User Manual for wxWindows Property Sheet Classes Version 1.1

Julian Smart
Artificial Intelligence Applications Institute
University of Edinburgh
EH1 1HN

November 1995

Contents

Copyright notice

1. Introduction

The Property Sheet Classes help the programmer to specify complex dialogs and their relationship with their associated data. By specifying data as a wxPropertySheet containing wxProperty objects, the programmer can use a range of available or custom wxPropertyView classes to allow the user to edit this data. Classes derived from wxPropertyView act as mediators between the wxPropertySheet and the actual window (and associated panel items).

For example, the wxPropertyListView is a kind of wxPropertyView which displays data in a Visual Basic-style property list (see *the next section* (page 1) for screen shots). This is a listbox containing names and values, with an edit control and other optional controls via which the user edits the selected data item.

wxPropertyFormView is another kind of wxPropertyView which mediates between the data and a panel or dialog box which has already been created. This makes it a contender for the replacement of wxForm, since programmer-controlled layout is going to be much more satisfactory. If automatic layout is desired, then wxPropertyListView could be used instead.

The main intention of this class library was to provide property *list* behaviour, but it has been generalised as much as possible so that the concept of a property sheet and its viewers can reduce programming effort in a range of user interface tasks.

For further details on the classes and how they are used, please see *Property classes overview* (page 33).

1.1. The appearance and behaviour of a property list view

The property list, as seen in an increasing number of development tools such as Visual Basic and Delphi, is a convenient and compact method for displaying and editing a number of items without the need for one control per item, and without the need for designing a special form. The controls are as follows:

- A listbox showing the properties and their current values, which has double-click properties dependent on the nature of the current property;
- a text editing area at the top of the display, allowing the user to edit the currently selected property if appropriate;
- 'confirm' and 'cancel' buttons to confirm or cancel an edit (for the property, not the whole sheet);
- an optional list that appears when the user can make a choice from several known possible values;
- a small Edit button to invoke 'detailed editing' (perhaps showing or hiding the above value list, or maybe invoking a common dialog);
- optional OK/Close, Cancel and Help buttons for the whole dialog.

The concept of 'detailed editing' versus quick editing gives the user a choice of editing mode, so novice and expert behaviour can be catered for, or the user can just use what he feels comfortable with.

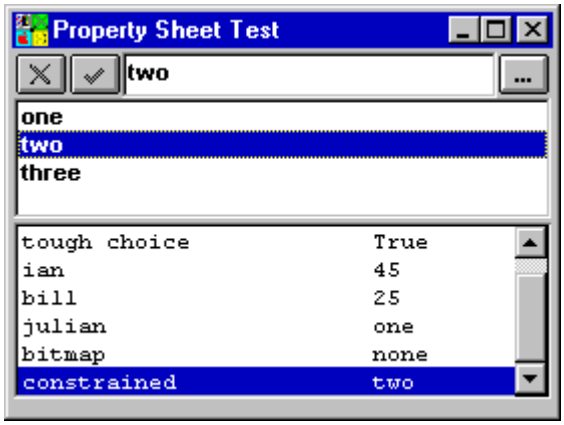
Behaviour alters depending on the kind of property being edited. For example, a boolean value has the following behaviour:

- Double-clicking on the item toggles between TRUE and FALSE.
- Showing the value list enables the user to select TRUE or FALSE.
- The user may be able to type in the word TRUE or FALSE, or the edit control may be read-only to disallow this since it is error-prone.

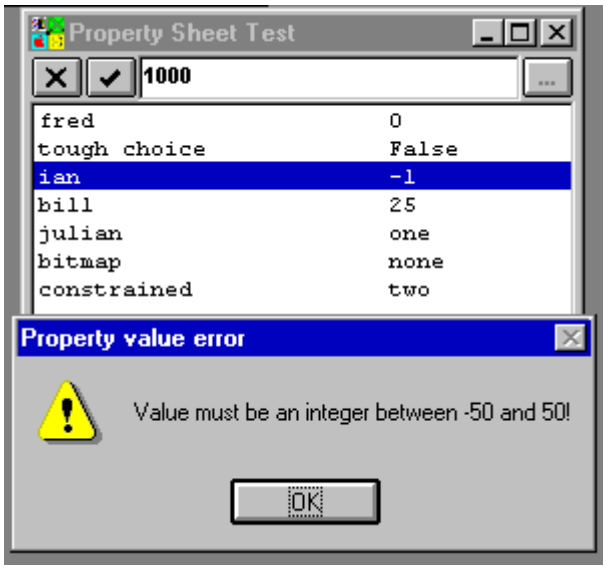
A list of strings may pop up a dialog for editing them, a simple string just allows text editing, double-clicking a colour property may show a colour selector, double-clicking on a filename property may show a file selector (in addition to being able to type in the name in the edit control), etc.

Note that the 'type' of property, such as string or integer, does not necessarily determine the behaviour of the property. The programmer has to be able to specify different behaviours for the same type, depending on the meaning of the property. For example, a colour and a filename may both be strings, but their editing behaviour should be different. This is why objects of type wxPropertyValidator need to be used, to define behaviour for a given class of properties or even specific property name. Objects of class wxPropertyView contain a list of property registries, which enable reuse of bunches of these validators in different circumstances. Or a wxProperty can be explicitly set to use a particular validator object.

The following screen shot of the property classes test program shows the user editing a string, which is constrained to be one of three possible values.



The second picture shows the user having entered a integer that was outside the range specified to the validator. Note that in this picture, the value list is hidden because it is not used when editing an integer.


Property Sheet Test
1000
fred 0
tough choice False
ian -1
bill 25
julian one
bitmap none
constrained two
Property value error
Value must be an integer between -50 and 50!
OK

2. Files

The property class library comprises the following files:

- wx_prop.h: base property class header
- wx_plist.h: wxPropertyListView and associated classes
- wx_pform.h: wxPropertyListView and associated classes
- wx_prop.cc: base property class implementation
- wx_plist.cc: wxPropertyListView and associated class implementions
- wx_pform.cc: wxPropertyFormView and associated class implementions

3. Alphabetical class reference

See also *Property classes overview* (page 33)

3.1. wxBoolFormValidator: wxPropertyFormValidator

See also *Validator classes* (page 31)

This class validates a boolean value for a form view. The associated panel item must be a wxCheckBox.

wxBoolFormValidator::wxBoolFormValidator

void wxBoolFormValidator(**long** *flags=0*)

Constructor.

3.2. wxBoolListValidator: wxPropertyListValidator

See also *Validator classes* (page 31)

This class validates a boolean value for a list view.

wxBoolListValidator::wxBoolListValidator

void wxBoolListValidator(**long** *flags=0*)

Constructor.

3.3. wxIntegerFormValidator: wxPropertyFormValidator

See also *Validator classes* (page 31)

This class validates a range of integer values for a form view. The associated panel item must be a wxText or wxSlider.

wxIntegerFormValidator::wxIntegerFormValidator

void wxIntegerFormValidator(**long** *min=0*, **long** *max=0*, **long** *flags=0*)

Constructor. Assigning zero to minimum and maximum values indicates that there is no range to check.

