



All Topics, MFC / C++ >> List Controls >> Extending the List Control
<http://www.codeproject.com/listctrl/ReportControl.asp>

VC7.1, VC7, VC6, XP,
 W2K, Win2003, Win9X,
 MFC

Posted 2 Dec 2003

Updated 1 Jan 2004

19,266 views

Another Report List Control

By = [Abin] =

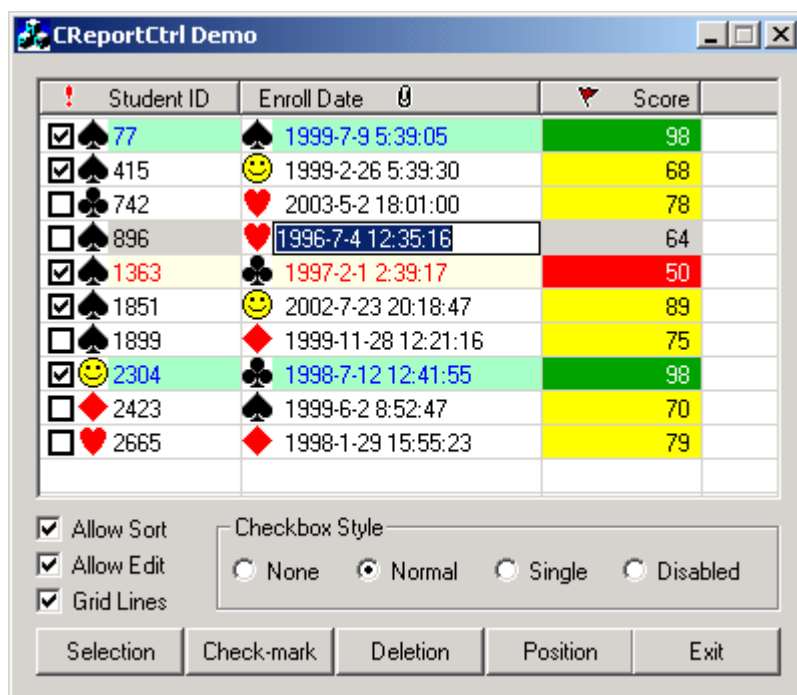
A report style CListCtrl supporting sorting, sub-item editing, sub-item image, sub-item color etc.

18 members have rated this article. Result:

Popularity: 4.61. Rating: 3.68 out of 5.

Download source files - 14.9 Kb

Download demo project - 143 Kb



Introduction

By seeing how widely "Report List Controls" are used in various applications, it is really needless to introduce what list controls are and what report style is, so I will just skip "what it is" and come right to "what it does".

Features

Item Sorting

When the user left-clicks on a column header, or the developer issues a call to `CReportCtrl::SortItems`, a whole column of items are sorted. Their text formats will be recognized appropriately (decimal numeric, hex numeric, decimal points, percentages, date & time, plain text) and sorted accordingly.

A message `WM_ITEM_SORTED` is sent to the parent window after a sorting is completed, the

column index is passed as `wParam` and sorting method is passed as `lParam` (0=descending, 1=ascending).

Related Methods:

```

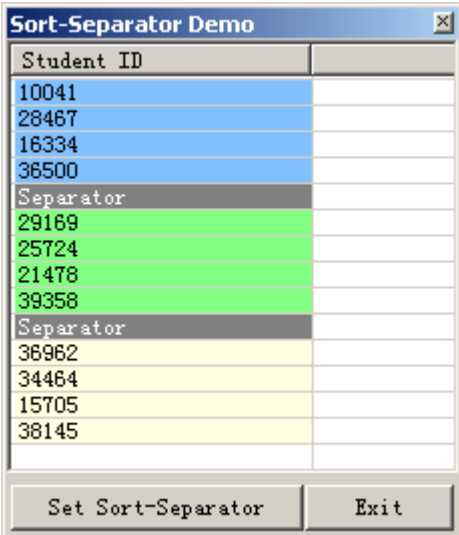
BOOL IsSortable() const; // Is sort allowed?
BOOL SetSortable(BOOL bSet); // Allow/disallow sorting
BOOL IsSortAscending() const;
int GetSortedColumn() const;
// Sort a specified column.
void SortItems(int nColumn, BOOL bAscending);

```

Item Sort-Separator

Sometimes you may not want a whole column to be sorted altogether, rather, you want them to act like multiple *parts* so that each *part* of items are sorted internally whereas *parts* themselves remain unaltered. Please imagine this scenario, you are using the control to display an audit report table, all rows are numeric values and the last row is the sum of all above values; now if the user sorts the column in descending order, the row which contains the sum-value will be moved to the top because basically the sum-value will most likely be greater than any other value. That's NOT what you want, you would like the sum row always stay on the bottom of the table, right? Now you would wish there's a way to insert some kind of a "separator" between the sum row and all other data rows, so the sorting would never affect the sum row. How to do that? Easy, use `CReportCtrl::SetSortSeparator`.

