TDD_001_Enhanced Grid Navigation

Created with NTT Coding AI Tool. Please evaluate critically.

August 1, 2025

0.1 Purpose

This MFC VC++ code implements a highly customizable grid control demo application, showcasing advanced grid features such as cell editing, drag-and-drop, clipboard support, custom cell types, and dynamic UI updates.

0.2 System Overview

This system is a Windows desktop application built using Microsoft Foundation Classes (MFC) in Visual C++. It demonstrates a feature-rich grid control, allowing users to interact with tabular data through a graphical user interface (UI). The application is structured into several high-level components:

- **UI Layer**: Handles all user interactions, displaying dialogs, grid controls, and responding to events such as clicks, edits, and menu commands. It includes dialog classes (e.g., `CGridCtrlDemoDlg`), grid windows, and custom controls for editing and navigation.
- **Business Logic Layer**: Manages the core logic for grid operations, such as editing cells, handling selection, sorting, and managing grid state. This layer is responsible for processing user actions and updating the grid accordingly.
- **Data Access Layer**: Abstracts the storage and retrieval of grid data, including cell values, formatting, and state. It provides interfaces for setting and getting cell data, supporting both normal and virtual (on-demand) data modes.
- **Integration Layer**: Facilitates integration with system features like clipboard operations, drag-and-drop, and printing. It also manages communication between the grid control and other application components, such as custom cell types and external data sources.

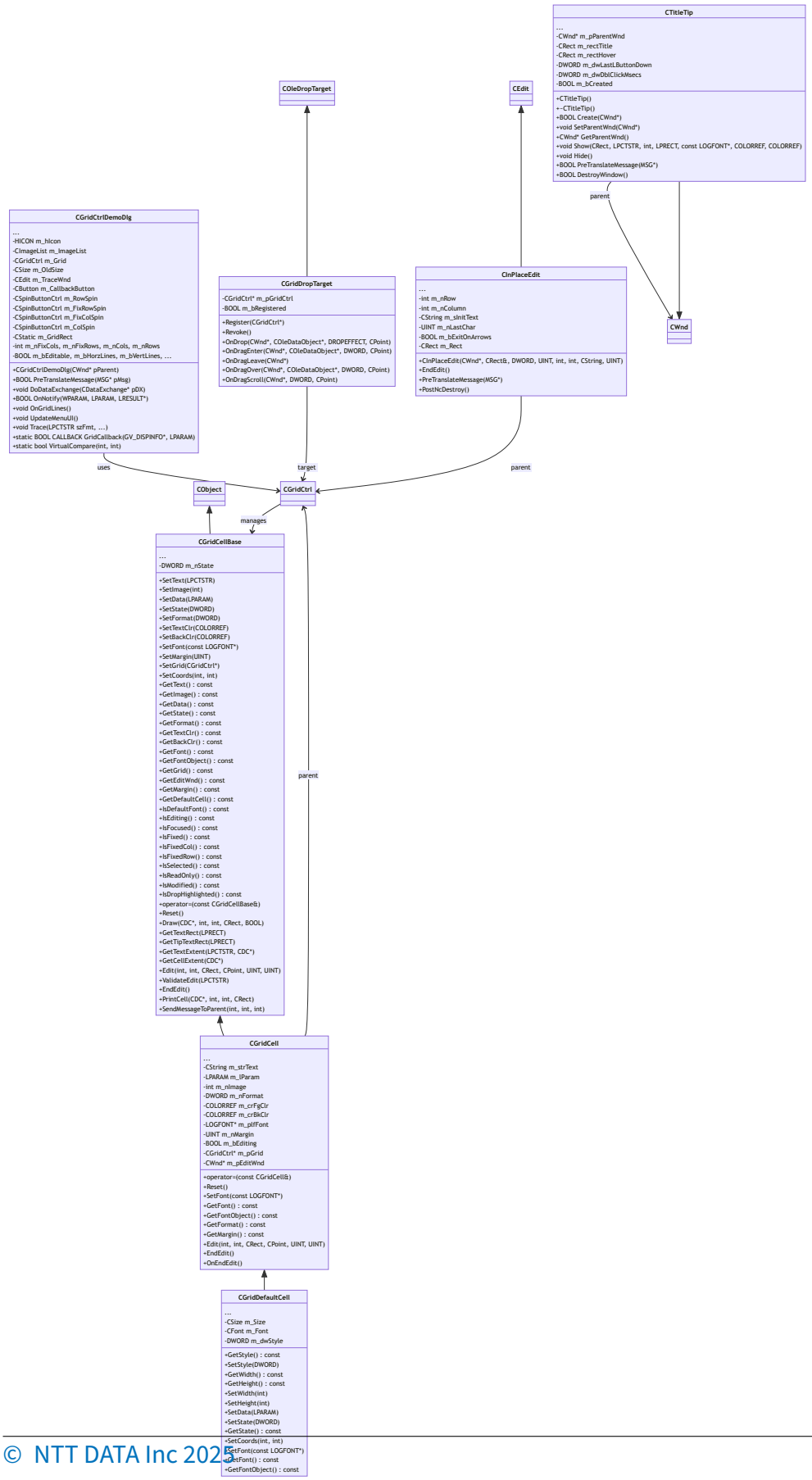
These components work together to provide a responsive, extensible, and interactive grid experience, supporting advanced features like custom cell rendering, in-place editing, and dynamic resizing.