3.4. wxIntegerListValidator: wxPropertyListValidator

See also *Validator classes* (page 31)

This class validates a range of integer values for a list view.

wxIntegerListValidator::wxIntegerListValidator

void wxIntegerListValidator(**long** *min=0*, **long** *max=0*, **long** *flags=wxPROP_ALLOW_TEXT_EDITING*)

Constructor. Assigning zero to minimum and maximum values indicates that there is no range to check.

3.5. wxFilenameListValidator: wxPropertyListValidator

See also *Validator classes* (page 31)

This class validates a filename for a list view, allowing the user to edit it textually and also popping up a file selector in "detailed editing" mode.

wxFilenameListValidator::wxFilenameListValidator

void wxFilenameListValidator(**wxString** *message = "Select a file"*, **wxString** *wildcard = "*.*"*, **long** *flags=0*)

Constructor. Supply an optional message and wildcard.

3.6. wxListOfStringsListValidator: wxPropertyListValidator

See also *Validator classes* (page 31)

This class validates a list of strings for a list view. When editing the property, a dialog box is presented for adding, deleting or editing entries in the list. At present no constraints may be supplied.

You can construct a string list property value by constructing a wxStringList object.

For example:

```
  myListValidatorRegistry.RegisterValidator((wxString)"stringlist",
      new wxListOfStringsListValidator);

  wxStringList *strings = new wxStringList("earth", "fire", "wind",
"water", NULL);

  sheet->AddProperty(new wxProperty("fred", strings, "stringlist"));
```

wxListOfStringsListValidator::wxListofStringsListValidator

void wxListOfStringsListValidator(**long** *flags=0*)

Constructor.

3.7. wxProperty: wxObject

The **wxProperty** class represents a property, with a *wxPropertyValue* (page 22) containing the actual value, a name a role, an optional validator, and an optional associated window.

A property might correspond to an actual C++ data member, or it might correspond to a

conceptual property, such as the width of a window. There is no explicit data member *wxWindow::width*, but it may be convenient to invent such a property for the purposes of editing attributes of the window. The properties in the property sheet can be mapped to "reality" by whatever means (in this case by calling wxWindow::SetSize when the user has finished editing the property sheet).

A validator may be associated with the property in order to ensure that this and only this validator will be used for editing and validating the property. An alternative method is to use the *role* parameter to specify what kind of validator would be appropriate; for example, specifying "filename" for the role would allow the property view to find an appropriate validator at edit time.

wxProperty::wxProperty

void wxProperty(**void**)

void wxProperty(**wxProperty&** *prop*)

void wxProperty(**wxString** *name*, **wxString** *role*, **wxPropertyValidator ***validator=NULL*)

void wxProperty(**wxString** *name*, **const wxPropertyValue&** *val*, **wxString** *role*, **wxPropertyValidator ***validator=NULL*)

Constructors.

wxProperty::~wxProperty

void ~wxProperty(**void**)

Destructor. Destroys the wxPropertyValue, and the property validator if there is one. However, if the actual C++ value in the wxPropertyValue is a pointer, the data in that variable is not destroyed.

wxProperty::GetValue

wxPropertyValue& GetValue(**void**)

Returns a reference to the property value.

wxProperty::GetValidator

wxPropertyValidator * GetValidator(**void**)

Returns a pointer to the associated property validator (if any).

wxProperty::GetName

wxString& GetName(**void**)

Returns the name of the property.

wxProperty::GetRole

wxRole& GetRole(**void**)

Returns the role of the property, to be used when choosing an appropriate validator.

wxProperty::GetWindow

wxWindow * GetWindow(**void**)

Returns the window associated with the property (if any).

wxProperty::SetValue

void SetValue(**wxPropertyValue&** *val*)

Sets the value of the property.

wxProperty::SetName

void SetName(**wxString&** *name*)

Sets the name of the property.

wxProperty::SetRole

void SetRole(**wxString&** *role*)

Sets the role of the property.

wxProperty::SetValidator

void SetValidator(**wxPropertyValidator ***validator*)

Sets the validator: this will be deleted when the property is deleted.

wxProperty::SetWindow

void SetWindow(**wxWindow ***win*)

Sets the window associated with the property.

wxProperty::operator =

void operator =(**const wxPropertyValue&** *val*)

Assignment operator.

3.8. wxPropertyFormValidator: wxPropertyValidator

The **wxPropertyFormValidator** abstract class is the root of classes that define validation for a wxPropertyFormView.

3.9. wxPropertyFormDialog: wxDialogBox

The **wxPropertyFormDialog** class is a prepackaged dialog which can be used for viewing a form property sheet. Pass a property form view object, and the dialog will pass OnClose and OnDefaultAction listbox messages to the view class for processing.

wxPropertyFormDialog::wxPropertyFormDialog

void wxPropertyFormDialog(**wxPropertyFormView ***_view_, **wxWindow ***_parent_, **char ***_title_, **Bool** _modal=FALSE_, **int** _x=-1_, **int** _y=-1_, **int** _width=-1_, **int**_height=-1_, **long** _style=wxDEFAULT_DIALOG_STYLE_, **char ***_name="dialogBox"_)

Constructor.

wxPropertyFormDialog::~wxPropertyFormDialog

void ~wxPropertyFormDialog(**void**)

Destructor.

3.10. wxPropertyFormFrame: wxFrame

The **wxPropertyFormFrame** class is a prepackaged frame which can be used for viewing a property form. Pass a property form view object, and the frame will pass OnClose messages to the view class for processing.

Call Initialize to create the panel and associate the view; override OnCreatePanel if you wish to use a panel class other than the default wxPropertyFormPanel.

wxPropertyFormFrame::wxPropertyFormFrame

void wxPropertyFormFrame(**wxPropertyFormView ***_view_, **wxFrame ***_parent_, **char ***_title_, **int** _x=-1_, **int** _y=-1_, **int** _width=-1_, **int**_height=-1_, **long** _style=wxSDI_ | _wxDEFAULT_FRAME_, **char ***_name="frame"_)

Constructor.

wxPropertyFormFrame::~wxPropertyFormFrame

void ~wxPropertyFormFrame(**void**)

Destructor.

wxPropertyFormFrame::GetPropertyPanel

wxPanel * GetPropertyPanel(**void**)

Returns the panel associated with the frame.

wxPropertyFormFrame::Initialize

Bool Initialize(**void**)

Must be called to create the panel and associate the view with the panel and frame.

wxPropertyFormFrame::OnCreatePanel

wxPanel * OnCreatePanel(**wxFrame ***parent*, **wxPropertyFormView ***view*)

Creates a panel. Override this to create a panel type other than wxPropertyFormPanel.

3.11. wxPropertyFormPanel: wxPanel

The **wxPropertyFormPanel** class is a prepackaged panel which can be used for viewing a property form. Pass a property form view object, and the panel will pass OnDefaultAction listbox messages to the view class for processing.

wxPropertyFormPanel::wxPropertyFormPanel

void wxPropertyFormPanel(**wxPropertyFormView ***view*, **wxWindow ***parent*, **int** *x=-1*, **int** *y=-1*, **int** *width=-1*, **int***height=-1*, **long** *style=0*, **char ***name="panel"*)

Constructor.

wxPropertyFormPanel::~wxPropertyFormPanel

void ~wxPropertyFormPanel(**void**)

Destructor.