Below is an unsorted table, I want the 3 parts (blue, green, yellow) to be sorted separately:



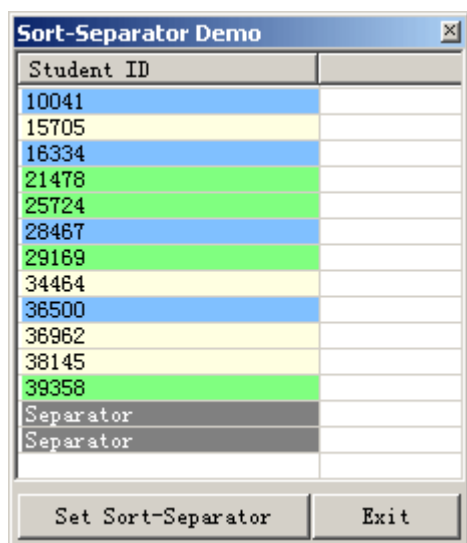
Student ID	
10041	
28467	
18334	
36500	
Separator	
29189	
25724	
21478	
39358	
Separator	
36962	
34464	
15705	
38145	

Sort it without using any separator:

```

m_wndList.SetSortSeparator(NULL); // Disable sort-separator
// Sort the first column in ascending order
m_wndList.SortItems(0, TRUE);

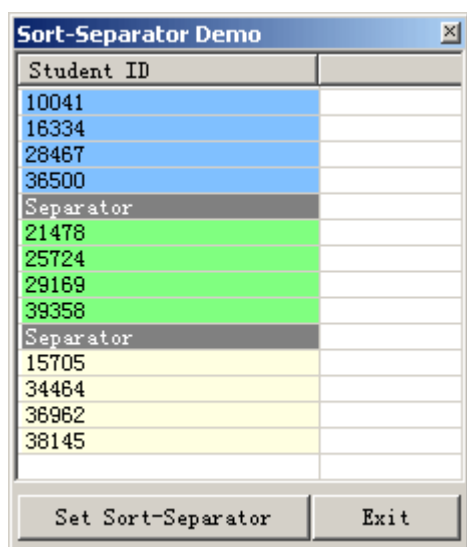
```



See how the table became a mess and the 3 parts got broken up, that's not what we wanted. But if at the above step, you used sort-separator instead:

```
// m_wndList.SetSortSeparator(NULL); // Disable sort-separator
// m_wndList.SortItems(0, TRUE); // Sort the first column in ascending order

// Use string "Separator" as the sort-separator
m_wndList.SetSortSeparator(_T("Separator"));
// Sort the first column in ascending order
m_wndList.SortItems(0, TRUE);
```



Now it's nice, the 3rd image is exactly what we were expecting.

Related Methods:

```
void SetSortSeparator(LPCTSTR lpSortSeparator);
LPCTSTR GetSortSeparator() const;
```

Sub-item Text Editing

When the user double-clicks on a grid, or left-clicks on a grid which previously had selection mark and focus, or the developer issues a call to `CReportCtrl::StartEdit`, the item text of

that sub-item becomes editable.

While the item text is being edited, if the user clicks anywhere other than the under-editing-grid in the control, or presses the "Esc" key, or presses the "Enter" key, or the developer issues a call to `CReportCtrl::EndEdit`, the item editing is ended and changes made will either be committed or cancelled, depending on the action types which ended the editing.

Related Methods:

```

BOOL IsEditable() const; // Is Item text editable?
void SetEditable(BOOL bSet = TRUE); // Allow item text editing

// Display the editbox, previous edit are committed
BOOL StartEdit(int nItem, int nSubItem);

BOOL EndEdit(BOOL bCommit = TRUE); // Commit/cancel text edit, hide the editbox
CEdit* GetEditControl();

```

Customizing Checkbox Styles

Checkboxes inside a report list control are useful but I often found that I really need to get a little more hold of them, for example, sometimes I want to be notified when the user clicks on a checkbox, or I want to make the checkboxes read-only so that users cannot alter the checked/unchecked value of the checkboxes by inputs. These could be quite tricky to achieve in a plain MFC `CListCtrl`, but became a piece of cake in this class.

