**Basics of Devextreme**

**Introduction to DevExtreme**

DevExtreme is a suite of feature-rich UI components by DevExpress that enables developers to build modern, responsive web applications. It supports jQuery, Angular, React, and Vue.

**Key Features:**

* Data visualization (charts, grids, pivot tables, etc.)
* Form elements and editors (DateBox, TextBox, SelectBox, etc.)
* Supports themes and custom styling

**Installation – NuGet Package**

Install DevExtreme in a Visual Studio 2022 project using NuGet:

1. Open **Package Manager Console** (Tools > NuGet Package Manager > Package Manager Console).
2. Run the command

**Install-Package DevExtreme.Web -Version 21.1.3**

1. Include required scripts and styles in your project:

A screen shot of a computer program

AI-generated content may be incorrect.

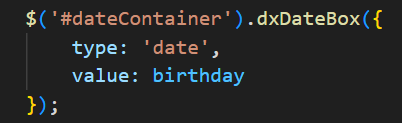
**Widget Basics – jQuery**

DevExtreme provides jQuery-based widgets that can be initialized using the $() function. Each widget has a structured API for configuration, interaction, and event handling.

**Create and Configure a Widget**

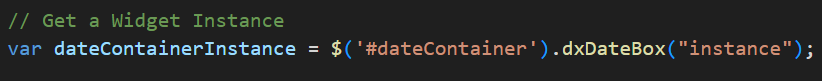
To create a widget, use jQuery to initialize it on an HTML element:





**Get a Widget Instance**

Once a widget is created, you can get its instance for further interactions:



**Get and Set Options**

Modify widget options dynamically:

A screen shot of a computer code

AI-generated content may be incorrect.

Retrieve current options:

A black background with white text

AI-generated content may be incorrect.

**Call Methods**

Invoke widget methods:

A black background with blue and green text

AI-generated content may be incorrect.

**Handle Events**

Bind event handlers to the widget:

A screen shot of a computer code

AI-generated content may be incorrect.

**Destroy a Widget**

Dispose and Remove the widget instance:

A computer code with text

AI-generated content may be incorrect.

**Editors – Overview**

**1. Check Box**

A Check Box represents a binary choice (checked or unchecked), commonly used for options like terms acceptance.

**Options:**

* value: Sets the checkbox state (true, false, undefined).
* disabled: Disables/enables the checkbox.
* elementAttr: Adds custom attributes (e.g., class, name, id) to the DOM element.
* focusStateEnabled: Enables keyboard navigation focus.
* accessKey: Assigns a keyboard shortcut (e.g., Alt + key).
* hint: Displays a tooltip on hover.
* name: Adds a name attribute (useful in forms).
* tabIndex: Sets the order for tab navigation.
* text: Displays a label alongside the checkbox.
* visible: Shows or hides the checkbox.
* rtlEnabled: Supports right-to-left languages.

**Methods:**

* instance(): Retrieves the component’s instance.
* option(property): Gets or sets a property value.
* focus(): Sets focus on the checkbox.
* reset(): Resets the value to its default.
* beginUpdate(): Suspends UI updates for batch changes.
* endUpdate(): Applies batch changes and updates the UI.
* registerKeyHandler(key, handler): Binds a custom keyboard event.
* element(): Returns the component’s root DOM element.
* dispose(): Destroys the component and removes it from the DOM.

**Events:**

* on("valueChanged", handler): Fires when the value changes.
* on("optionChanged", handler): Fires when any option is modified.
* on("focusIn", handler): Fires when the checkbox receives focus.
* on("focusOut", handler): Fires when the checkbox loses focus.
* off(eventName): Unsubscribes from an event.

**2. Date Box**

The Date Box allows date selection with multiple formats.