3.12. wxPropertyFormValidator: wxPropertyValidator

See also *wxPropertyFormValidator overview* (page 38)

The **wxPropertyFormValidator** class defines a base class for form validators. By overriding virtual functions, the programmer can create custom behaviour for kinds of property.

wxPropertyFormValidator::wxPropertyFormValidator

void wxPropertyFormValidator(**long** *flags = 0*)

Constructor.

wxPropertyFormValidator::~wxPropertyFormValidator

void ~wxPropertyFormValidator(**void**)

Destructor.

wxPropertyFormValidator::OnCommand

Bool OnCommand(**wxProperty ***_property_, **wxPropertyFormView ***_view_, **wxWindow ***_parentWindow_, **wxCommandEvent&** _event_)

Called when the control corresponding to the property receives a command (if not intercepted by a callback associated with the actual control).

wxPropertyFormValidator::OnCheckValue

Bool OnCheckValue(**wxProperty ***_property_, **wxPropertyFormView ***_view_, **wxWindow ***_parentWindow_) Called when the view checks the property value. The value checked by this validator should be taken from the panel item corresponding to the property.

wxPropertyFormValidator::OnDisplayValue

Bool OnDisplayValue(**wxProperty ***_property_, **wxPropertyFormView ***_view_, **wxWindow ***_parentWindow_)

Should display the property value in the appropriate control.

wxPropertyFormValidator::OnDoubleClick

Bool OnDoubleClick(**wxProperty ***_property_, **wxPropertyFormView ***_view_, **wxWindow ***_parentWindow_)

Called when the control corresponding to the property is double clicked (listboxes only).

wxPropertyFormValidator::OnRetrieveValue

Bool OnRetrieveValue(**wxProperty ***_property_, **wxPropertyFormView ***_view_, **wxWindow ***_parentWindow_)

Should do the transfer from the property editing area to the property itself.

3.13. wxPropertyFormView: wxPropertyView

See also *wxPropertyFormView overview* (page 38)

The **wxPropertyFormView** class shows a wxPropertySheet as a view onto a panel or dialog box which has already been created.

wxPropertyFormView::wxPropertyFormView

void wxPropertyFormView(**long** *flags = 0*)

Constructor.

wxPropertyFormView::~wxPropertyFormView

void ~wxPropertyFormView(**void**)

Destructor.

wxPropertyFormView::AssociateNames

void AssociateNames(**void**)

Associates the properties with the controls on the panel. For each panel item, if the panel item name is the same as a property name, the two objects will be associated. This function should be called manually since the programmer may wish to do the association manually.

wxPropertyFormView::Check

Bool Check(**void**)

Checks all properties by calling the appropriate validators; returns FALSE if a validation failed.

wxPropertyFormView::GetPanel

wxPanel * GetPanel(**void**)

Returns the panel associated with the view.

wxPropertyFormView::GetManagedWindow

wxWindow * GetManagedWindow(**void**)

Returns the managed window (a frame or dialog) associated with the view.

wxPropertyFormView::OnOk

void OnOk(**void**)

Virtual function that will be called when the OK button on the physical window is pressed. By default, checks and updates the form values, closes and deletes the frame or dialog, then deletes the view.

wxPropertyFormView::OnCancel

void OnCancel(**void**)

Virtual function that will be called when the Cancel button on the physical window is pressed. By default, closes and deletes the frame or dialog, then deletes the view.

wxPropertyFormView::OnHelp

void OnHelp(**void**)

Virtual function that will be called when the Help button on the physical window is pressed. This needs to be overridden by the application for anything interesting to happen.

wxPropertyFormView::OnRevert

void OnRevert(**void**)

Virtual function that will be called when the Revert button on the physical window is pressed. By default transfers the wxProperty values to the panel items (in effect undoing any unsaved changes in the items).

wxPropertyFormView::OnUpdate

void OnUpdate(**void**)

Virtual function that will be called when the Update button on the physical window is pressed. By defaults transfers the displayed values to the wxProperty objects.

wxPropertyFormView::SetManagedWindow

void SetManagedWindow(**wxWindow ***_win_)

Sets the managed window (a frame or dialog) associated with the view.

wxPropertyFormView::TransferToDialog

Bool TransferToDialog(**void**)

Transfers property values to the controls in the dialog.

wxPropertyFormView::TransferToPropertySheet

Bool TransferToPropertySheet(**void**)

Transfers property values from the controls in the dialog to the property sheet.

3.14. wxPropertyListDialog: wxDialogBox

The **wxPropertyListDialog** class is a prepackaged dialog which can be used for viewing a property list. Pass a property list view object, and the dialog will pass OnClose and OnDefaultAction listbox messages to the view class for processing.

wxPropertyListDialog::wxPropertyListDialog

void wxPropertyListDialog(**wxPropertyListView ***_view_, **wxWindow ***_parent_, **char ***_title_, **Bool** _modal=FALSE_, **int** _x=-1_, **int** _y=-1_, **int** _width=-1_, **int**_height=-1_, **long** _style=wxDEFAULT_DIALOG_STYLE_, **char ***_name="dialogBox"_)

Constructor.

wxPropertyListDialog::~wxPropertyListDialog

void ~wxPropertyListDialog(**void**)

Destructor.

3.15. wxPropertyListFrame: wxFrame

The **wxPropertyListFrame** class is a prepackaged frame which can be used for viewing a property list. Pass a property list view object, and the frame will pass OnClose messages to the view class for processing.

Call Initialize to create the panel and associate the view; override OnCreatePanel if you wish to use a panel class other than the default wxPropertyListPanel.

wxPropertyListFrame::wxPropertyListFrame

void wxPropertyListFrame(**wxPropertyListView ***_view_, **wxFrame ***_parent_, **char ***_title_, **int** _x=-1_, **int** _y=-1_, **int** _width=-1_, **int**_height=-1_, **long** _style=wxSDI_ | _wxDEFAULT_FRAME_, **char ***_name="frame"_)

Constructor.

wxPropertyListFrame::~wxPropertyListFrame

void ~wxPropertyListFrame(**void**)

Destructor.

wxPropertyListFrame::GetPropertyPanel

wxPanel * GetPropertyPanel(**void**)

Returns the panel associated with the frame.

wxPropertyListFrame::Initialize

Bool Initialize(**void**)

Must be called to create the panel and associate the view with the panel and frame.

wxPropertyListFrame::OnCreatePanel

wxPanel * OnCreatePanel(**wxFrame ***parent*, **wxPropertyListView ***view*)

Creates a panel. Override this to create a panel type other than wxPropertyListPanel.

3.16. wxPropertyListPanel: wxPanel

The **wxPropertyListPanel** class is a prepackaged panel which can be used for viewing a property list. Pass a property list view object, and the panel will pass OnDefaultAction listbox messages to the view class for processing.

wxPropertyListPanel::wxPropertyListPanel

void wxPropertyListPanel(**wxPropertyListView ***view*, **wxWindow ***parent*, **int** *x=-1*, **int** *y=-1*, **int** *width=-1*, **int***height=-1*, **long** *style=0*, **char ***name="panel"*)

Constructor.

wxPropertyListPanel::~wxPropertyListPanel

void ~wxPropertyListPanel(**void**)

Destructor.