0.3 Glossary

Term/Abbreviation	Description
MFC	Microsoft Foundation Classes, a C++ library for Windows application development.
VC++	Visual C++, Microsoft's C++ development environment.

Term/Abbreviation	Description
Grid Control	A UI component that displays and manages tabular data in rows and columns.
Dialog	A window or form used for user interaction.
Cell	The intersection of a row and column in a grid, holding data or controls.
Clipboard	System feature for copy-paste operations.
Drag-and-Drop	UI interaction for moving or copying data by dragging with the mouse.
Virtual Mode	A mode where grid data is provided on-demand, not stored in memory.
Callback	A function pointer or handler invoked in response to an event.
CDC	Device Context, used for drawing in Windows.
CWnd	MFC class representing a window.
CDialog	MFC class for dialog windows.
CEdit	MFC class for editable text controls.
CFont	MFC class for font management.
CImageList	MFC class for managing lists of images/icons.
CRect	MFC class for rectangle geometry.
GV_ITEM	Structure representing grid cell data.
NMHDR	Notification message header structure.
LRESULT	Windows message result type.
BOOL	Boolean type in Windows/MFC.
LPARAM/WPARAM	Windows message parameters.
ASSERT	Macro for debugging checks.
AFX_MSG	MFC macro for message map functions.

