
 /\*\*  
 \* Automatically lets you ignore the events in the timer message queue. You should use  
 \* this if you want to ignore the event queue altogether. Enabling/disabling this will not  
 \* affect {@link #getTimerCount}. It is possible that the queue will at some point contain   
 \* messages, but it will be flushed eventually.  
 \* @see #autoFlushKeyboardQueue  
 \* @see #autoFlushMouseQueue  
 \* @see #flushTimerQueue  
 \* @see #getTimerCount  
 \* @see #getTimerEvent  
 \* @see #isTimerQueueEmpty  
 \* @see #registerNewTimerEvent  
 \* @since 2.1.2  
 \*/  
 void autoFlushTimerQueue(bool enable);  
 /\*\*   
 \*   
 \*/  
// void ignoreRepeatKeys(bool enable);  
  
 /\*\*  
 \* Functions to add events to the input queues.  
 \* These should generally not be used, but it is  
 \* possible they may be useful in some circumstances.  
 \* <P><B>Note: </B> calling this function will not set the key to be down  
 \* @see #getKeyboardEvent  
 \*/  
 void addKeyboardEvent(const KeyboardEvent& keyboard);  
 /\*\*  
 \* Functions to add events to the input queues.  
 \* These should generally not be used, but it is  
 \* possible they may be useful in some circumstances.  
 \* <P><B>Note: </B> calling this function will not set the button to be down  
 \* @see #getMouseEvent  
 \*/  
 void addMouseEvent(const MouseEvent& mouse);  
 /\*\*  
 \* Functions to add events to the input queues.  
 \* These should generally not be used, but it is  
 \* possible they may be useful in some circumstances.  
 \* @see #getTimerEvent  
 \* @see #registerNewTimerEvent  
 \*/  
 void addTimerEvent(const TimerEvent& timer);  
  
 ////// Drawing commands //////  
  
 /\*\*  
 \* Draw a pixel at (x,y) with style s.  
 \* If you want to do a lot of bit manipulation, you should use {@link .EditableImage}.   
 \* drawPixel can be slow if you do a lot of pixel-level editing.  
 \* @see .EditableImage  
 \*/  
 void drawPixel(const Style &s, const int x, const int y);  
  
 /\*\*  
 \* Draw a line from (x1,y1) to (x2, y2) with style s  
 \*/  
 void drawLine(const Style &s, const int x1, const int y1, const int x2, const int y2);  
  
 /\*\*  
 \* Draw a polyline using the specified style and the two vectors to specifiy vertices  
 \*/  
 void drawPolyLine(const Style &s, const std::vector<int>& xcoords, const std::vector<int>& ycoords);  
  
 /\*\*  
 \* Draw an elliptical arc from angle start to end in degrees. The x and y  
 \* coordinates specify the bounding box of the ellipse   
 \*/  
 void drawArc(const Style &s, const int x1, const int y1, const int x2, const int y2, const double start, const double end);  
  
 /\*\*  
 \* Draw a filled rectangle with (x1, y1) as the upper left-hand corner,  
 \* and (x2, y2) as the lower right-hand corner with style s  
 \*/  
 void drawRectangleFilled(const Style &s, const int x1, const int y1, const int x2, const int y2);  
  
 /\*\*  
 \* Draw a rectangle outline with (x1, y1) as the upper left-hand corner,  
 \* and (x2, y2) as the lower right-hand corner with style s  
 \*/  
 void drawRectangleOutline(const Style &s, const int x1, const int y1, const int x2, const int y2);  
  
 /\*\*  
 \* Draw a filled triangle with vertices (x1, y1), (x2, y2), (x3, y3) with in style s  
 \*/  
 void drawTriangleFilled(const Style &s, const int x1, const int y1, const int x2, const int y2, const int x3, const int y3);  
  
 /\*\*   
 \* Draw a triangle outline with vertices (x1, y1), (x2, y2), (x3, y3) with in style s  
 \*/  
 void drawTriangleOutline(const Style &s, const int x1, const int y1, const int x2, const int y2, const int x3, const int y3);  
  
