Руководство по UI-Генератору

Это руководство описывает, как использовать схему для генерации пользовательского интерфейса, какие типы элементов поддерживаются, и как с ними взаимодействовать.

Основная структура схемы

```
{
  "controller": {
    "name": "string",          // Уникальный идентификатор контроллера
    "display_name": "string"  // Отображаемое имя контроллера
    "visibility_head": boolean, // Видимость заголовка (h1)
    "visibilitytabs": boolean,  // Видимость левой колонки с вкладками
    "visibilitygraph": boolean  // Видимость правой колонки (предполагаемой
для графика)
  },
  "tabs": [                    // Массив вкладок
    {
      "id": "string",          // Уникальный идентификатор вкладки
      "title": "string",       // Заголовок вкладки
      "visibility": boolean,  // Видима ли вкладка (по умолчанию true)
      "groups": [              // Массив групп внутри вкладки
        {
          "id": "string",      // Уникальный идентификатор группы
          "title": "string",  // Заголовок группы
          "items": [           // Массив элементов внутри группы
            // ... элементы ...
          ]
        }
      ]
    }
  ]
}
```

Типы элементов

Элементы определяются полем "type".

1. parameter (Параметр) Представляет собой элемент, который можно отображать, изменять (в некоторых случаях) и читать.

Общие поля:

- "type": "parameter"
- "param_id": string - Уникальный идентификатор параметра.
- "display_name": string - Отображаемое имя параметра.
- "frontend_type": string - Тип отображения параметра (см. ниже).
- "units": string (опционально) - Единицы измерения, отображаются рядом с элементом.

- "frontend_props": object (опционально) - Объект с дополнительными свойствами, специфичными для типа.

Поддерживаемые frontend_type: A. number (Числовое поле ввода)

- Описание: Текстовое поле для ввода числа.
- frontend_props:
  - "min": number (опционально) - Минимальное значение.
  - "max": number (опционально) - Максимальное значение.
  - "step": number (опционально) - Шаг изменения (для клавиатуры/колеса мыши).
  - "default": number (опционально) - Значение по умолчанию.
  - "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента.

```
{
  "type": "parameter",
  "param_id": "speed",
  "display_name": "Speed",
  "frontend_type": "number",
  "units": "RPM",
  "frontend_props": { "min": 0, "max": 3000, "step": 100, "default": 1500 }
}
```

B. readonly (Только для чтения)

- Описание: Поле, отображающее значение, которое нельзя изменить через UI.
- frontend_props:
  - "default": string или number (опционально) - Значение по умолчанию.
  - "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента.

```
{
  "type": "parameter",
  "param_id": "position",
  "display_name": "Position",
  "frontend_type": "readonly",
  "units": "steps",
  "frontend_props": { "default": 500 }
}
```

C. flags (Флаги)

- Описание: Набор чекбоксов, представляющих биты числа. Пользователь может отмечать/снимать флаги, значение передается как целое число.
- frontend_props:
  - "bits": object - Объект, где ключи имеют формат "N-описание" (например, "0-ready"), а значения — отображаемые метки.
  - "default": number (опционально) - Значение по умолчанию (битовая маска).

- "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента.

```
{
  "type": "parameter",
  "param_id": "status_flags",
  "display_name": "Status Flags",
  "frontend_type": "flags",
  "frontend_props": {
    "bits": {
      "0-ready": "Ready",
      "1-running": "Running",
      "2-error": "Error"
    },
    "default": 2
  }
}
```

D. slider (Слайдер)

- Описание: Горизонтальный ползунок для выбора числового значения.
- frontend_props:
  - "min": number (опционально) - Минимальное значение.
  - "max": number (опционально) - Максимальное значение.
  - "step": number (опционально) - Шаг изменения.
  - "default": number (опционально) - Значение по умолчанию. -"show_value_display": boolean (опционально, по умолчанию true) - Показывать ли текущее значение рядом со слайдером.
  - "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента.

```
{
  "type": "parameter",
  "param_id": "volume",
  "display_name": "Volume",
  "frontend_type": "slider",
  "units": "%",
  "frontend_props": {
    "min": 0,
    "max": 100,
    "step": 1,
    "default": 50,
    "show_value_display": true
  }
}
```

E. select (Выпадающий список)

- Описание: Элемент, позволяющий выбрать одно значение из предопределенного списка.
- frontend_props:

- "options": array - Массив объектов { "value": "...", "label": "..." }, определяющих доступные опции.
- "default": string (опционально) - Значение по умолчанию (должно совпадать с одним из value).
- "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента.

profile Qwen3-Coder 3:15 pm Хорошо, давайте создадим подробное руководство по каждому элементу, его отображению, возможным параметрам и примерам использования, включая обновление и чтение данных.