Checkbox styles can be specified by calling `CReportCtrl::SetCheckboxStyle`. Recognizable styles are:

- | `RC_CHKBOX_NONE` -- No checkbox displayed
- | `RC_CHKBOX_NORMAL` -- Normal style, multiple check marks allowed
- | `RC_CHKBOX_SINGLE` -- Single check only. Checkboxes are mutually exclusive, just like using radio buttons.
- | `RC_CHKBOX_DISABLED` -- Disabled, cannot be checked/unchecked by user input.

A message `WM_ON_CHKBOX` is sent to the parent window whenever the user clicks on the checkbox of a list item. The item index is passed as `wParam` and the mouse event (Such as `WM_LBUTTONDOWN`, `WM_RBUTTONDOWN`, etc.) is passed as `lParam`.

Related Methods:

```

void SetCheckboxStyle(int nStyle = RC_CHKBOX_NORMAL); // Set checkbox styles.
int GetCheckboxStyle() const;

// Example:

// Make the checkbox read-only
m_wndList.SetCheckboxStyle(RC_CHKBOX_DISABLED);

```

Sub-item Images & Sub-item Colors

Each grid can have its own image, text color and background color. To use images, you need to first call `CReportCtrl::SetImageList` to specify an image list or a bitmap resource ID.

Related Methods:

```

// Column header images
BOOL SetHeaderImage(int nColumn, int nIndex, BOOL bLeftSide = TRUE);
int GetHeaderImage(int nColumn) const;
CImageList* SetHeaderImageList(UINT nBitmapID, COLORREF crMask = RGB(255, 0, 255));
CImageList* SetHeaderImageList(CImageList* pImageList);

// Sub-item images
BOOL SetItemImage(int nItem, int nSubItem, int nIndex);
int GetItemImage(int nItem, int nSubItem) const;
CImageList* SetImageList(UINT nBitmapID, COLORREF crMask = RGB(255, 0, 255));
CImageList* SetImageList(CImageList* pImageList);
CImageList* GetImageList() const;

// Sub-item Text & Background Color
void SetItemTextColor(int nItem = -1, int nSubItem = -1,
    COLORREF color = COLOR_INVALID, BOOL bRedraw = TRUE);
COLORREF GetItemTextColor(int nItem, int nSubItem) const;
void SetItemBkColor(int nItem = -1, int nSubItem = -1,
    COLORREF color = COLOR_INVALID, BOOL bRedraw = TRUE);
COLORREF GetItemBkColor(int nItem, int nSubItem) const;

// Example: Set image and color for grid - 1st row 3rd column
m_wndList.SetImageList(IDB_BITMAP1); // Using bitmap resource "IDB_BITMAP1"
m_wndList.SetItemImage(0, 2, 0); // Set the 1st row 3rd column with image index 0

// Set grid colors
m_wndList.SetItemTextColor(0, 2, RGB(255, 0, 0), TRUE); // Grid text color: red
m_wndList.SetItemBkColor(0, 2, RGB(0, 255, 0), TRUE); // Grid background color: green

```

Convenient Item Operating

List item operations are made very simple and intuitive, you can select/unselect, check/uncheck, delete items which match a given state or a combination of given states. Every list item may have one or more of the following states:

- | **RC_ITEM_ALL** - All items regardless of states
- | **RC_ITEM_SELECTED** - Selected items
- | **RC_ITEM_UNSELECTED** - Unselected items
- | **RC_ITEM_CHECKED** - Checked items
- | **RC_ITEM_UNCHECKED** - Unchecked items
- | **RC_ITEM_FOCUSED** - Focused item
- | **RC_ITEM_UNFOCUSED** - Unfocused items

Multiple states can be combined using the "|" operator. Note that in any combination of states, if **RC_ITEM_ALL** is present, then all other states will be ignored and the state examination will always return **TRUE**.