 /\*\*  
 \* Draw a filled circle with style s and the specified radius at point (x, y)  
 \*/  
 void drawCircleFilled(const Style &s, const int x, const int y, const int radius);  
  
 /\*\*  
 \* Draw a circle outline with style s and the specified radius at point (x, y)  
 \*/  
 void drawCircleOutline(const Style &s, const int x, const int y, const int radius);  
  
 /\*\*  
 \* Draw a filled ellipse with inside the bounding box specfied by the two   
 \* points (x1, y1) and (x2, y2)  
 \*/  
 void drawEllipseFilled(const Style &s, const int x1, const int y1, const int x2, const int y2);  
  
 /\*\*  
 \* Draw an ellipse outline with inside the bounding box specified by the two  
 \* points (x1, y1) and (x2, y2)  
 \*/  
 void drawEllipseOutline(const Style &s, const int x1, const int y1, const int x2, const int y2);  
  
 /\*\*  
 \* Draw a filled elliptical wedge with from start to end degrees  
 \* The two points (x1, y1) and (x2, y2) specify the bounding box  
 \*/  
 void drawWedgeFilled(const Style &s, const int x1, const int y1, const int x2, const int y2, const double start, const double end);  
  
 /\*\*  
 \* Draw an elliptical wedge outline with from start to end degrees  
 \* The two points (x1, y1) and (x2, y2) specify the bounding box  
 \*/  
 void drawWedgeOutline(const Style &s, const int x1, const int y1, const int x2, const int y2, const double start, const double end);  
  
 /\*\*   
 \* Draw a filled elliptical chord with from start to end degrees  
 \* The two points (x1, y1) and (x2, y2) specify the bounding box  
 \*/  
 void drawChordFilled(const Style &s, const int x1, const int y1, const int x2, const int y2, const double start, const double end);  
  
 /\*\*  
 \* Draw an elliptical chord outline with from start to end degrees  
 \* The two points (x1, y1) and (x2, y2) specify the bounding box  
 \*/  
 void drawChordOutline(const Style &s, const int x1, const int y1, const int x2, const int y2, const double start, const double end);  
  
 /\*\*  
 \* Draw a filled polygon using the specified style and the two vectors to specifiy vertices  
 \*/  
 void drawPolygonFilled(const Style &s, const std::vector<int>& xcoords, const std::vector<int>& ycoords);  
  
 /\*\*  
 \* Draw a polygon outline using the specified style and the two vectors to specifiy vertices  
 \*/  
 void drawPolygonOutline(const Style &s, const std::vector<int>& xcoords, const std::vector<int>& ycoords);  
  
 /\*\*  
 \* Draw a bezier curve using the specified style, and the two vectors specifying the control points  
 \*/  
 void drawBezierCurve(const Style &s, const std::vector<int>& xcoords, const std::vector<int>& ycoords);  
  
 /\*\*  
 \* Draw the image at with its upper right-hand corner located at (x,y)  
 \* @see #drawImage(constImage&,constint,constint,constTransform&)  
 \* @see .Image  
 \*/  
 void drawImage(const Image &image, const int x, const int y);  
  
 /\*\*  
 \* Draw the image at with its upper right-hand corner located at (x,y) using the specified transform  
 \* @see #drawImage(constImage&,constint,constint)  
 \* @see .Image  
 \* @since 2.10  
 \*/  
 void drawImage(const Image &image, const int x, const int y,const Transform &transform);  
  
 /\*\*  
 \* Draw text in the specified font and style  
 \* @see #drawText(constStyle&,constFont&,constint,constint,conststring&,constTransform&)  
 \* @see #numberToString(int)  
 \* @see #numberToString(double)  
 \*/  
 void drawText(const Style &s, const Font &f, const int x, const int y, const std::string& text);  
  