3.17. wxPropertyListValidator: wxPropertyValidator

See also *wxPropertyListValidator overview* (page 37)

The **wxPropertyListValidator** abstract class is the base class for deriving validators for property lists.

wxPropertyListValidator::wxPropertyListValidator

void wxPropertyListValidator(**long** *flags = wxPROP_ALLOW_TEXT_EDITING*)

Constructor.

wxPropertyListValidator::~wxPropertyListValidator

void ~wxPropertyListValidator(**void**)

Destructor.

wxPropertyListValidator::OnCheckValue

Bool OnCheckValue(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_) Called when the Tick (Confirm) button is pressed or focus is list. Return FALSE if the value was invalid, which is a signal restores the old value. Return TRUE if the value was valid.

wxPropertyListValidator::OnClearControls

Bool OnClearControls(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_) Allows the clearing (enabling, disabling) of property list controls, when the focus leaves the current property.

wxPropertyListValidator::OnClearDetailControls

Bool OnClearDetailControls(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_) Called when the focus is lost, if the validator is in detailed editing mode.

wxPropertyListValidator::OnDisplayValue

Bool OnDisplayValue(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_)

Should display the value in the appropriate controls. The default implementation gets the textual value from the property and inserts it into the text edit control.

wxPropertyListValidator::OnDoubleClick

Bool OnDoubleClick(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_)

Called when the property is double clicked. Extra functionality can be provided, such as cycling through possible values.

wxPropertyListValidator::OnEdit

Bool OnEdit(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_) Called when the Edit (detailed editing) button is pressed. The default implementation calls wxPropertyListView::BeginDetailedEditing; a filename validator (for example) overrides this function to show the file selector.

wxPropertyListValidator::OnPrepareControls

Bool OnPrepareControls(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_)

Called to allow the validator to setup the display, such enabling or disabling buttons, and setting the values and selection in the standard listbox control (the one optionally used for displaying

value options).

wxPropertyListValidator::OnPrepareDetailControls

Bool OnPrepareDetailControls(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_) Called when the property is edited 'in detail', i.e. when the Edit button is pressed.

wxPropertyListValidator::OnRetrieveValue

Bool OnRetrieveValue(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_)

Called when Tick (Confirm) is pressed or focus is lost or view wants to update the property list. Should do the transfer from the property editing area to the property itself

wxPropertyListValidator::OnSelect

Bool OnSelect(**Bool** _select_, **wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_)

Called when the property is selected or deselected: typically displays the value in the edit control (having chosen a suitable control to display: (non)editable text or listbox).

wxPropertyListValidator::OnValueListSelect

Bool OnValueListSelect(**wxProperty ***_property_, **wxPropertyListView ***_view_, **wxWindow ***_parentWindow_)

Called when the value listbox is selected. The default behaviour is to copy string to text control, and retrieve the value into the property.

3.18. wxPropertyListView: wxPropertyView

See also *wxPropertyListView overview* (page 38)

The **wxPropertyListView** class shows a wxPropertySheet as a Visual Basic-style property list.

wxPropertyListView::wxPropertyListView

void wxPropertyListView(**long** _flags = wxPROP_BUTTON_DEFAULT_)

Constructor.

The *flags* argument can be a bit list of the following:

- wxPROP_BUTTON_CLOSE
- wxPROP_BUTTON_OK
- wxPROP_BUTTON_CANCEL

- wxPROP_BUTTON_CHECK_CROSS
- wxPROP_BUTTON_HELP
- wxPROP_DYNAMIC_VALUE_FIELD
- wxPROP_PULLDOWN

wxPropertyListView::~wxPropertyListView

void ~wxPropertyListView(**void**)

Destructor.

wxPropertyListView::AssociatePanel

void AssociatePanel(**wxPanel ***_panel_)

Associates the window on which the controls will be displayed, with the view (sets an internal pointer to the window).

wxPropertyListView::BeginShowingProperty

Bool BeginShowingProperty(**wxProperty ***_property_)

Finds the appropriate validator and loads the property into the controls, by calling wxPropertyValidator::OnPrepareControls and then wxPropertyListView::DisplayProperty.

wxPropertyListView::DisplayProperty

Bool DisplayProperty(**wxProperty ***_property_)

Calls wxPropertyValidator::OnDisplayValue for the current property's validator. This function gets called by wxPropertyListView::BeginShowingProperty, which is in turn called from ShowProperty, called by OnPropertySelect, called by the listbox callback when selected.

wxPropertyListView::EndShowingProperty

Bool EndShowingProperty(**wxProperty ***_property_)

Finds the appropriate validator and unloads the property from the controls, by calling wxPropertyListView::RetrieveProperty, wxPropertyValidator::OnClearControls and (if we're in detailed editing mdoe) wxPropertyValidator::OnClearDetailControls.

wxPropertyListView::GetPanel

wxPanel * GetPanel(**void**)

Returns the panel associated with the view.

wxPropertyListView::GetManagedWindow

wxWindow * GetManagedWindow(**void**)

Returns the managed window (a frame or dialog) associated with the view.

wxPropertyListView::GetWindowCancelButton

wxButton * GetWindowCancelButton(**void**)

Returns the window cancel button, if any.

wxPropertyListView::GetWindowCloseButton

wxButton * GetWindowCloseButton(**void**)

Returns the window close or OK button, if any.

wxPropertyListView::GetWindowHelpButton

wxButton * GetWindowHelpButton(**void**)

Returns the window help button, if any.

wxPropertyListView::SetManagedWindow

void SetManagedWindow(**wxWindow ***win*)

Sets the managed window (a frame or dialog) associated with the view.

wxPropertyListView::UpdatePropertyDisplayInList

Bool UpdatePropertyDisplayInList(**wxProperty ***property*)

Updates the display for the given changed property.

wxPropertyListView::UpdatePropertyList

Bool UpdatePropertyList(**Bool** *clearEditArea = TRUE*)

Updates the whole property list display.

3.19. wxPropertySheet: wxObject

See also *wxPropertySheet overview* (page 39)

The **wxPropertySheet** class is used for storing a number of wxProperty objects (essentially names and values).

wxPropertySheet::wxPropertySheet

void wxPropertySheet(**void**)

Constructor.

wxPropertySheet::~wxPropertySheet

void ~wxPropertySheet(**void**)

Destructor. Destroys all contained properties.

wxPropertySheet::AddProperty

void AddProperty(**wxProperty ***_property_)

Adds a property to the sheet.

wxPropertySheet::Clear

void Clear(**void**)

Clears all the properties from the sheet (deleting them).

wxPropertySheet::GetProperties

wxList& GetProperties(**void**)

Returns a reference to the internal list of properties.

wxPropertySheet::GetProperty

wxProperty * GetProperty(**char ***_name_)

Gets a property by name.

wxPropertySheet::SetAllModified

void SetAllModified(**Bool** _flag_)

Sets the 'modified' flag of each property value.

