NAME

World − root scene for a display

SYNOPSIS

#include <InterViews/world.h>

DESCRIPTION

A world is an application's root scene for a particular display. The world must be created before any other InterViews objects are created. Scene operations such as Insert, Raise, and Lower can be used to affect the display. Thus, an interactor is placed as a top-level window on the screen by inserting it into the world.

PUBLIC OPERATIONS

World(const char* classname, int& argc, char* argv[])

Construct the world object for a display. Top-level interactors that have not called SetClassName will use the world's class name when looking up default attributes. Top-level interactors that have not called SetInstance will use the world's instance name when looking up attributes. The world's instance name comes from the -name option if it was given; otherwise the instance name comes from the environment variable RESOURCE_NAME if it is nonnil. If the name has still not been defined, then argv[0] with all leading directories stripped is used. The following arguments are interpreted automatically and are removed from *argv*:

−background	next argument sets the background color
−bg	same as −background
−display	next argument specifies the target workstation display
−foreground	next argument sets the foreground color
−fg	same as −foreground
−fn	same as −font
−font	next argument sets the text font
−geometry	next argument sets the first top-level interactor's position and size
−iconic	starts up the first top-level interactor in iconic form
−name	next argument sets the instance name of all top-level interactors that don't have their own instance names
−reverse	swaps default foreground and background colors
−rv	same as −reverse
−title	next argument sets the first top-level interactor's title bar name
−xrm	next argument sets an ''attribute: value'' property

The geometry specification has the form ''=WxH+XOFF+YOFF''. A negative XOFF (YOFF) specifies the offset of the interactor's right (bottom) edge from the right (bottom) side of the screen. The constructor sets *argc* to the number of uninterpreted arguments that remain.

World(const char*, OptionDesc*, int&, char*[])

Construct a world as above, but using an array of option descriptors to parse application-specific attributes. OptionDesc contains four fields: a *name* used on the command line, a *path* specifying the attribute, a *style* specifying where the value is, and a default *value*. Valid styles are *OptionPropertyNext* (use next argument as an attribute:value pair, not just the value), *OptionValueNext* (use next argument as value), *OptionValueImplicit* (use default value), *OptionValueIsArg* (use argument as value), and *OptionValueAfter* (use remainder of argument as value).

**World(const char*, PropertyData*, OptionDesc*, int&,
char*[])**

Construct a world as above and specify a set of initial attribute values. PropertyData contains three string fields: a *path* specifying the attribute, a *value* specifying the value, and a *type* specifying the type name. Attributes are entered in the following order: first any application defaults (specified by the PropertyData array), then user defaults (usually in $HOME/.Xdefaults), and then command-line arguments. Thus, command-line arguments override user defaults, and both override application defaults.

World(const char* instance = nil, const char* display = nil)

Construct the world object for a specific display. This constructor is for backward compatibility and should no longer be used. Since the application cannot get its instance name for looking up resources from the command line, the first parameter specifies the instance name to use rather than the class name. If *display* is omitted, then the DISPLAY environment variable is used to connect to the appropriate window server. The format of the string is ‘‘hostname:N.S’’ where N is the display number and S is the screen number (S is optional).

void InsertPopup(Interactor*)
**void InsertPopup(Interactor*, Coord x, Coord y,
Alignment = BottomLeft)**

Insert a popup interactor into the world, optionally aligned with respect to a specific position relative to the lower-left corner of the screen, without any interaction whatsoever with window managers. A popup interactor inserted without a specific position will be centered on the screen, since the user cannot use a window manager’s help to place the interactor.

void InsertTransient(Interactor*, Interactor*)
**void InsertTransient(Interactor*, Interactor* owner,
Coord x, Coord y, Alignment = BottomLeft)**

Insert a transient interactor into the world on behalf of another interactor, optionally aligned with respect to a specific position relative to the lower-left corner of the screen. Most window managers will not decorate the transient interactor or ask the user to place it. Some window managers may also unmap the transient interactor when its *owner* interactor is iconized.

void InsertToplevel(Interactor*, Interactor*)
**void InsertToplevel(Interactor*, Interactor* group, Coord x,
Coord y, Alignment = BottomLeft)**

Insert a toplevel interactor into the world, optionally aligned with respect to a specific position relative to the lower-left corner of the screen. A nonnil *group* informs window managers that this toplevel interactor is part of a group of toplevel interactors led by *group*. Some window managers may unmap the toplevel interactor when its ‘‘leader’’ interactor is iconized.

