ValueHelperWrapper Control - User Manual

Overview

The `ValueHelperWrapper` is a custom SAP UI5 control that provides an enhanced value help dialog functionality. It wraps the standard `sap.m.MultiInput` control and extends it with powerful filtering, selection, and configuration capabilities through a programmatically created value help dialog.

Features

- **Dynamic Configuration**: Fully configurable through a JSON configuration object
- **Smart Filtering**: Built-in filter bar with customizable filter fields
- **Single/Multi Selection**: Support for both single and multiple selection modes
- **Responsive Design**: Works on desktop, tablet, and mobile devices
- **Busy State Management**: Smart loading indicators with configurable delays
- **Token Management**: Full token support for selected values
- **Hidden Mode**: Can be rendered as invisible for programmatic use only

Installation

1. Replace `yourAppId` in the control definition with your actual application ID:

```
javascript
```

```javascript
return Control.extend("yourAppId.controls.ValueHelperWrapper", {
```

2. Ensure the control file is placed in your project's controls folder (e.g., `webapp/controls/ValueHelperWrapper.js`)
3. Load the control in your view or controller:

```
javascript
```

```javascript
sap.ui.define([
    "yourAppId/controls/ValueHelperWrapper"
], function(ValueHelperWrapper) {
    // Your code here
});
```

Configuration

Basic Configuration Object Structure

The control is configured through a comprehensive configuration object that defines the data source, fields, and behavior:

```javascript
var valueHelpConfig = {
    entitySet: "/ValueHelpSet",
    filters: [
        new Filter("EntitySet", FilterOperator.EQ, "Country")
    ],
    fields: [{
        code: "Vhkey",
        label: this.getResourceBundle().getText("countryId"),
        filter: true
    }, {
        code: "Text",
        label: this.getResourceBundle().getText("countryAr"),
        filter: true
    }, {
        code: "AdditionText1",
        label: this.getResourceBundle().getText("countryEn"),
        filter: true
    }],
    selectedKey: "Vhkey",
    selectedDescription: "AdditionText1"
};
```

Configuration Properties

Property	Type	Required	Description
`entitySet`	String	Yes	The OData entity set path (e.g., "/ValueHelpSet")
`filters`	Filter[]	No	Array of initial filters to apply to the data
`fields`	Object[]	Yes	Array of field definitions for table columns
`selectedKey`	String	Yes	Property name to use as the token key
`selectedDescription`	String	Yes	Property name to use as the token display text

Field Configuration

Each field object in the `fields` array supports:

Property	Type	Required	Description
`code`	String	Yes	The OData property name
`label`	String	Yes	Display label for the column header
`filter`	Boolean	No	Whether this field should appear in the filter bar

Usage Examples

1. Basic Setup in Controller

javascript

```javascript
onInit: function() {
    // Define your configuration
    var valueHelpConfig = {
        entitySet: "/ValueHelpSet",
        filters: [
            new Filter("EntitySet", FilterOperator.EQ, "Country")
        ],
        fields: [{
            code: "Vhkey",
            label: this.getResourceBundle().getText("countryId"),
            filter: true
        }, {
            code: "Text",
            label: this.getResourceBundle().getText("countryAr"),
            filter: true
        }, {
            code: "AdditionText1",
            label: this.getResourceBundle().getText("countryEn"),
            filter: true
        }],
        selectedKey: "Vhkey",
        selectedDescription: "AdditionText1"
    };

    // Create and set the view model
    this.oViewModel = new JSONModel({
        busy: false,
        delay: 100,
        countryConfig: valueHelpConfig
    });

    this.getView().setModel(this.oViewModel, "viewModel");
}
```

2. XML View Declaration

xml

```xml
<yourNamespace:ValueHelperWrapper
    id="countryValueHelper"
    config="{viewModel>/countryConfig}"
    busy="{viewModel>/busy}"
    busyIndicatorDelay="{viewModel>/delay}"
    singleMode="true"
    selectionChange="onCountrySelectionChange"
    editable="true"
    width="100%"
    placeholder="Select Country..." />
```