3.20. wxPropertyValidator: wxEvtHandler

See also *wxPropertyValidator overview* (page 37)

The **wxPropertyValidator** abstract class is the base class for deriving validators for properties.

wxPropertyValidator::wxPropertyValidator

void wxPropertyValidator(**long** *flags = 0*)

Constructor.

wxPropertyValidator::~wxPropertyValidator

void ~wxPropertyValidator(**void**)

Destructor.

wxPropertyValidator::GetFlags

long GetFlags(**void**)

Returns the flags for the validator.

wxPropertyValidator::GetValidatorProperty

wxProperty * GetValidatorProperty(**void**)

Gets the property for the validator.

wxPropertyValidator::SetValidatorProperty

void SetValidatorProperty(**wxProperty ***property*)

Sets the property for the validator.

3.21. wxPropertyValidatorRegistry: wxHashTable

The **wxPropertyValidatorRegistry** class is used for storing validators, indexed by the 'role name' of the property, by which groups of property can be identified for the purpose of validation and editing.

wxPropertyValidatorRegistry::wxPropertyValidatorRegistry

void wxPropertyValidatorRegistry(**void**)

Constructor.

wxPropertyValidatorRegistry::~wxPropertyValidatorRegistry

void ~wxPropertyValidatorRegistry(**void**)

Destructor.

wxPropertyValidatorRegistry::Clear

void ClearRegistry(**void**)

Clears the registry, deleting the validators.

wxPropertyValidatorRegistry::GetValidator

wxPropertyValidator * GetValidator(**wxString&** *roleName*)

Retrieve a validator by the property role name.

wxPropertyValidatorRegistry::RegisterValidator

void RegisterValidator(**wxString&** *roleName*, **wxPropertyValidator ***validator*)

Register a validator with the registry. *roleName* is a name indicating the role of the property, such as "filename". Later, when a validator is chosen for editing a property, this role name is matched against the class names of the property, if the property does not already have a validator explicitly associated with it.

3.22. wxPropertyValue: wxObject

The **wxPropertyValue** class represents the value of a property, and is normally associated with a wxProperty object.

A wxPropertyValue has one of the following types:

- wxPropertyValueNull
- wxPropertyValueInteger
- wxPropertyValueReal
- wxPropertyValueBool
- wxPropertyValueString
- wxPropertyValueList
- wxPropertyValueIntegerPtr
- wxPropertyValueRealPtr
- wxPropertyValueBoolPtr
- wxPropertyValueStringPtr

wxPropertyValue::wxPropertyValue

void wxPropertyValue(**void**)

Default constructor.

void wxPropertyValue(**const wxPropertyValue&** *copyFrom*)

Copy constructor.

void wxPropertyValue(**char ***val*)

Construction from a string value.

void wxPropertyValue(**long** *val*)

Construction from an integer value. You may need to cast to (long) to avoid confusion with other constructors (such as the Bool constructor).

void wxPropertyValue(**Bool** *val*)

Construction from a boolean value.

void wxPropertyValue(**float** *val*)

Construction from a floating point value.

void wxPropertyValue(**double** *val*)

Construction from a floating point value.

void wxPropertyValue(**wxList *** *val*)

Construction from a list of wxPropertyValue objects. The list, but not each contained wxPropertyValue, will be deleted by the constructor. The wxPropertyValues will be assigned to this wxPropertyValue list. In other words, so do not delete wxList or its data after calling this constructor.

void wxPropertyValue(**wxStringList *** *val*)

Construction from a list of strings. The list (including the strings contained in it) will be deleted by the constructor, so do not destroy *val* explicitly.

void wxPropertyValue(**char ****val*)

Construction from a string pointer.

void wxPropertyValue(**long ***val*)

Construction from an integer pointer.

void wxPropertyValue(**Bool ***val*)

Construction from an boolean pointer.

void wxPropertyValue(**float ***val*)

Construction from a floating point pointer.

The last four constructors use pointers to various C++ types, and do not store the types themselves; this allows the values to stand in for actual data values defined elsewhere.

wxPropertyValue::~wxPropertyValue

void ~wxPropertyValue(**void**)

Destructor.

wxPropertyValue::Append

void Append(**wxPropertyValue ***expr*)

Appends a property value to the list.

wxPropertyValue::BoolValue

Bool BoolValue(**void**)

Returns the boolean value.

wxPropertyValue::BoolValuePtr

Bool * BoolValuePtr(**void**)

Returns the pointer to the boolean value.

wxPropertyValue::ClearList

void ClearList(**void**)

Deletes the contents of the list.

wxPropertyValue::Delete

void Delete(**wxPropertyValue ***expr*)

Deletes *expr* from this list.

wxPropertyValue::GetFirst

wxPropertyValue * GetFirst(**void**)

Gets the first value in the list.

wxPropertyValue::GetLast

wxPropertyValue * GetFirst(**void**)

Gets the last value in the list.

wxPropertyValue::GetModified

Bool GetModified(**void**)

Returns TRUE if the value was modified since being created (or since SetModified was called).

wxPropertyValue::GetNext

wxPropertyValue * GetNext(**void**)

Gets the next value in the list (the one after 'this').

wxPropertyValue::GetStringRepresentation

wxString GetStringRepresentation(**void**)

Gets a string representation of the value.

wxPropertyValue::IntegerValue

long IntegerValue(**void**)

Returns the integer value.

wxPropertyValue::Insert

void Insert(**wxPropertyValue ***_expr_)

Inserts a property value at the front of a list.

wxPropertyValue::IntegerValuePtr

long * IntegerValuePtr(**void**)

Returns the pointer to the integer value.

wxPropertyValue::Nth

wxPropertyValue * Nth(**int** _n_)

Returns the nth value of a list expression (starting from zero).

wxPropertyValue::Number

int Number(**void**)

Returns the number of elements in a list expression.

wxPropertyValue::RealValue

float RealValue(**void**)

Returns the floating point value.

wxPropertyValue::RealValuePtr

float * RealValuePtr(**void**)

Returns the pointer to the floating point value.

wxPropertyValue::SetModified

void SetModified(**Bool** *flag*)

Sets the 'modified' flag.

wxPropertyValue::StringValue

char * StringValue(**void**)

Returns the string value.

wxPropertyValue::StringValuePtr

char ** StringValuePtr(**void**)

Returns the pointer to the string value.

wxPropertyValue::Type

wxPropertyValueType Type(**void**)

Returns the value type.

wxPropertyValue::operator =

void operator =(**const wxPropertyValue&** *val*)

void operator =(**const char ***val*)

void operator =(**const long** *val*)

void operator =(**const Bool** *val*)

void operator =(**const float** *val*)

void operator =(**const char ****val*)

void operator =(**const long ***val*)

void operator =(**const Bool ***val*)

void operator =(**const float ***val*)

Assignment operators.

3.23. wxPropertyView: wxEvtHandler

See also *wxPropertyView overview* (page 38)

The **wxPropertyView** abstract class is the base class for views of property sheets, acting as intermediaries between properties and actual windows.

wxPropertyView::wxPropertyView

void wxPropertyView(**long** *flags = wxPROP_BUTTON_DEFAULT*)

Constructor.