 /\*\*  
 \* Draw text in the specified font and style, using the specified transform  
 \* @see #drawText(constStyle&,constFont&,constint,constint,conststring&,constTransform&)  
 \* @see #numberToString(int)  
 \* @see #numberToString(double)  
 \* @since 2.10  
 \*/  
 void drawText(const Style &s, const Font &f, const int x, const int y, const std::string& text,const Transform &transform);   
  
 /\*\*  
 \* Convert the given number to a string. Recommended for use with drawText.  
 \* @see #drawText(constStyle&,constFont&,constint,constint,conststring&,constTransform&)  
 \* @see #drawText(constStyle&,constFont&,constint,constint,conststring&,constTransform&)  
 \* @see #numberToString(double, int)  
 \* @since 2.1.4a  
 \*/  
 static std::string numberToString(int n);  
  
 /\*\*  
 \* Convert the given double number to a string. Recommended for use with drawText.  
 \* @param d [in] number to convert to string  
 \* @param precision [in, optional] number of digits of precision (numbers after decimal point)  
 \* @see #drawText(constStyle&,constFont&,constint,constint,conststring&,constTransform&)  
 \* @see #drawText(constStyle&,constFont&,constint,constint,conststring&,constTransform&)  
 \* @see #numberToString(int)  
 \* @since 2.1.4a  
 \*/  
 static std::string numberToString(double d, int precision = -1);  
  
 /\*\*  
 \* Create an image from the rectangular region with an upper-left hand corner at (x, y)  
 \* @see .Image  
 \* @see #drawImage  
 \*/  
 Image createImage(const int x, const int y, const int width, const int height);  
  
 /\*\*  
 \* Return the color at point (x, y)  
 \*/  
 Color readPixel(const int x, const int y);  
  
 /\*\*  
 \* Copy the rectangular region at (x1, y1) to (x2, y2)  
 \*/  
 void copyRegion(const int x1, const int y1, const int x2, const int y2, const int width, const int height);  
  
 /\*\*  
 \* Enabled by default; everything you draw will automatically appear on screen.   
 \* To get much better performance, use {@link #flipPage} and {@link #disableAutoPageFlip}.  
 \* @see #disableAutoPageFlip  
 \* @see #flipPage  
 \* @see #isAutoPageFlipEnabled  
 \*/  
 void enableAutoPageFlip();  
 /\*\*  
 \* By default each window will handle refreshing the graphics automatically  
 \* for the user. However in some circumstances it may be desirable to   
 \* perform these refreshes manually. First call {@link #disableAutoPageFlip}  
 \* to disable automatic refreshes and then {@link #flipPage} whenever you want to   
 \* show what you've drawn. {@link #enableAutoPageFlip} may then be used to enable automatic refreshes.  
 \* @see #enableAutoPageFlip  
 \* @see #flipPage  
 \* @see #isAutoPageFlipEnabled  
 \*/  
 void disableAutoPageFlip();  
  
 /\*\*  
 \* Make the changes that you've made appear on screen. All drawing is done off screen,   
 \* this will show the user the off-screen page.  
 \* @see #disableAutoPageFlip  
 \* @see #enableAutoPageFlip  
 \* @see #isAutoPageFlipEnabled  
 \*/  
 void flipPage();  
  
 /\*\*  
 \* Will return whether or not auto page flipping is enabled.  
 \* @see #disableAutoPageFlip  
 \* @see #enableAutoPageFlip  
 \* @see #flipPage  
 \* @since 2.1.5  
 \*/  
 bool isAutoPageFlipEnabled();  
  
 /\*\*  
 \* Event handlers for when in event handling mode. To be useful you must   
 \* derive your own version of Window that overides these functions  
 \* @see .WindowManager  
 \*/  
 virtual void handleKeyboardEvent(const KeyboardEvent& event);  
 /\*\*  
 \* Event handlers for when in event handling mode. To be useful you must   
 \* derive your own version of Window that overides these functions  
 \* @see .WindowManager  
 \*/  
 virtual void handleMouseEvent(const MouseEvent& event);  
 /\*\*  
 \* Event handlers for when in event handling mode. To be useful you must   
 \* derive your own version of Window that overides these functions  
 \* @see .WindowManager  
 \*/  
 virtual void handleIdleEvent();  
 /\*\*  
 \* Event handlers for when in event handling mode. To be useful you must   
 \* derive your own version of Window that overides these functions  
 \* @see .WindowManager  
 \*/  
 virtual void handleTimerEvent(const TimerEvent& event);  
  