**Options**

* type: Defines the type of input (date, time, datetime).
* acceptCustomValue: Allows custom values.
* accessKey: Sets a keyboard shortcut (e.g., Alt + t).
* applyValueMode: Controls value submission (instantly, useButtons).
* max / min: Sets allowed date range.
* dateOutOfRangeMessage: Displays a custom error for out-of-range dates.
* disabled: Disables the input.
* disabledDates: Prevents selection of specific dates (e.g., weekends).
* elementAttr: Adds custom HTML attributes.
* height, width: Sets component dimensions.
* hint: Displays a tooltip on hover.
* inputAttr: Adds attributes to the input element.
* openOnFieldClick: Opens the calendar when clicking the field.
* placeholder: Displays a hint inside the input.
* rtlEnabled: Enables right-to-left layout.
* showClearButton: Shows a button to clear the value.
* stylingMode: Sets input style (outlined, underlined, filled).
* useMaskBehavior: Restricts input to the date format.

**Methods :**

* .instance(): Gets the component’s instance.
* .option(name, value): Gets/sets property values.
* .focus(): Sets focus to the component.
* .blur(): Removes focus from the component.
* .beginUpdate()/.endUpdate(): Suspends and resumes UI updates for batch changes.
* .getButton("clear"): Retrieves the clear button element.
* .close(): Closes the calendar popup.
* .dispose(): Destroys the component and removes it from the DOM.

**Event :**

* .on("valueChanged", handler): Fires when the date changes.
* .on("keyDown", handler): Fires when a key is pressed.
* .on("copy", handler): Fires when content is copied.
* .on("cut", handler): Fires when content is cut.
* .on("paste", handler): Fires when content is pasted.
* .off("eventName"): Unsubscribes from an event.

**3. Drop Down Box**

A Drop Down Box offers selectable options from a list.

**1. Single Selection DropDownBox**

* **items**: Static list of anime titles.
* **placeholder, height, width, hint**: Basic UI configurations.
* **showDropDownButton**: Disabled dropdown button.
* **acceptCustomValue**: Disallows custom input.
* **contentTemplate**: Renders a dxList with single selection.
  + Sets the selected value on click.
  + Closes the dropdown programmatically.

**2. Multiple Selection DropDownBox**

* **items**: List of names.
* **contentTemplate**: Renders a dxList with multiple selection.
  + Displays selected items as a comma-separated string.
* **buttons**: Adds a clear button:
  + Clears selected items in dxList.
  + Resets the dxDropDownBox value.

**Key Methods and Events:**

* .option('value', value): Updates the displayed value.
* .dxList({ selectionMode }): Manages selection types.
* .onSelectionChanged(): Triggers on item selection.
* .content().find().dxList('instance'): Accesses the list component for programmatic updates.

**4. Number Box**

A Number Box restricts input to numeric values.

**Options:**

* value: Initial value (NumberBox: numeric, Phone: string)
* format: Defines input format (e.g., "00.0" for decimals).
* step: Incremental step value for NumberBox.
* showSpinButtons, useLargeSpinButtons: Enable spin controls for NumberBox.
* placeholder: Displays hint text.
* width: Sets input width.
* showClearButton: Shows a clear button inside the input.
* validationRules: Custom validation rules (e.g., regex for phone numbers).

**Methods:**

* .option(name, value): Gets or updates component options.
* .dispose(): Removes the component from memory.
* .focus(), .blur(): Manages input focus.
* .reset(): Clears input value.

**Events:**

* onInitialized: Triggered on component creation.
* onContentReady: Fires when rendering completes.
* onDisposing: Fires on component removal.
* onFocusIn, onFocusOut: Handle focus events.
* onValueChanged: Captures input changes.
* onInput, onKeyDown, onKeyUp: Capture typing events.

**5. Select Box**

Displays selectable options with advanced search and grouping.

**Options**

* **SelectBox (Array Data Source):**
  + dataSource: Uses DevExpress.data.DataSource with array store.
  + displayExpr: Displays the name field.
  + valueExpr: Stores the id field.
  + grouped: Groups options based on anime or genre.
  + groupTemplate: Customizes group header with an icon.
  + fieldTemplate: Displays a custom text box when an item is selected.
  + itemTemplate: Customizes how items are shown in the dropdown.
* **SelectBox (Custom Store from JSON):**
  + dataSource: DevExpress.data.CustomStore with .getJSON().
  + searchEnabled: Enables search.
  + searchExpr, minSearchLength, searchMode, searchTimeout: Configure search behavior.
  + noDataText: Displays when no results are found.
  + itemTemplate: Displays item details.