The *flags* argument can be a bit list of the following:

- wxPROP_BUTTON_CLOSE
- wxPROP_BUTTON_OK
- wxPROP_BUTTON_CANCEL
- wxPROP_BUTTON_CHECK_CROSS
- wxPROP_BUTTON_HELP
- wxPROP_DYNAMIC_VALUE_FIELD
- wxPROP_PULLDOWN

wxPropertyView::~wxPropertyView

void ~wxPropertyView(**void**)

Destructor.

wxPropertyView::AddRegistry

void AddRegistry(**wxPropertyValidatorRegistry ***registry*)

Adds a registry (list of property validators) the view's list of registries, which is initially empty.

wxPropertyView::FindPropertyValidator

wxPropertyValidator * FindPropertyValidator(**wxProperty ***property*)

Finds the property validator that is most appropriate to this property.

wxPropertyView::GetPropertySheet

wxPropertySheet * GetPropertySheet(**void**)

Gets the property sheet for this view.

wxPropertyView::GetRegistryList

wxList& GetRegistryList(**void**)

Returns a reference to the list of property validator registries.

wxPropertyView::OnOk

void OnOk(**void**)

Virtual function that will be called when the OK button on the physical window is pressed (if it exists).

wxPropertyView::OnCancel

void OnCancel(**void**)

Virtual function that will be called when the Cancel button on the physical window is pressed (if it exists).

wxPropertyView::OnClose

Bool OnClose(**void**)

Virtual function that will be called when the physical window is closed. The default implementation returns FALSE.

wxPropertyView::OnHelp

void OnHelp(**void**)

Virtual function that will be called when the Help button on the physical window is pressed (if it exists).

wxPropertyView::OnPropertyChanged

void OnPropertyChanged(**wxProperty ***property*)

Virtual function called by a view or validator when a property's value changed. Validators must be written correctly for this to be called. You can override this function to respond immediately to property value changes.

wxPropertyView::OnUpdateView

Bool OnUpdateView(**void**)

Called by the viewed object to update the view. The default implementation just returns FALSE.

wxPropertyView::SetPropertySheet

void SetPropertySheet(**wxPropertySheet ***_sheet_)

Sets the property sheet for this view.

wxPropertyView::ShowView

void ShowView(**wxPropertySheet ***_sheet_, **wxPanel ***_panel_)

Associates this view with the given panel, and shows the view.

3.24. wxRealFormValidator: wxPropertyFormValidator

See also *Validator classes* (page 31)

This class validates a range of real values for form views. The associated panel item must be a wxText.

wxRealFormValidator::wxRealFormValidator

void wxRealFormValidator(**float** *min=0.0*, **float** *max=0.0*, **long** *flags=0*)

Constructor. Assigning zero to minimum and maximum values indicates that there is no range to check.

3.25. wxStringFormValidator: wxPropertyFormValidator

See also *Validator classes* (page 31)

This class validates a string value for a form view, with an optional choice of possible values. The associated panel item must be a wxText, wxListBox or wxChoice. For wxListBox and wxChoice items, if the item is empty, the validator attempts to initialize the item from the strings in the validator. Note that this does not happen for XView wxChoice items since XView cannot reinitialize a wxChoice.

wxStringFormValidator::wxStringFormValidator

void wxStringFormValidator(**wxStringList ***_list=NULL_, **long** *flags=0*)

Constructor. Supply a list of strings to indicate a choice, or no strings to allow the user to freely edit the string. The string list will be deleted when the validator is deleted.

3.26. wxRealListValidator: wxPropertyListValidator

See also *Validator classes* (page 31)

This class validates a range of real values for property lists.

wxRealListValidator::wxreallistvalidator

void wxRealListValidator(**float** *min=0.0*, **float** *max=0.0*, **long** *flags=wxPROP_ALLOW_TEXT_EDITING*)

Constructor. Assigning zero to minimum and maximum values indicates that there is no range to check.

3.27. wxStringListValidator: wxPropertyListValidator

See also *Validator classes* (page 31)

This class validates a string value, with an optional choice of possible values.

wxStringListValidator::wxStringListValidator

void wxStringListValidator(**wxStringList ***_list=NULL_, **long** *flags=0*)

Constructor. Supply a list of strings to indicate a choice, or no strings to allow the user to freely edit the string. The string list will be deleted when the validator is deleted.

4. Classes by category

A classification of property sheet classes by category.

4.1. Data classes

- *wxProperty* (page 6)
- *wxPropertyValue* (page 22)
- *wxPropertySheet* (page 19)

4.2. Validator classes

Validators check that the values the user has entered for a property are valid. They can also define specific ways of entering data, such as a file selector for a filename, and they are responsible for transferring values between the wxProperty and the physical display.

Base classes:

- *wxPropertyValidator* (page 6)
- *wxPropertyListValidator* (page 15)
- *wxPropertyFormValidator* (page 9)

List view validators:

- *wxBoolListValidator* (page 5)
- *wxFilenameListValidator* (page 6)
- *wxIntegerListValidator* (page 5)
- *wxListOfStringsListValidator* (page 6)
- *wxRealListValidator* (page 30)
- *wxStringListValidator* (page 30)

Form view validators:

- *wxBoolFormValidator* (page 5)
- *wxIntegerFormValidator* (page 5)
- *wxRealFormValidator* (page 29)
- *wxStringFormValidator* (page 29)

4.3. View classes

View classes mediate between a property sheet and a physical window.

- *wxPropertyView* (page 27)
- *wxPropertyListView* (page 17)
- *wxPropertyFormView* (page 11)

4.4. Window classes

The class library defines some window classes that can be used as-is with a suitable view class and property sheet.

- *wxPropertyFormFrame* (page 9)
- *wxPropertyFormDialog* (page 9)

- *wxPropertyFormPanel* (page 10)
- *wxPropertyListFrame* (page 14)
- *wxPropertyListDialog* (page 13)
- *wxPropertyListPanel* (page 15)

4.5. Registry classes

A validator registry is a list of validators that can be applied to properties in a property sheet. There may be one or more registries per property view.

- *wxPropertyValidatorRegistry* (page 21)

5. Topic overviews

This chapter contains a selection of topic overviews.

5.1. Property classes overview

The property classes help a programmer to express relationships between data and physical windows, in particular:

- the transfer of data to and from the physical controls;
- the behaviour of various controls and custom windows for particular types of data;
- the validation of data, notifying the user when incorrect data is entered, or even better, constraining the input so only valid data can be entered.

With a consistent framework, the programmer should be able to use existing components and design new ones in a principled manner, to solve many data entry requirements.

Each datum is represented in a *wxProperty* (page 6), which has a name and a value. Various C++ types are permitted in the value of a property, and the property can store a pointer to the data instead of a copy of the data. A *wxPropertySheet* (page 19) represents a number of these properties.

These two classes are independent from the way in which the data is visually manipulated. To mediate between property sheets and windows, the abstract class *wxPropertyView* (page 27) is available for programmers to derive new kinds of view. One kind of view that is available is the *wxPropertyListView* (page 17), which displays the data in a Visual Basic-style list, with a small number of controls for editing the currently selected property. Another is *wxPropertyFormView* (page 11) which mediates between an existing dialog or panel and the property sheet.

The hard work of mediation is actually performed by validators, which are instances of classes derived from *wxPropertyValidator* (page 20). A validator is associated with a particular property and is responsible for responding to user interface events, and displaying, updating and checking the property value. Because a validator's behaviour depends largely on the kind of view being used, there has to be a separate hierarchy of validators for each class of view. So for wxPropertyListView, there is an abstract class *wxPropertyListValidator* (page 15) from which concrete classes are derived, such as *wxRealListValidator* (page 30) and *wxStringListValidator* (page 30).

