Comprehensive Guide to VSCode Menu Contributions

This guide covers all aspects of VSCode menu customization including locations, groups, and when clauses.

1. Menu Locations

VSCode provides several context menu locations where you can add your extension's commands:

Explorer Context Menus

- `explorer/context` - File Explorer right-click menu
- `explorer/context` (with `explorerResourceIsFolder` when clause) - Folder-specific items
- `explorer/context` (with `explorerResourceIsFile` when clause) - File-specific items

Editor Context Menus

- `editor/context` - Editor right-click menu
- `editor/context` (with specific language ID) - Language-specific items
- `editor/context` (with selection context) - Items that appear only when text is selected

Editor Title Menus

- `editor/title` - Editor title area (top of editor)
- `editor/title/context` - Editor tab right-click menu

Other Menu Locations

- `view/title` - View title area (e.g., in side panels)
- `view/item/context` - Items in views (e.g., items in the Explorer)
- `commandPalette` - Commands that appear in the Command Palette
- `touchBar` - MacBook Touch Bar items
- `statusBar/context` - Status bar right-click menu
- `scm/title` - Source Control title area
- `scm/resourceGroup/context` - SCM resource groups
- `scm/resourceState/context` - SCM resource states
- `scm/change/title` - SCM change titles
- `debug/callstack/context` - Debug call stack context
- `debug/toolbar` - Debug toolbar

- `extension/context` - Extension view items
- `comments/commentThread/context` - Comment thread context
- `comments/comment/context` - Individual comment context

2. Menu Groups

Groups control the ordering and separation of menu items. Built-in groups include:

Common Groups

- `navigation` - Top of the menu (primary actions)
- `1_modification` - Middle section (modification actions)
- `9_cutcopypaste` - Bottom section (clipboard actions)
- `z_commands` - Very bottom (miscellaneous commands)

Editor-Specific Groups

- `1_diff` - Diff editor actions
- `2_lines` - Line operations
- `3_word` - Word operations
- `4_refactor` - Refactoring actions
- `5_surround` - Surround with actions
- `6_inlay` - Inlay hint actions

Explorer-Specific Groups

- `1_open` - Open actions
- `2_workspace` - Workspace actions
- `3_compare` - Compare actions
- `4_search` - Search actions
- `5_cutcopypaste` - File operations
- `6_copypath` - Path operations
- `7_upload` - Remote file operations

Custom Grouping

You can create custom groups by using numbers:

```json
{
  "group": "7_mygroup"
}
```

Lower numbers appear higher in the menu.

3. When Clauses

When clauses control the visibility and availability of menu items. They use VSCode's context keys system.

Common Context Keys

Resource States: - `explorerResourceIsFolder` - Resource is a folder - `explorerResourceIsFile` - Resource is a file - `resourceScheme` - Scheme of the resource (file, git, etc.) - `resourceLangId` - Language ID of the resource (javascript, typescript, etc.)

Editor States: - `editorTextFocus` - Editor has focus - `editorHasSelection` - Text is selected - `editorReadonly` - Editor is read-only - `editorLangId` - Language ID of active editor

Workspace States: - `workspaceFolderCount` - Number of workspace folders - `isLinux`, `isMac`, `isWindows` - Operating system - `inDebugMode` - Debug session is active

UI States: - `view == viewId` - Specific view is visible - `panelVisible` - Panel is visible - `sideBarVisible` - Sidebar is visible

Logical Operators

Combine conditions with: - `==` - Equals - `!=` - Not equals - `&&` - Logical AND - `||` - Logical OR - `!` - Logical NOT

Examples

```
"when": "resourceLangId == javascript && editorHasSelection"
"when": "explorerResourceIsFolder && workspaceFolderCount >= 2"
"when": "editorTextFocus && !editorReadonly"
```

4. Full Documentation Reference

Menu Contribution Points

```
{
  "contributes": {
    "menus": {
      "[menu location]": [
        {
          "command": "extension.commandId",
          "alt": "extension.alternateCommandId",
          "when": "context key expression",
          "group": "group name",
          "icon": {
            "light": "path/to/light/icon.svg",
            "dark": "path/to/dark/icon.svg"
          }
        }
      ],
      "submenus": [
        {
          "id": "extension.submenuId",
          "label": "Submenu Label",
          "icon": {
            "light": "path/to/light/icon.svg",
            "dark": "path/to/dark/icon.svg"
          }
        }
      ]
    }
  }
}
```

Complete Menu Contribution Schema

```
{
  "type": "object",
  "properties": {
    "menus": {
      "type": "object",
      "properties": {
        "commandPalette": {
          "type": "array",
          "description": "Contribute commands to the command
        palette",
          "items": {
            "$ref": "#/definitions/menuItem"
          }
        },
        "editor/title": {
          "type": "array",
          "description": "Contribute to the editor title menu",
          "items": {
            "$ref": "#/definitions/menuItem"
          }
        },
        // All other menu locations follow the same pattern
      },
      "additionalProperties": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/menuItem"
        }
      }
    },
    "submenus": {
      "type": "array",
      "description": "Contribute submenu items",
      "items": {
        "type": "object",
        "required": ["id", "label"],
        "properties": {
          "id": {
            "type": "string",
            "description": "Identifier of the submenu"
          },
          "label": {
            "type": "string",
            "description": "Label of the submenu"
          },
          "icon": {
            "$ref": "#/definitions/icon"
          }
        }
      }
    }
  },
  "definitions": {
    "menuItem": {
      "type": "object",
      "required": ["command"],
      "properties": {
        "command": {
          "type": "string",
          "description": "Identifier of the command to execute"
        },
        "alt": {
          "type": "string",
          "description": "Alternative command to execute on Alt-
        click"
        },
          "when": {
            "type": "string",
            "description": "Condition when the item should be
visible"
          },
          "group": {
            "type": "string",
            "description": "Group into which the item belongs"
          },
          "icon": {
            "$ref": "#/definitions/icon"
          }
        }
      },
      "icon": {
        "type": "object",
        "properties": {
          "light": {
            "type": "string",
            "description": "Icon path for light theme"
          },
          "dark": {
            "type": "string",
            "description": "Icon path for dark theme"
          }
        }
      }
    }
  }
```

5. Advanced Patterns

Dynamic Menu Items

For dynamic menu items, use the `vscode.MenuItem` API in your extension code:

```
vscode.window.registerTreeDataProvider('myView', {
  provideMenuItems(element) {
    return [
      new vscode.MenuItem('Dynamic Item',
        'extension.dynamicCommand')
    ];
  }
});
```

Submenus

1. First define the submenu:

```json
"submenus": [
  {
    "id": "extension.mySubmenu",
    "label": "My Submenu"
  }
]
```

1. Then add items to it:

```json
"menus": {
  "extension.mySubmenu": [
    {
      "command": "extension.submenuCommand1"
    },
    {
      "command": "extension.submenuCommand2"
    }
  ],
  "editor/context": [
    {
      "submenu": "extension.mySubmenu",
      "group": "navigation"
    }
  ]
}
```

6. Best Practices

1. **Use specific when clauses** to ensure your commands only appear where relevant
2. **Follow group conventions** to maintain consistent UX
3. **Provide icons** for important commands
4. **Keep menus concise** - use submenus for related but less common commands
5. **Test different contexts** to ensure your commands appear as expected

For the most up-to-date information, always refer to the official VSCode documentation on menu contributions.

Would you like me to provide specific examples for any particular scenario you have in mind?