3. Programmatic Usage

javascript

```javascript
// Open value help dialog programmatically
onOpenValueHelp: function() {
    var oValueHelper = this.byId("countryValueHelper");
    oValueHelper.openValueHelpDialog();
},

// Handle selection changes
onCountrySelectionChange: function(oEvent) {
    var aSelectedTokens = oEvent.getParameter("selectedTokens");
    console.log("Selected tokens:", aSelectedTokens);

    // Process selected values
    aSelectedTokens.forEach(function(oToken) {
        console.log("Key:", oToken.getKey());
        console.log("Text:", oToken.getText());
    });
},

// Clear selections
onClearSelection: function() {
    var oValueHelper = this.byId("countryValueHelper");
    oValueHelper.clearTokens();
},

// Get current tokens
onGetCurrentSelection: function() {
    var oValueHelper = this.byId("countryValueHelper");
    var aTokens = oValueHelper.getTokens();
    return aTokens;
}
```

Control Properties

Custom Properties

Property	Type	Default	Description
`config`	Object	{}	Configuration object defining behavior and data
`hidden`	Boolean	false	Renders control as invisible but accessible
`singleMode`	Boolean	false	Enables single selection mode
`busy`	Boolean	false	Shows busy indicator on the table
`busyIndicatorDelay`	Integer	1000	Delay before showing busy indicator (ms)

Inherited Properties

The control inherits all properties from `sap.m.MultiInput`, including:

- `editable`
- `enabled`
- `width`
- `placeholder`
- `value`
- And many more...

Events

Custom Events

Event	Parameters	Description
`selectionChange`	`selectedTokens: sap.m.Token[]`	Fired when user confirms selection

Inherited Events

All `sap.m.MultiInput` events are supported except `valueHelpRequest` (handled internally).

Public Methods

Token Management

```javascript
// Get all tokens
var aTokens = oControl.getTokens();

// Set tokens
oControl.setTokens(aTokenArray);

// Add single token
oControl.addToken(oToken);

// Remove specific token
oControl.removeToken(oToken);

// Clear all tokens
oControl.clearTokens();
```

Dialog Management

```javascript
// Open value help dialog
oControl.openValueHelpDialog();
```

Property Management

```javascript
// Set busy state
oControl.setBusy(true);

// Set busy delay
oControl.setBusyIndicatorDelay(500);

// Set editable state
oControl.setEditable(false);
```

Hidden Property Usage

The `hidden` property is a powerful feature that allows you to render the ValueHelperWrapper control as invisible while keeping it functional. This enables you to trigger value help dialogs from other UI elements like buttons, combo boxes, or any custom controls.

Setting up Hidden Mode

```xml
<!-- Hidden ValueHelperWrapper - invisible but functional -->
<yourNamespace:ValueHelperWrapper
    id="hiddenCountryHelper"
    config="{viewModel>/countryConfig}"
    hidden="true"
    singleMode="true"
    selectionChange="onCountrySelectionChange" />

<!-- Visible trigger button -->
<Button
    id="openCountryDialogBtn"
    text="Select Country"
    press="onOpenCountryDialog"
    icon="sap-icon://value-help" />
```

Triggering from Buttons

```javascript
// Controller method to trigger value help from button
onOpenCountryDialog: function() {
    var oHiddenValueHelper = this.byId("hiddenCountryHelper");
    oHiddenValueHelper.openValueHelpDialog();
},

// Handle selection from hidden control
onCountrySelectionChange: function(oEvent) {
    var aSelectedTokens = oEvent.getParameter("selectedTokens");

    if (aSelectedTokens.length > 0) {
        var oSelectedToken = aSelectedTokens[0];

        // Update your UI with selected value
        var oButton = this.byId("openCountryDialogBtn");
        oButton.setText("Country: " + oSelectedToken.getText());

        // Store selected key for business logic
        this.getModel("viewModel").setProperty("/selectedCountryKey", oSelectedToken.getKey());
    }
}
```