A validator can be explicitly set for a property, so there is no doubt which validator should be used to edit that property. However, it is also possible to define a registry of validators, and have the validator chosen on the basis of the *role* of the property. So a property with a "filename" role would match the "filename" validator, which pops up a file selector when the user double clicks on the property.

You don't have to define your own frame or window classes: there are some predefined that will work with the property list view. See *Window classes* (page 31) for further details.

5.1.1. Example 1: Property list view

The following code fragment shows the essentials of creating a registry of standard validators, a property sheet containing some properties, and a property list view and dialog or frame. RegisterValidators will be called on program start, and PropertySheetTest is called in response to a menu command.

Note how some properties are created with an explicit reference to a validator, and others are provided with a "role" which can be matched against a validator in the registry.

The interface generated by this test program is shown in the section *Appearance and behaviour of a property list view* (page 1).

```
void RegisterValidators(void)
{
  myListValidatorRegistry.RegisterValidator((wxString)"real", new
wxRealListValidator);
  myListValidatorRegistry.RegisterValidator((wxString)"string", new
wxStringListValidator);
  myListValidatorRegistry.RegisterValidator((wxString)"integer", new
wxIntegerListValidator);
  myListValidatorRegistry.RegisterValidator((wxString)"bool", new
wxBoolListValidator);
}

void PropertyListTest(Bool useDialog)
{
  wxPropertySheet *sheet = new wxPropertySheet;

  sheet->AddProperty(new wxProperty("fred", 1.0, "real"));
  sheet->AddProperty(new wxProperty("tough choice", (Bool)TRUE,
"bool"));
  sheet->AddProperty(new wxProperty("ian", (long)45, "integer", new
wxIntegerListValidator(-50, 50)));
  sheet->AddProperty(new wxProperty("bill", 25.0, "real", new
wxRealListValidator(0.0, 100.0)));
  sheet->AddProperty(new wxProperty("julian", "one", "string"));
  sheet->AddProperty(new wxProperty("bitmap", "none", "string", new
wxFilenameListValidator("Select a bitmap file", "*.bmp")));
  wxStringList *strings = new wxStringList("one", "two", "three",
NULL);
  sheet->AddProperty(new wxProperty("constrained", "one", "string", new
wxStringListValidator(strings)));

  wxPropertyListView *view =
    new wxPropertyListView(NULL,

wxPROP_BUTTON_CHECK_CROSS|wxPROP_DYNAMIC_VALUE_FIELD|wxPROP_PULLDOWN);

  wxDialogBox *propDialog = NULL;
  wxPropertyListFrame *propFrame = NULL;
  if (useDialog)
  {
    propDialog = new wxPropertyListDialog(view, NULL, "Property Sheet
Test", TRUE, -1, -1, 400, 500);
  }
  else
  {
    propFrame = new wxPropertyListFrame(view, NULL, "Property Sheet
Test", -1, -1, 400, 500);
  }

  view->AddRegistry(&myListValidatorRegistry);
```

```
  if (useDialog)
  {
    view->ShowView(sheet, propDialog);
    propDialog->Centre(wxBOTH);
    propDialog->Show(TRUE);
  }
  else
  {
    propFrame->Initialize();
    view->ShowView(sheet, propFrame->GetPropertyPanel());
    propFrame->Centre(wxBOTH);
    propFrame->Show(TRUE);
  }
}
```

5.1.2. Example 2: Property form view

This example is similar to Example 1, but uses a property form view to edit a property sheet using a predefined dialog box.

```
void RegisterValidators(void)
{
  myFormValidatorRegistry.RegisterValidator((wxString)"real", new
wxRealFormValidator);
  myFormValidatorRegistry.RegisterValidator((wxString)"string", new
wxStringFormValidator);
  myFormValidatorRegistry.RegisterValidator((wxString)"integer", new
wxIntegerFormValidator);
  myFormValidatorRegistry.RegisterValidator((wxString)"bool", new
wxBoolFormValidator);
}

void PropertyFormTest(Bool useDialog)
{
  wxPropertySheet *sheet = new wxPropertySheet;

  sheet->AddProperty(new wxProperty("fred", 25.0, "real", new
wxRealFormValidator(0.0, 100.0)));
  sheet->AddProperty(new wxProperty("tough choice", (Bool)TRUE,
"bool"));
  sheet->AddProperty(new wxProperty("ian", (long)45, "integer", new
wxIntegerFormValidator(-50, 50)));
  sheet->AddProperty(new wxProperty("julian", "one", "string"));
  wxStringList *strings = new wxStringList("one", "two", "three",
NULL);
  sheet->AddProperty(new wxProperty("constrained", "one", "string", new
wxStringFormValidator(strings)));

  wxPropertyFormView *view = new wxPropertyFormView(NULL);

  wxDialogBox *propDialog = NULL;
  wxPropertyFormFrame *propFrame = NULL;
  if (useDialog)
  {
    propDialog = new wxPropertyFormDialog(view, NULL, "Property Form
```

```
Test", TRUE, -1, -1, 400, 300);
  }
  else
  {
    propFrame = new wxPropertyFormFrame(view, NULL, "Property Form
Test", -1, -1, 400, 300);
    propFrame->Initialize();
  }

  wxPanel *panel = propDialog ? propDialog : propFrame-
>GetPropertyPanel();
  panel->SetLabelPosition(wxVERTICAL);

  // Add items to the panel

  (void) new wxButton(panel, (wxFunction)NULL, "OK", -1, -1, -1, -1, 0,
"ok");
  (void) new wxButton(panel, (wxFunction)NULL, "Cancel", -1, -1, 80, -
1, 0, "cancel");
  (void) new wxButton(panel, (wxFunction)NULL, "Update", -1, -1, 80, -
1, 0, "update");
  (void) new wxButton(panel, (wxFunction)NULL, "Revert", -1, -1, -1, -
1, 0, "revert");
  panel->NewLine();

  // The name of this text item matches the "fred" property
  (void) new wxText(panel, (wxFunction)NULL, "Fred", "", -1, -1, 90, -
1, 0, "fred");
  (void) new wxCheckBox(panel, (wxFunction)NULL, "Yes or no", -1, -1, -
1, -1, 0, "tough choice");
  (void) new wxSlider(panel, (wxFunction)NULL, "Sliding scale", 0, -50,
50, 100, -1, -1, wxHORIZONTAL, "ian");
  panel->NewLine();
  (void) new wxListBox(panel, (wxFunction)NULL, "Constrained",
wxSINGLE, -1, -1, 100, 90, 0, NULL, 0, "constrained");

  view->AddRegistry(&myFormValidatorRegistry);

  if (useDialog)
  {
    view->ShowView(sheet, propDialog);
    view->AssociateNames();
    view->TransferToDialog();
    propDialog->Centre(wxBOTH);
    propDialog->Show(TRUE);
  }
  else
  {
    view->ShowView(sheet, propFrame->GetPropertyPanel());
    view->AssociateNames();
    view->TransferToDialog();
    propFrame->Centre(wxBOTH);
    propFrame->Show(TRUE);
  }
}
```

5.2. Validator classes overview

Classes: *Validator classes* (page 31)

The validator classes provide functionality for mediating between a wxProperty and the actual display. There is a separate family of validator classes for each class of view, since the differences in user interface for these views implies that little common functionality can be shared amongst validators.