**Methods**

* option(): Updates control options (e.g., searchExpr).
* repaint(): Redraws the component after option changes.
* dxSelectBox("instance"): Retrieves the component instance.

**Events**

* onValueChanged: Triggered when selection changes (e.g., grouping checkbox toggling anime/genre).
* onInput, onKeyDown, onKeyUp: Captures input and keyboard events.
* onInitialized, onContentReady, onDisposing: Lifecycle events for the components.
* onSearch: Fires during search operations (if configured).

**6. Text Area**

Multi-line text input component.

**Options:**

* **Common Options:**
  + stylingMode: Sets the style of the editor (filled, underlined, etc.).
  + placeholder: Displays placeholder text.
* **For #textArea:**
  + height: Sets a fixed height (autoResize is disabled when height is set).
* **For #textArea2:**
  + autoResizeEnabled: Enables auto resizing (false by default).
  + minHeight / maxHeight: Sets height boundaries.
  + maxLength: Limits character input.

**7. Text Box**

Single-line text input component.

**Options:**

* mask: Adds an input mask (e.g., '00/00/0000' for dates).
* maskChar: Character to represent empty spaces ('x').
* showMaskMode: Displays mask on focus or always.
* hint: Tooltip for user guidance.
* maskInvalidMessage: Custom error message for invalid input.
* mode: Input type ('password', 'email', 'text', etc.).
* buttons: Adds buttons inside the input (e.g., password toggle).
* useMaskedValue: Returns value with mask if true.
* maskRules: Custom validation for masked input (e.g., hours/minutes format)

**Methods:**

* option(): Dynamically changes properties (e.g., toggle input mode).
* dxTextBox('instance'): Gets the widget instance.

**Events:**

* **onValueChanged**: Logs previous and current values.
* **onClick** (in button): Toggles visibility for password inputs.

**8. Button**

Triggers actions on user clicks.

**Options:**

* stylingMode: Variants - 'contained', 'outlined', 'text'.
* type: Button types - 'normal', 'success', 'danger', 'default'.
* icon: Displays icons, e.g., 'preferences', custom icons like 'discord.svg'.
* text: Sets button label.

**Methods**: instance(), option(), dispose()

**Events**: onClick, onOptionChanged

**9. File Uploader**

Uploads files with drag-and-drop support.

**Options:**

* uploadUrl: Upload endpoint URL.
* abortUpload: Custom abort handler.
* selectButtonText: Button label for file selection.
* dropZone & dialogTrigger: Enable drag-and-drop and dialog triggers.
* multiple: Allow multiple file uploads.
* uploadMode: Upload trigger behavior (instantly, useButtons, useForm).
* accept: Allowed file types.
* maxFileSize, minFileSize: File size restrictions with custom messages.
* allowCanceling: Enable/disable upload cancellation.
* labelText: Label display for accepted file types.
* chunkSize: Size for chunked uploads.

**Methods & Events:**

* Lifecycle events: onBeforeSend, onUploadStarted, onProgress, onUploaded, onFilesUploaded, onUploadError, onUploadAborted.
* Drop zone interactions: onDropZoneEnter, onDropZoneLeave.
* abortUpload(): Manually stops an ongoing upload.

**10. Validation**

Ensures input accuracy according to rules.

* Options: validationRules, isValid, validationMessage
* Methods: validate(), reset(), option()
* Events: onValidated, onOptionChanged

**11. Radio Group**

Displays multiple options for single selection.

**Options:**

* dataSource: Provides review options with id, text, and color.
* itemTemplate: Customizes item display, applying color styling from itemData.color.
* layout: Displays items in a horizontal layout.

**Methods**: option(), reset(), instance()

**Events**: onValueChanged, onOptionChanged