Triggering from ComboBox Selection

xml

```xml
<!-- ComboBox that triggers different value helps based on selection -->
<ComboBox
    id="entityTypeCombo"
    items="{viewModel>/entityTypes}"
    selectionChange="onEntityTypeChange"
    placeholder="Select Entity Type">
    <core:Item key="{key}" text="{text}" />
</ComboBox>

<!-- Hidden value helpers for different entity types -->
<yourNamespace:ValueHelperWrapper
    id="hiddenCountryHelper"
    config="{viewModel>/countryConfig}"
    hidden="true"
    selectionChange="onEntitySelectionChange" />

<yourNamespace:ValueHelperWrapper
    id="hiddenCityHelper"
    config="{viewModel>/cityConfig}"
    hidden="true"
    selectionChange="onEntitySelectionChange" />

<yourNamespace:ValueHelperWrapper
    id="hiddenCustomerHelper"
    config="{viewModel>/customerConfig}"
    hidden="true"
    selectionChange="onEntitySelectionChange" />
```

```
// Controller logic for dynamic value help based on ComboBox selection
onEntityTypeChange: function(oEvent) {
    var sSelectedKey = oEvent.getParameter("selectedItem").getKey();
    var oValueHelper;

    switch(sSelectedKey) {
        case "country":
            oValueHelper = this.byId("hiddenCountryHelper");
            break;
        case "city":
            oValueHelper = this.byId("hiddenCityHelper");
            break;
        case "customer":
            oValueHelper = this.byId("hiddenCustomerHelper");
            break;
    }

    if (oValueHelper) {
        // Open the appropriate value help dialog
        oValueHelper.openValueHelpDialog();
    }
},

// Common handler for all entity selections
onEntitySelectionChange: function(oEvent) {
    var aSelectedTokens = oEvent.getParameter("selectedTokens");
    var oSource = oEvent.getSource();

    if (aSelectedTokens.length > 0) {
        var oToken = aSelectedTokens[0];

        // Update UI based on which value helper was used
        if (oSource.getId().includes("Country")) {
            this.getModel("viewModel").setProperty("/selectedCountry", {
                key: oToken.getKey(),
                text: oToken.getText()
            });
        } else if (oSource.getId().includes("City")) {
            this.getModel("viewModel").setProperty("/selectedCity", {
                key: oToken.getKey(),
                text: oToken.getText()
            });
        } else if (oSource.getId().includes("Customer")) {
```

```
            this.getModel("viewModel").setProperty("/selectedCustomer", {
                key: oToken.getKey(),
                text: oToken.getText()
            });
        }
    }
}
```

Multiple Hidden Value Helpers Pattern

```
// Controller setup for multiple hidden value helpers
onInit: function() {
    // Define configurations for different entities
    var oConfigs = {
        country: {
            entitySet: "/CountrySet",
            fields: [
                {code: "CountryCode", label: "Code", filter: true},
                {code: "CountryName", label: "Name", filter: true}
            ],
            selectedKey: "CountryCode",
            selectedDescription: "CountryName"
        },
        customer: {
            entitySet: "/CustomerSet",
            fields: [
                {code: "CustomerID", label: "ID", filter: true},
                {code: "CustomerName", label: "Name", filter: true},
                {code: "City", label: "City", filter: true}
            ],
            selectedKey: "CustomerID",
            selectedDescription: "CustomerName"
        },
        product: {
            entitySet: "/ProductSet",
            fields: [
                {code: "ProductCode", label: "Code", filter: true},
                {code: "ProductName", label: "Name", filter: true},
                {code: "Category", label: "Category", filter: true}
            ],
            selectedKey: "ProductCode",
            selectedDescription: "ProductName"
        }
    };

    this.oViewModel = new JSONModel({
        busy: false,
        delay: 100,
        countryConfig: oConfigs.country,
        customerConfig: oConfigs.customer,
        productConfig: oConfigs.product
    });
```

```
        this.getView().setModel(this.oViewModel, "viewModel");
    },

    // Generic method to open any hidden value helper
    openValueHelp: function(sEntityType) {
        var sHelperId = "hidden" + sEntityType.charAt(0).toUpperCase() +
                        sEntityType.slice(1) + "Helper";
        var oValueHelper = this.byId(sHelperId);

        if (oValueHelper) {
            oValueHelper.openValueHelpDialog();
        } else {
            console.error("Value helper not found: " + sHelperId);
        }
    }
```