0.4 ControlFlowDiagram



0.5 ClassDiagram

```
%%{init: {"flowchart": { "htmlLabels": false}} }%%
classDiagram
    class CGridCtrlDemoDlg {
        +CGridCtrlDemoDlg(CWnd* pParent)
        +BOOL PreTranslateMessage(MSG* pMsg)
        +void DoDataExchange(CDataExchange* pDX)
        +BOOL OnNotify(WPARAM, LPARAM, LRESULT*)
        +void OnGridLines()
        +void UpdateMenuUI()
        +void Trace(LPCTSTR szFmt, ...)
        +static BOOL CALLBACK GridCallback(GV_DISPINFO*, LPARAM)
        +static bool VirtualCompare(int, int)
        ...
        -HICON m_hIcon
        -CImageList m_ImageList
        -CGridCtrl m_Grid
        -CSize m_OldSize
        -CEdit m_TraceWnd
        -CButton m_CallbackButton
        -CSpinButtonCtrl m_RowSpin
        -CSpinButtonCtrl m_FixRowSpin
        -CSpinButtonCtrl m_FixColSpin
        -CSpinButtonCtrl m_ColSpin
        -CStatic m_GridRect
        -int m_nFixCols, m_nFixRows, m_nCols, m_nRows
        -BOOL m_bEditable, m_bHorzLines, m_bVertLines, ...
    }
    class CGridCtrl
    class CGridCellBase {
        +SetText(LPCTSTR)
        +SetImage(int)
        +SetData(LPARAM)
        +SetState(DWORD)
        +SetFormat(DWORD)
        +SetTextClr(COLORREF)
        +SetBackClr(COLORREF)
        +SetFont(const LOGFONT*)
        +SetMargin(UINT)
        +SetGrid(CGridCtrl*)
        +SetCoords(int, int)
        +GetText() const
        +GetImage() const
        +GetData() const
```

```
        +GetState() const
        +GetFormat() const
        +GetTextClr() const
        +GetBackClr() const
        +GetFont() const
        +GetFontObject() const
        +GetGrid() const
        +GetEditWnd() const
        +GetMargin() const
        +GetDefaultCell() const
        +IsDefaultFont() const
        +IsEditing() const
        +IsFocused() const
        +IsFixed() const
        +IsFixedCol() const
        +IsFixedRow() const
        +IsSelected() const
        +IsReadOnly() const
        +IsModified() const
        +IsDropHighlighted() const
        +operator=(const CGridCellBase&)
        +Reset()
        +Draw(CDC*, int, int, CRect, BOOL)
        +GetTextRect(LPRECT)
        +GetTipTextRect(LPRECT)
        +GetTextExtent(LPCTSTR, CDC*)
        +GetCellExtent(CDC*)
        +Edit(int, int, CRect, CPoint, UINT, UINT)
        +ValidateEdit(LPCTSTR)
        +EndEdit()
        +PrintCell(CDC*, int, int, CRect)
        +SendMessageToParent(int, int, int)
        ...
        -DWORD m_nState
    }
    class CGridCell {
        +operator=(const CGridCell&)
        +Reset()
        +SetFont(const LOGFONT*)
        +GetFont() const
        +GetFontObject() const
        +GetFormat() const
        +GetMargin() const
        +Edit(int, int, CRect, CPoint, UINT, UINT)
        +EndEdit()
        +OnEndEdit()
        ...
        -CString m_strText
```

```
        -LPARAM m_lParam
        -int m_nImage
        -DWORD m_nFormat
        -COLORREF m_crFgClr
        -COLORREF m_crBkClr
        -LOGFONT* m_plfFont
        -UINT m_nMargin
        -BOOL m_bEditing
        -CGridCtrl* m_pGrid
        -CWnd* m_pEditWnd
    }
    class CGridDefaultCell {
        +GetStyle() const
        +SetStyle(DWORD)
        +GetWidth() const
        +GetHeight() const
        +SetWidth(int)
        +SetHeight(int)
        +SetData(LPARAM)
        +SetState(DWORD)
        +GetState() const
        +SetCoords(int, int)
        +SetFont(const LOGFONT*)
        +GetFont() const
        +GetFontObject() const
        ...
        -CSize m_Size
        -CFont m_Font
        -DWORD m_dwStyle
    }
    class CGridDropTarget {
        +Register(CGridCtrl*)
        +Revoke()
        +OnDrop(CWnd*, COleDataObject*, DROPEFFECT, CPoint)
        +OnDragEnter(CWnd*, COleDataObject*, DWORD, CPoint)
        +OnDragLeave(CWnd*)
        +OnDragOver(CWnd*, COleDataObject*, DWORD, CPoint)
        +OnDragScroll(CWnd*, DWORD, CPoint)
        -CGridCtrl* m_pGridCtrl
        -BOOL m_bRegistered
    }
    class CInPlaceEdit {
        +CInPlaceEdit(CWnd*, CRect&, DWORD, UINT, int, int, CString, UINT)
        +EndEdit()
        +PreTranslateMessage(MSG*)
        +PostNcDestroy()
        ...
        -int m_nRow
```

```
        -int m_nColumn
        -CString m_sInitText
        -UINT m_nLastChar
        -BOOL m_bExitOnArrows
        -CRect m_Rect
    }
    class CTitleTip {
        +CTitleTip()
        +~CTitleTip()
        +BOOL Create(CWnd*)
        +void SetParentWnd(CWnd*)
        +CWnd* GetParentWnd()
        +void Show(CRect, LPCTSTR, int, LPRECT, const LOGFONT*, COLORREF,
↪   COLORREF)
        +void Hide()
        +BOOL PreTranslateMessage(MSG*)
        +BOOL DestroyWindow()
        ...
        -CWnd* m_pParentWnd
        -CRect m_rectTitle
        -CRect m_rectHover
        -DWORD m_dwLastLButtonDown
        -DWORD m_dwDblClickMsecs
        -BOOL m_bCreated
    }

    %% Inheritance
    CGridCellBase <|-- CGridCell
    CGridCell <|-- CGridDefaultCell
    CObject <|-- CGridCellBase
    CEdit <|-- CInPlaceEdit
    COleDropTarget <|-- CGridDropTarget
    CWnd <|-- CTitleTip

    %% Associations
    CGridCtrlDemoDlg --> CGridCtrl : uses
    CGridCtrl --> CGridCellBase : manages
    CGridCell --> CGridCtrl : parent
    CGridDropTarget --> CGridCtrl : target
    CInPlaceEdit --> CGridCtrl : parent
    CTitleTip --> CWnd : parent
```

0.6 Code Breakdown

0.6.1 Main Components

- **CGridCtrlDemoDlg**: Main dialog class managing the grid control, UI controls, and event handlers.
- **CGridCtrl**: The grid control itself, managing rows, columns, and cell data.
- **CGridCellBase / CGridCell / CGridDefaultCell**: Hierarchy for grid cell representation, supporting custom cell types, formatting, and editing.
- **CGridDropTarget**: Handles drag-and-drop operations for the grid.
- **CInPlaceEdit**: Provides in-place editing for grid cells.
- **CTitleTip**: Displays tooltips for truncated cell content.

