Manual for wxTreeLayout 2.0: a tree layout library for wxWindows

Julian Smart
Anthemion Software

July 1998

Contents

1. Introduction

This manual describes a tree-drawing class library for wxWindows. It provides layout of simple trees with one root node, drawn left-to-right, with user-defined spacing between nodes.

wxTreeLayout is an abstract class that must be subclassed. The programmer defines various member functions which will access whatever data structures are appropriate for the application, and wxTreeLayout uses these when laying out the tree.

wxStoredTree is a class derived from wxTreeLayout that may be used directly to draw trees on a canvas. It supplies storage for the nodes, and draws to a device context.

Below is the example tree generated by the program test.cc.

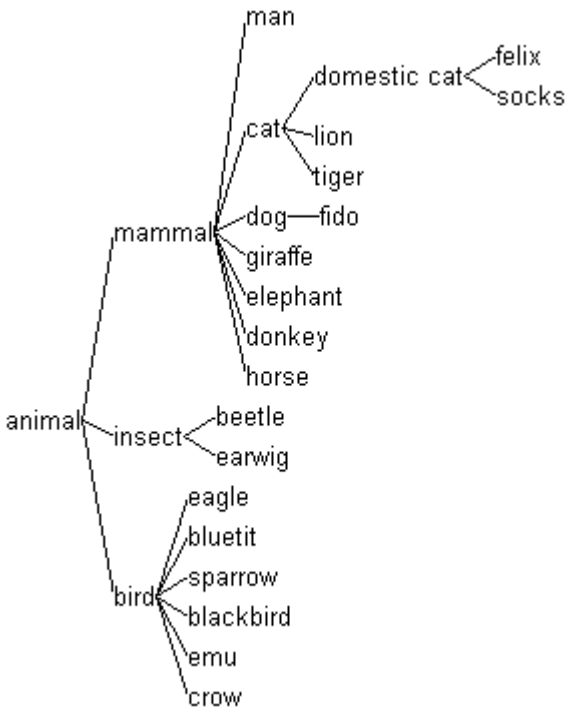

Figure 1: Example tree

2. Implementation

The algorithm is due to Gabriel Robins [1], a linear-time algorithm originally implemented in LISP for AI applications.

The original algorithm has been modified so that both X and Y planes are calculated simultaneously, increasing efficiency slightly. The basic code is only a page or so long.

3. wxTreeLayout Class Reference

3.1. wxTreeLayout

This abstract class is used for drawing a tree. You must derive a new class from this, and define member functions to access the data that wxTreeLayout needs.

Nodes are identified by long integer identifiers. The derived class communicates the actual tree structure to wxTreeLayout by defining *wxTreeLayout::GetChildren* and *wxTreeLayout::GetNodeParent* functions.

The application should call *DoLayout* to do the tree layout. Depending on how the derived class has been defined, either *wxTreeLayout::Draw* must be called (for example by the OnPaint member of a wxScrolledWindow) or the application-defined drawing code should be called as normal.

For example, if you have an image drawing system already defined, you may want wxTreeLayout to position existing node images in that system. So you just need a way for wxTreeLayout to set the node image positions according to the layout algorithm, and the rest will be done by your own image drawing system.

Derived from

wxObject

wxTreeLayout::wxTreeLayout

wxTreeLayout()

Constructor.

wxTreeLayout::ActivateNode

void ActivateNode(**long** *id*, **bool** *active*)

Define this so wxTreeLayout can turn nodes on and off for drawing purposes (not all nodes may be connected in the tree). See also *NodeActive*.

wxTreeLayout::CalcLayout

void CalcLayout(**long** *id*, **int** *level*)

Private function for laying out a branch.

wxTreeLayout::DoLayout

void DoLayout(**wxDC&** *dc*, **long** *topNode = -1*)

Calculates the layout for the tree, optionally specifying the top node.

wxTreeLayout::Draw

void Draw(**wxDC&** *dc*)

Call this to let wxTreeLayout draw the tree itself, once the layout has been calculated with *DoLayout*.

wxTreeLayout::DrawBranch

void DrawBranch(**long** *from*, **long** *to*, **wxDC&** *dc*)

Defined by wxTreeLayout to draw an arc between two nodes.

wxTreeLayout::DrawBranches

void DrawBranches(**wxDC&** *dc*)

Defined by wxTreeLayout to draw the arcs between nodes.

wxTreeLayout::DrawNode

void DrawNode(**long** *id*, **wxDC&** *dc*)

Defined by wxTreeLayout to draw a node.

wxTreeLayout::DrawNodes

void DrawNodes(**wxDC&** *dc*)

Defined by wxTreeLayout to draw the nodes.

wxTreeLayout::GetChildren

void GetChildren(**long** *id*, **wxList &***list*)

Must be defined to return the children of node *id* in the given list of integers.

wxTreeLayout::GetNextNode

long GetNextNode(**long** *id*)

Must be defined to return the next node after *id*, so that wxTreeLayout can iterate through all relevant nodes. The ordering is not important. The function should return -1 if there are no more nodes.