5.2.1. wxPropertyValidator overview

Class: *wxPropertyValidator* (page 20)

This class is the root of all property validator classes. It contains a small amount of common functionality, including functions to convert between strings and C++ values.

A validator is notionally an object which sits between a property and its displayed value, and checks that the value the user enters is correct, giving an error message if the validation fails. In fact, the validator does more than that, and is akin to a view class but at a finer level of detail. It is also responsible for loading the dialog box control with the value from the property, putting it back into the property, preparing special controls for editing the value, and may even invoke special dialogs for editing the value in a convenient way.

In a property list dialog, there is quite a lot of scope for supplying custom dialogs, such as file or colour selectors. For a form dialog, there is less scope because there is no concept of 'detailed editing' of a value: one control is associated with one property, and there is no provision for invoking further dialogs. The reader may like to work out how the form view could be extended to provide some of the functionality of the property list!

Validator objects may be associated explictly with a wxProperty, or they may be indirectly associated by virtue of a property 'kind' that matches validators having that kind. In the latter case, such validators are stored in a validator registry which is passed to the view before the dialog is shown. If the validator takes arguments, such as minimum and maximum values in the case of a wxIntegerListValidator, then the validator must be associated explicitly with the property. The validator will be deleted when the property is deleted.

5.2.2. wxPropertyListValidator overview

Class: *wxPropertyListValidator* (page 15)

This class is the abstract base class for property list view validators. The list view acts upon a user interface containing a list of properties, a text item for direct property value editing, confirm/cancel buttons for the value, a pulldown list for making a choice between values, and OK/Cancel/Help buttons for the dialog (see *property list appearance* (page 1)).

By overriding virtual functions, the programmer can create custom behaviour for different kinds of property. Custom behaviour can use just the available controls on the property list dialog, or the validator can invoke custom editors with quite different controls, which pop up in 'detailed editing' mode.

See the detailed class documentation for the members you should override to give your validator appropriate behaviour.

5.2.3. wxPropertyFormValidator overview

This class is the abstract base class for property form view validators. The form view acts upon an existing dialog box or panel, where either the panel item names correspond to property names, or the programmer has explicitly associated the panel item with the property.

By overriding virtual functions, the programmer determines how values are displayed or retrieved, and the checking that the validator does.

See the detailed class documentation for the members you should override to give your validator appropriate behaviour.

5.3. View classes overview

Classes: *View classes* (page 31)

An instance of a view class relates a property sheet with an actual window. Currently, there are two classes of view: wxPropertyListView and wxPropertyFormView.

5.3.1. wxPropertyView overview

Class: *wxPropertyView* (page 27)

This is the abstract base class for property views.

5.3.2. wxPropertyListView overview

Class: *wxPropertyListView* (page 17)

The property list view defines the relationship between a property sheet and a property list dialog or panel. It manages user interface events such as clicking on a property, pressing return in the text edit field, and clicking on Confirm or Cancel. These events cause member functions of the class to be called, and these in turn may call member functions of the appropriate validator to be called, to prepare controls, check the property value, invoke detailed editing, etc.

5.3.3. wxPropertyFormView overview

Class: *wxPropertyFormView* (page 11)

The property form view manages the relationship between a property sheet and an existing dialog or panel.

You must first create a panel or dialog box for the view to work on. The panel should contain panel items with names that correspond to properties in your property sheet; or you can explicitly set the panel item for each property.

Apart from any custom panel items that you wish to control independently of the property-editing items, wxPropertyFormView takes over the processing of item events. It can also control normal dialog behaviour such as OK, Cancel, so you should also create some standard buttons that the property view can recognise. Just create the buttons with standard names and the view will do the rest. The following button names are recognised:

- **ok**: indicates the OK button. Calls wxPropertyFormView::OnOk. By default, checks and updates the form values, closes and deletes the frame or dialog, then deletes the view.
- **cancel**: indicates the Cancel button. Calls wxPropertyFormView::OnCancel. By default, closes and deletes the frame or dialog, then deletes the view.
- **help**: indicates the Help button. Calls wxPropertyFormView::OnHelp. This needs to be overridden by the application for anything interesting to happen.
- **revert**: indicates the Revert button. Calls wxPropertyFormView::OnRevert, which by default transfers the wxProperty values to the panel items (in effect undoing any unsaved changes in the items).
- **update**: indicates the Revert button. Calls wxPropertyFormView::OnUpdate, which by defaults transfers the displayed values to the wxProperty objects.

5.4. wxPropertySheet overview

Classes: *wxPropertySheet* (page 19), *wxProperty* (page 6), *wxPropertyValue* (page 22)

A property sheet defines zero or more properties. This is a bit like an explicit representation of a C++ object. wxProperty objects can have values which are pointers to C++ values, or they can allocate their own storage for values.

Because the property sheet representation is explicit and can be manipulated by a program, it is a convenient form to be used for a variety of editing purposes. wxPropertyListView and wxPropertyFormView are two classes that specify the relationship between a property sheet and a user interface. You could imagine other uses for wxPropertySheet, for example to generate a form-like user interface without the need for GUI programming. Or for storing the names and values of command-line switches, with the option to subsequently edit these values using a wxPropertyListView.

A typical use for a property sheet is to represent values of an object which are only implicit in the current representation of it. For example, in Visual Basic and similar programming environments, you can 'edit a button', or rather, edit the button's properties. One of the properties you can edit is *width* - but there is no explicit representation of width in a wxWindows button; instead, you call SetSize and GetSize members. To translate this into a consisent, property-oriented scheme, we could derive a new class wxButtonWithProperties, which has two new functions: SetProperty and GetProperty. SetProperty accepts a property name and a value, and calls an appropriate function for the property that is being passed. GetProperty accepts a property name, returning a property value. So instead of having to use the usual arbitrary set of C++ member functions to set or access attributes of a window, programmer deals merely with SetValue/GetValue, and property names and values. We now have a single point at which we can modify or query an object by specifying names and values at run-time. (The implementation of SetProperty and GetProperty is probably quite messy and involves a large if-then-else statement to test the property name and act accordingly.)

When the user invokes the property editor for a wxButtonWithProperties, the system creates a wxPropertySheet with 'imaginary' properties such as width, height, font size and so on. For each property, wxButtonWithProperties::GetProperty is called, and the result is passed to the corresponding wxProperty. The wxPropertySheet is passed to a wxPropertyListView as described elsewhere, and the user edits away. When the user has finished editing, the system calls wxButtonWithProperties::SetProperty to transfer the wxProperty value back into the button by way of an appropriate call, wxWindow::SetSize in the case of width and height properties.

6. Change log

November 26th 1995, Version 1.1

- Added wxListOfStringsListValidator - allows adding, deleting, editing strings.
- Added wxPropertyValue::ClearList, wxPropertyValue::Delete, wxPropertyValue::wxPropertyValue(wxStringList *).
- Added wxPropertyValue::Set/GetModified, wxPropertySheet::SetAllModified.
- Added wxPropertyView::OnPropertyChanged support, for immediate feedback.

October 1995, Version 1.0

- First release.

Index

—R—

—S—

—T—

—U—

—W—