0.6.2 Inputs

- User interactions: mouse clicks, keyboard input, menu selections.
- Dialog controls: spin buttons, checkboxes, edit fields for grid configuration.
- Clipboard data (for copy/paste).
- Drag-and-drop data.

0.6.3 Outputs

- Updated grid display reflecting user actions.
- Dialog and control state changes.
- Clipboard operations (copy/cut/paste).
- Printed grid output (if enabled).
- Trace/debug output in the trace window.

0.7 Code Explanation

0.7.1 Step-by-Step Guide

1. **Application Startup**
    - The main dialog (`CGridCtrlDemoDlg`) is created and initialized.
    - Grid control and UI controls are set up, including spin buttons, checkboxes, and the trace window.
2. **Grid Initialization**

- The grid is configured with default rows, columns, and properties (editable, list mode, virtual mode, etc.).
- Image lists and custom cell types are registered.
- Menu and UI state are updated to reflect grid settings.

3. **User Interaction**
   - The message loop dispatches events to the dialog and grid.
   - Event handlers respond to user actions:
     - **Editing**: In-place editing is managed by `CInPlaceEdit`, with validation and commit/cancel logic.
     - **Selection**: Handlers update selection state and trace output.
     - **Cell Type Changes**: Handlers allow switching cell types (e.g., combo, checkbox, URL).
     - **Grid Resizing**: The grid and controls are resized dynamically.
     - **Clipboard**: Copy, cut, and paste operations are supported.
     - **Drag-and-Drop**: Managed by `CGridDropTarget`.
     - **Virtual Mode**: Grid can operate in virtual mode, fetching data on demand via callbacks.
4. **Rendering and Updates**
   - The grid and its cells are rendered using device contexts (CDC), supporting custom drawing, images, and tooltips.
   - UI and menu state are updated in response to grid changes.
5. **Advanced Features**
   - Title tips are shown for truncated cell content.
   - Printing support is available (if enabled).
   - Custom cell types (combo, numeric, date/time, checkbox, URL) are supported.

0.8 Testing

0.8.1 Unit Testing

- **Cell Classes**: Test cell creation, property setting, and editing logic.
- **Grid Control**: Test row/column management, selection, and data retrieval.
- **Custom Cell Types**: Test rendering and interaction for each custom cell type.

0.8.2 Integration Testing

- **Dialog and Grid**: Test end-to-end user interactions, including editing, selection, and resizing.
- **Clipboard and Drag-and-Drop**: Test data transfer between grid and external applications.
- **Virtual Mode**: Test large data sets and callback functionality.

0.8.3 Manual Testing

- **UI Interactions**: Manually verify all menu commands, buttons, and grid features.
- **Edge Cases**: Test with maximum/minimum rows/columns, hidden cells, and invalid input.
- **Performance**: Test responsiveness with large grids and virtual mode.

0.9 Usage Instructions

1. **Setup Development Environment**
   - Install Microsoft Visual Studio with MFC support.
   - Ensure all required header and source files are included in the project.
2. **Build the Application**
   - Open the solution/project file in Visual Studio.
   - Build the project (F7 or Build menu).
3. **Run the Application**
   - Start debugging (F5) or run the built executable.
   - The main dialog will appear with the grid control.
4. **Interact with the Grid**
   - Use spin buttons and edit fields to change grid size.
   - Use menu commands and checkboxes to toggle grid features.
   - Double-click or press Enter to edit cells.
   - Right-click for context menus.
   - Use drag-and-drop and clipboard features as needed.
5. **Advanced Features**
   - Switch between normal and virtual mode for large data sets.
   - Test custom cell types via provided buttons.
   - Use the trace window to view debug output.
6. **Configuration**
   - No special parameters are required; all configuration is done via the UI.

End of Technical Documentation