 /\*\*   
 \* Renderer dependent capability. If not supported by current  
 \* renderer these will be ignored. Initial state is also dependent on   
 \* the supplied renderer.  
 \*/  
 void enableAntialiasing();  
 /\*\*   
 \* Renderer dependent capability. If not supported by current  
 \* renderer these will be ignored. Initial state is also dependent on   
 \* the supplied renderer.  
 \*/  
 void disableAntialiasing();  
   
 /\*\*   
 \* @deprecated Do not use. This works only on some systems. To ensure that you will always go into fullscreen mode, use the constructor {@link #Window}.  
 \*/  
 void enableFullScreen(int bitdepth=16,int refreshrate=60);  
 /\*\*   
 \* @deprecated Do not use. This works only on some systems. To ensure that you will always go into fullscreen mode, use the constructor {@link #Window}.  
 \*/  
 void disableFullScreen();  
  
 /\*\*  
 \* Set filters on the input queues, useful in eliminating non-interesting  
 \* events. Intended for advanced users only. If set to NULL, no filtering   
 \* is performed. If set to some value, only events that do not match the   
 \* event pattern will be placed into the queue. By default, when not using   
 \* the CarnegieMellonGraphics event handler, MOUSE\_MOVE events are filtered out to prevent   
 \* unnecessary events from cluttering the queue.   
 \* @see #setMouseFilter  
 \*/  
 MouseEvent getMouseFilter();  
 /\*\*  
 \* Set filters on the input queues, useful in eliminating non-interesting  
 \* events. Intended for advanced users only. If set to NULL, no filtering   
 \* is performed. If set to some value, only events that do not match the   
 \* event pattern will be placed into the queue. There is no default   
 \* keyboard filter.  
 \* @see #setKeyboardFilter  
 \*/  
 KeyboardEvent getKeyboardFilter();  
 /\*\*  
 \* Set filters on the input queues, useful in eliminating non-interesting  
 \* events. Intended for advanced users only. If set to NULL, no filtering   
 \* is performed. If set to some value, only events that do not match the   
 \* event pattern will be placed into the queue. There is no default   
 \* timer filter.  
 \* @see #setTimerFilter  
 \*/  
 TimerEvent getTimerFilter();  
 /\*\*  
 \* Change the filter for the mouse queue.   
 \* @see #getMouseFilter  
 \*/  
 void setMouseFilter(const MouseEvent &filter);  
 /\*\*  
 \* Change the filter for the keyboard queue.   
 \* @see #getKeyboardFilter  
 \*/  
 void setKeyboardFilter(const KeyboardEvent &filter);  
 /\*\*  
 \* Change the filter for the timer queue.   
 \* @see #getTimerFilter  
 \*/  
 void setTimerFilter(const TimerEvent &filter);  
  
 /\*\*   
 \* Changes the size of the window after creation.  
 \* @since 2.1.5  
 \* @see #getPosition  
 \* @see #setPosition  
 \* @see #setTitle  
 \*/  
 void resize(int newwidth, int newheight);  
  
 /\*\*  
 \* Changes the position of the window after creation.  
 \* @since 2.1.5  
 \* @see #getPosition  
 \* @see #resize  
 \* @see #setTitle  
 \*/  
 void setPosition(int x, int y);  
  
 /\*\*  
 \* Retrieves the current position of the window.  
 \* @since 2.1.5  
 \* @param x [out] x position of the window  
 \* @param y [out] y position of the window  
 \* @see #resize  
 \* @see #setTitle  
 \* @see #setPosition  
 \*/  
 void getPosition(int &x, int &y);  
  