Related Methods:

```

int GetFirstItem(DWORD dwStates = RC_ITEM_ALL, int nStartAfter = -1) const;
int GetLastItem(DWORD dwStates = RC_ITEM_ALL, int nStartBefore = -1) const;
int GetItemCount(DWORD dwStates = RC_ITEM_ALL) const;
DWORD GetItemStates(int nItem) const;
BOOL ExamItemStates(int nItem, DWORD dwStates) const;
BOOL SetItemStates(int nItem, DWORD dwNewStates);
int SetAllItemStates(DWORD dwOldStates, DWORD dwNewStates);

```

```

void InvertItems(int nType); // RC_INVERT_SELECTION or RC_INVERT_CHECKMARK

// Example:

// Select all items which were unselected and unchecked
m_wndList.SetAllItemStates(RC_ITEM_UNSELECTED | RC_ITEM_UNCHECKED, RC_ITEM_SELECTED);

// How many items are not checked?
int n = m_wndList.GetItemCount(RC_ITEM_UNCHECKED);

//Unselect and check all items regardless of their previous states
m_wndList.SetAllItemStates(RC_ITEM_ALL, RC_ITEM_UNSELECTED | RC_ITEM_CHECKED);

// Delete all items which were selected and checked
m_wndList.DeleteAllItems(RC_ITEM_SELECTED | RC_ITEM_CHECKED);

```

Convenient Item Positioning

Item position manipulations are extensively implemented in this class, you can easily move items up or down, or to a particular position, or swap 2 existing items, etc. The following member functions are designed for this kind of operations:

```

int MoveUp(int nItem, int nCount = 1); // Move an item upwards by "nCount" positions.

// Move an item downwards by "nCount" positions.
int MoveDown(int nItem, int nCount = 1);

int MoveToTop(int nItem); // Move an item up to the top.
int MoveToBottom(int nItem); // Move an item down to the bottom.
int MoveTo(int nItem, int nNewPosition); // Move an item to a particular position
BOOL SwapItems(int nItem1, int nItem2); // Swap two items including all attributes.

```

Item Text Operations

`CListCtrl::SetItemText` is one of the most frequently utilized functions of `CListCtrl`. However, it only accepts string parameters. So if we want to display some other data types, we must first convert them into string representation, but in this class we are freed from doing such a chore.

```

BOOL SetItemText(int nItem, int nSubItem, INT val);
BOOL SetItemText(int nItem, int nSubItem, UINT val);
BOOL SetItemText(int nItem, int nSubItem, LONG val);
BOOL SetItemText(int nItem, int nSubItem, ULONG val);
BOOL SetItemText(int nItem, int nSubItem, TCHAR val);
BOOL SetItemText(int nItem, int nSubItem, DOUBLE val, int nPrecision = -1);
BOOL SetItemText(int nItem, int nSubItem,
    const COleDateTime& dateTime,
    DWORD dwFlags = 0);

```

How to Use in a Dialog Based Application

To use the class, you need to add *ReportCtrl.h* and *ReportCtrl.cpp* into your workspace and include *ReportCtrl.h* where needed. To create an instance of `CReportCtrl`, you may either draw a `CListCtrl` on your dialog and assign a `CReportCtrl` type to it using class wizard, or manually declare a `CReportCtrl` variable and create the control window at runtime using `CReportCtrl::Create`.

How to Use in a List View

I often heard someone ask "Hey this control works great in a dialog based app, but how can I use it in a `CListView`"? Well, the answer is you don't, instead, you should let your view derive from `CFormView`, not `CListView`, then put a list control on your form view and make the control derive from `CReportCtrl`. Then add one single line of code to your view class' `OnSize` function:

```
void CMyView::OnSize(UINT nType, int cx, int cy)
{
    CFormView::OnSize(nType, cx, cy);

    // TODO: Add your message handler code here
    m_wndList.ResizeToFitParent(); // Add this line!
}
```

And that's it, whenever your view resizes, the list control will automatically resize itself to occupy the whole client area of your view, that's exactly what a `CListView` looks like.

History

Dec. 2nd, 2003

- | Initial release.

Dec. 3rd, 2003

- | Fixed a bug in `EndEdit` where item text was not properly committed.
- | Completed the implementation of the "Sort-Separator" feature.
- | Updated source and demo project file downloads.

Jan. 01, 2004

- | Fixed a bug in `SetItemData`.
- | Added message `WM_EDIT_COMMITTED` which is sent to the parent window when an item text editing is committed.
- | Fixed a bug in `SetItemText` (double type).
- | Fixed a bug where item sorting does not work properly when there are multiple `CReportCtrl` objects on same window.

= [Abin] =

Click [here](#) to view = [Abin] = 's online profile.



Discussions and Feedback

 23 comments have been posted for this article. Visit <http://www.codeproject.com/listctrl/ReportControl.asp> to post and view comments on this article.