wxTreeLayout::GetNodeName

wxString GetNodeName(**long** *id*) **const**

May optionally be defined to get a node's name (for example if leaving the drawing to wxTreeLayout).

wxTreeLayout::GetNodeSize

void GetNodeSize(**long** *id*, **long*** *x*, **long*** *y*) **const**

Can be defined to indicate a node's size, or left to wxTreeLayout to use the name as an indication of size.

wxTreeLayout::GetNodeParent

long GetNodeParent(**long** *id*) **const**

Must be defined to return the parent node of *id*. The function should return -1 if there is no parent.

wxTreeLayout::GetNodeX

long GetNodeX(**long** *id*) **const**

Must be defined to return the current X position of the node. Note that coordinates are assumed to be at the top-left of the node so some conversion may be necessary for your application.

wxTreeLayout::GetNodeY

long GetNodeY(**long** *id*) **const**

Must be defined to return the current Y position of the node. Note that coordinates are assumed to be at the top-left of the node so some conversion may be necessary for your application.

wxTreeLayout::GetLeftMargin

long GetLeftMargin() **const**

Gets the left margin set with *SetMargins*.

wxTreeLayout::GetOrientation

bool GetOrientation() **const**

Gets the orientation: TRUE means top-to-bottom, FALSE means left-to-right (the default).

wxTreeLayout::GetTopMargin

long GetTopMargin() **const**

Gets the top margin set with *SetMargins*.

wxTreeLayout::GetTopNode

long GetTopNode() **const**

wxTreeLayout calls this to get the top of the tree. Don't redefine this; call *SetTopNode* instead before calling *DoLayout*.

wxTreeLayout::GetXSpacing

long GetXSpacing() **const**

Gets the horizontal spacing between nodes.

wxTreeLayout::GetYSpacing

long GetYSpacing() **const**

Gets the vertical spacing between nodes.

wxTreeLayout::Initialize

void Initialize()

Initializes wxTreeLayout. Call from application or overridden **Initialize**or constructor.

wxTreeLayout::NodeActive

bool NodeActive(**long** *id*)

Define this so wxTreeLayout can know which nodes are to be drawn (not all nodes may be connected in the tree). See also *ActivateNode*.

wxTreeLayout::SetNodeName

void SetNodeName(**long** *id*, **const wxString&** *name*)

May optionally be defined to set a node's name.

wxTreeLayout::SetNodeX

void SetNodeX(**long** *id*, **long** *x*)

Must be defined to set the current X position of the node. Note that coordinates are assumed to be at the top-left of the node so some conversion may be necessary for your application.

wxTreeLayout::SetNodeY

void SetNodeY(**long** *id*, **long** *y*)

Must be defined to set the current Y position of the node. Note that coordinates are assumed to be at the top-left of the node so some conversion may be necessary for your application.

wxTreeLayout::SetOrientation

void SetOrientation(**bool** *orientation*)

Sets the tree orientation: TRUE means top-to-bottom, FALSE means left-to-right (the default).

wxTreeLayout::SetTopNode

void SetTopNode(**long** *id*)

Call this to identify the top of the tree to wxTreeLayout.

wxTreeLayout::SetSpacing

void SetSpacing(**long** *x*, **long** *y*)

Sets the horizontal and vertical spacing between nodes in the tree.

wxTreeLayout::SetMargins

void SetMargins(**long** *x*, **long** *y*)

Sets the left and top margins of the whole tree.

3.2. wxStoredTree

wxStoredTree provides storage for node labels, position and client data. It also provides hit-testing (which node a mouse event occurred on). It is usually a more convenient class to use than wxTreeLayout.

Derived from

wxTreeLayout (page 3)

wxStoredTree::wxStoredTree

wxStoredTree(**int** *noNodes = 200*)

Constructor. Specify the maximum number of nodes to be allocated.

wxStoredTree::AddChild

long AddChild(**const wxString&** *name*, **const wxString&** *parent = ""*)

Adds a child with a given parent, returning the node id.

wxStoredTree::GetClientData

long GetClientData(**long** *id*) **const**

Gets the client data for the given node.

wxStoredTree::GetNode

wxStoredNode* GetNode(**long** *id*) **const**

Returns the wxStoredNode object for the given node id.

wxStoredTree::GetNodeCount

int GetNodeCount() **const**

Returns the current number of nodes.

wxStoredTree::GetNumNodes

int GetNumNodes() **const**

Returns the maximum number of nodes.

wxStoredTree::HitTest

wxString HitTest(**wxMouseEvent&** *event*, **wxDC&** *dc*)

Returns a string with the node name corresponding to the position of the mouse event, or the empty string if no node was detected.

wxStoredTree::NameToId

long NameToId(**const wxString&** *name*)

Returns the id for the given node name, or -1 if there was no such node.

wxStoredTree::SetClientData

void SetClientData(**long** *id*, **long** *clientData*)

Sets client data for the given node.

References

[1] **Robins, Gabriel**. 1987 (September). *The ISI grapher: a portable tool for displaying graphs pictorially (ISI/RS-87-196).* Technical report. University of South California.

Index