 /\*\*  
 \* Changes the title of the window after creation.  
 \* @since 2.1.5  
 \* @see #getPosition  
 \* @see #setPosition  
 \* @see #resize  
 \*/  
 void setTitle(const std::string &title);  
  
  
  
/\* CPPDOC\_BEGIN\_EXCLUDE \*/  
 typedef long SafeWindowPtr;  
/\* CPPDOC\_END\_EXCLUDE \*/  
 private:  
 ////// Ignore these //////  
  
 void \_init(int xpos, int ypos, int width, int height, const std::string &title, bool fullscreen,int bitdepth,int refreshrate);  
  
 std::deque<MouseEvent\*> mousequeue;  
 std::deque<KeyboardEvent\*> keyqueue;  
 std::deque<TimerEvent\*> timerqueue;  
  
 // Prevent passing by value -- does not operate as desired  
 Window (const Window& window);  
  
 // Prevent assignment on windows because it is ill defined.  
 Window& operator=(const Window& rhs);   
  
 // Comparison on windows is also ill defined.  
 bool operator==(const Window& rhs) const;  
 bool operator!=(const Window& rhs) const;  
  
 void \_setWindowSize(int newheight, int newwidth);  
  
 int width, height;  
 int mousex, mousey;  
 MouseEvent mousefilter;  
 KeyboardEvent keyfilter;  
 TimerEvent timerfilter;  
 class PortMutex\* mousemutex;  
 class PortMutex\* keymutex;  
 class PortMutex\* timermutex;  
 class PortMutex\* sizemutex;  
  
 SemaphoreT<Color> \*colorsignal;  
 SemaphoreT<Image> \*imagesignal;  
 SemaphoreT<std::pair<int, int> > \*positionsignal;  
 bool hidden, m\_autoPageFlipStatus;  
 bool autoflushkeyboard,autoflushmouse,autoflushtimer;  
 class Semaphore \*mousesem,\*keyboardsem,\*timersem;  
  
 std::map<int,bool> key\_state;  
 std::map<MouseEvent::Button,bool> button\_state;  
 std::map<int,int> timer\_count;   
 KeyModifiers currentmodifiers;  
  
 class Daemon \*daemon;  
 friend class DaemonImp;  
 friend class RENDERER;  
  
};  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The FullScreenWindow class does everything a normal window does but its   
 \* constructor automatically makes the window full screen.   
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class FullScreenWindow : public Window {  
public:  
 FullScreenWindow(int width = 640, int height = 480, int bitdepth=32, int refreshrate=60);  
};  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The WindowManager class is used when operating the library in an event driven  
 \* mode. For this to do anything useful, you must first derive your own Window  
 \* class that overloads some of the eventhandlers. After that, all you  
 \* need to do is call WindowManager::handleEvents() somewhere in your code, and  
 \* it will begin event handling for you. To exit out of the handler, simply   
 \* call WindowManager::exitHandler() somewhere inside your derived Window class.  
 \* The event handler will clear our input filters on a window when started,  
 \* but will return them to their original state upon exiting.  
 \*  
 \* WindowManager also encapsulates some simple timing commands.  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
class WindowManager {  
public:  
 /\*\*  
 \* Called to begin event handling  
 \*/  
 static void handleEvents();  
  
 /\*\*  
 \* Called inside the event loop to exit the handle  
 \*/  
 static void exitHandler();  
  
 /\*\*  
 \* Return the number of "ticks" that have ellapsed since the  
 \* program was started. A "tick" is usually a microsecond (as it   
 \* depends on the platfrom the library was compiled).   
 \* This number will probably wrap around after about half an hour.  
 \*/  
 static unsigned long getTime();  
  
 private:  
 ////// Ignore these //////  
 static void registerWindow(Window \*window);  
 static void removeWindow(Window \*window);  
  
 // Not allowed to create instances of WindowManager.  
 WindowManager() { };  
  
 friend class Window;  
};  
  
///////////////////////////////////////////////////////////////////////////////  
  
#endif //\_\_CARNEGIEMELLONGRAPHICS\_H\_\_