Benefits of Hidden Mode

1. **Flexible UI Design**: Trigger value help from any UI element without being constrained by the MultiInput appearance
2. **Space Optimization**: Save screen real estate by not showing the input field
3. **Custom Interactions**: Create unique user experiences like tile-based selection or wizard-style flows
4. **Conditional Value Help**: Show different value helps based on user selections or application state
5. **Integration Flexibility**: Easily integrate with existing forms and controls

Advanced Configuration Examples

1. Multi-Selection with Custom Filters

```javascript
var multiSelectConfig = {
    entitySet: "/EmployeeSet",
    filters: [
        new Filter("Department", FilterOperator.EQ, "IT"),
        new Filter("Status", FilterOperator.EQ, "Active")
    ],
    fields: [{
        code: "EmployeeId",
        label: "Employee ID",
        filter: true
    }, {
        code: "FirstName",
        label: "First Name",
        filter: true
    }, {
        code: "LastName",
        label: "Last Name",
        filter: true
    }, {
        code: "Email",
        label: "Email",
        filter: false
    }],
    selectedKey: "EmployeeId",
    selectedDescription: "FirstName"
};
```

2. Single Selection without Initial Filters

```javascript
var singleSelectConfig = {
    entitySet: "/ProductSet",
    fields: [{
        code: "ProductCode",
        label: "Product Code",
        filter: true
    }, {
        code: "ProductName",
        label: "Product Name",
        filter: true
    }, {
        code: "Category",
        label: "Category",
        filter: true
    }],
    selectedKey: "ProductCode",
    selectedDescription: "ProductName"
};
```

Best Practices

1. **Resource Bundle Usage**: Always use resource bundles for labels to support internationalization
2. **Filter Optimization**: Only set `filter: true` for fields that users commonly search by
3. **Key Selection**: Choose unique, stable properties for `selectedKey`
4. **Description Selection**: Use human-readable properties for `selectedDescription`
5. **Busy State**: Configure appropriate `busyIndicatorDelay` based on expected response times
6. **Memory Management**: The control automatically cleans up dialog resources on close

Troubleshooting

Common Issues

1. **Control not rendering**: Ensure `yourAppId` is correctly replaced in the control definition
2. **No data showing**: Verify the `entitySet` path and any initial `filters`
3. **Columns not displaying**: Check that field `code` values match OData property names
4. **Selection not working**: Ensure `selectedKey` property exists in your data
5. **Filtering not working**: Verify that fields with `filter: true` have valid property names

Debug Tips

```javascript
// Log current configuration
console.log("Config:", oControl.getConfig());

// Check current tokens
console.log("Tokens:", oControl.getTokens());

// Monitor selection changes
oControl.attachSelectionChange(function(oEvent) {
    console.log("Selection changed:", oEvent.getParameters());
});
```

Browser Support

The control supports all browsers supported by SAP UI5, including:

- Chrome (latest)
- Firefox (latest)
- Safari (latest)
- Edge (latest)
- Internet Explorer 11+ (with UI5 compatibility)

Version Compatibility

- SAP UI5 1.60+
- Tested with UI5 versions up to 1.120+
- Compatible with both Fiori 2.0 and Fiori 3.0 themes