void InsertApplication(Interactor*)
**void InsertApplication(Interactor*, Coord x, Coord y,
Alignment = BottomLeft)**

Insert an application interactor into the world, optionally aligned with respect to a specific position relative to the lower-left corner of the screen. Window and session managers will consider an application interactor to be the program’s main top-level interactor.

void InsertIcon(Interactor*)
**void InsertIcon(Interactor*, Coord x, Coord y,
Alignment = BottomLeft)**

Insert an icon interactor into the world, optionally aligned with respect to a specific position relative to the lower-left corner of the screen, but do not make it visible (‘‘map’’ it). Another top level interactor may call Interactor::SetIconInteractor to tell window managers to use this interactor for its icon. InsertIcon is called automatically when a top level interactor is inserted if the interactor has an icon and the icon has not yet been inserted into the world.

int Width()
int Height()

Return the width and height in pixels of the screen.

int NPlanes()

Return the number of user-accessible bit planes the workstation display has.

int NButtons()

Return the number of buttons on the input pointing device associated with the world.

int PixelSize()

Return the natural size in bits for storing a pixel.

unsigned ParseGeometry(const char* spec, Coord& x, Coord& y,
int& w, int& h)

Interpret a geometry specification (e.g., =80x24+1+1), returning a bit mask that identifies which fields were specified.

void SetHint(const char*)

Provide window system specific information. See next section.

void RingBell(int vol)

Ring the workstation bell with a volume equal to a percentage of the possible range, with 0 the quietest and 100 the loudest.

void SetKeyClick(int vol)

Set the volume of the audible keyclick equal to a percentage of the possible range, with 0 the quietest and 100 the loudest.

void SetAutoRepeat(boolean on)

Turn on or off autorepeat for the keyboard.

void SetFeedback(int threshold, int scale)

Set the amount of magnification to give the user's movements with the pointing device. The *threshold* is the minimum distance after which to scale the movement and the *scale* is the factor by which to magnify the movement.

void SetCurrent()

Set the current world. Some routines use global state associated with a world object. Creating a world automatically sets the state to that of the new world.

int Fileno()

Return the file descriptor of the connection to the window server.

WINDOW SYSTEM HINTS

The performance of some InterViews library functions depends heavily on the characteristics of the underlying window system. Window system hints provide a way for the programmer or user to provide information to maximize performance. There is usually no need to provide hints: the default behavior is appropriate for most installations. By convention, a hint consists of a name and a value, separated by a colon. Meaningful names and values are implementation-specific. Non-meaningful hints are simply ignored. Currently defined hints include: **RubberbandPixel**

The value is a hexadecimal number which is used to control the drawing of rubber bands. On many window systems, rubber bands are drawn in XOR mode so that they can easily be erased without damaging underlying screen contents. On a color workstation, the resulting color depends on the screen contents and the XOR mask pixel. The mask should be chosed to maximize the visibility of the rubber band, but the best value may depend on the workstation color rendering model and on how an application uses colors.

TransformFonts

The value is one of ''off'', ''on'', or ''cache''. On many window systems, drawing text which has been rotated or scaled is much slower than drawing untransformed text. This hint suggests a strategy for optimizing text transformations. ''Off'' suggests that fonts should not be transformed. This usually results in text drawn with the origins of the characters at the correct positions but with untransformed character glyphs. ''On'' suggests that the character glyphs should be transformed. ''Cache'' further suggests that transformed glyphs be cached to speed up future text operations that might require the same glyph. Cached glyphs consume both application and window system resources but are drawn much faster.

Dash

The value is one of ''none'', ''thin'', or ''all''. Some window systems draw dashed lines slowly, particularly if the line is not of minimal thickness. InterViews can simulate dashed lines with stipple patterns, but the results may be less that satisfactory. This hint suggest a strategy for dashed lines. ''None'' suggests that all dashed lines should be simulated with stipple patterns; ''thin'' suggests that minimal-width lines should be dashed, but that thicker dashed lines should be simulated; ''all'' suggests that lines of all thicknesses should be dashed.

Window system hints can be specified through the function *World::SetHint* or as application attributes. *SetHint* allows an application to specify hints dynamically. For example, an application that uses transformed text might temporarily disable font transformations to speed up drawing of a rough draft and re-enable transformations to show the final version. Attributes allow users to control the default values of hints from their user preference files or from the application command line. Attribute names are derived from the corresponding hint names by appending the string ''Hint''. For example, the RubberbandPixel hint can be set through the ''RubberbandPixelHint'' attribute.

SEE ALSO

Interactor , Scene