Руководство по UI-Генератору Это руководство описывает, как использовать схему для генерации пользовательского интерфейса, какие типы элементов поддерживаются, и как с ними взаимодействовать.

Основная структура схемы Схема — это объект JSON, описывающий структуру UI.

json

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 ⌄ ⌄ ⌄ ⌄ ⌄ ⌄ ⌄ { "controller": { "name": "string", // Уникальный идентификатор контроллера "display_name": "string" // Отображаемое имя контроллера }, "tabs": [// Массив вкладок { "id": "string", // Уникальный идентификатор вкладки "title": "string", // Заголовок вкладки "visibility": boolean, // Видима ли вкладка (по умолчанию true) "groups": [// Массив групп внутри вкладки { "id": "string", // Уникальный идентификатор группы "title": "string", // Заголовок группы "items": [// Массив элементов внутри группы // ... элементы ...] }] }] } Типы элементов Элементы определяются полем "type".

1. parameter (Параметр) Представляет собой элемент, который можно отображать, изменять (в некоторых случаях) и читать.

Общие поля:

"type": "parameter" "param_id": string - Уникальный идентификатор параметра. "display_name": string - Отображаемое имя параметра. "frontend_type": string - Тип отображения параметра (см. ниже). "units": string (опционально) - Единицы измерения, отображаются рядом с элементом. "frontend_props": object (опционально) - Объект с дополнительными свойствами, специфичными для типа. Поддерживаемые frontend_type:

A. number (Числовое поле ввода)

Описание: Текстовое поле для ввода числа. frontend_props: "min": number (опционально) - Минимальное значение. "max": number (опционально) - Максимальное значение. "step": number (опционально) - Шаг изменения (для клавиатуры/колеса мыши). "default": number (опционально) - Значение по умолчанию. "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента. Пример: json

1 2 3 4 5 6 7 8 ⌄ { "type": "parameter", "param_id": "speed", "display_name": "Speed", "frontend_type": "number", "units": "RPM", "frontend_props": { "min": 0, "max": 3000, "step": 100, "default": 1500 } } B. readonly (Только для чтения)

Описание: Поле, отображающее значение, которое нельзя изменить через UI. frontend_props: "default": string или number (опционально) - Значение по умолчанию. "style": string (опционально) -

CSS-класс, применяемый к контейнеру элемента. Пример: json

1 2 3 4 5 6 7 8 ⌄ { "type": "parameter", "param_id": "position", "display_name": "Position", "frontend_type": "readonly", "units": "steps", "frontend_props": { "default": 500 } } C. flags (Флаги)

Описание: Набор чекбоксов, представляющих биты числа. Пользователь может отмечать/снимать флаги, значение передается как целое число. frontend_props: "bits": object - Объект, где ключи имеют формат "N-описание" (например, "0-ready"), а значения — отображаемые метки. "default": number (опционально) - Значение по умолчанию (битовая маска). "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента. Пример: json

1 2 3 4 5 6 7 8 9 10 11 12 13 14 ⌄ ⌄ ⌄ { "type": "parameter", "param_id": "status_flags", "display_name": "Status Flags", "frontend_type": "flags", "frontend_props": { "bits": { "0-ready": "Ready", "1-running": "Running", "2-error": "Error" }, "default": 2 } } D. slider (Слайдер)

Описание: Горизонтальный ползунок для выбора числового значения. frontend_props: "min": number (опционально) - Минимальное значение. "max": number (опционально) - Максимальное значение. "step": number (опционально) - Шаг изменения. "default": number (опционально) - Значение по умолчанию. "show_value_display": boolean (опционально, по умолчанию true) - Показывать ли текущее значение рядом со слайдером. "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента. Пример: json

1 2 3 4 5 6 7 8 9 10 11 12 13 14 ⌄ ⌄ { "type": "parameter", "param_id": "volume", "display_name": "Volume", "frontend_type": "slider", "units": "%", "frontend_props": { "min": 0, "max": 100, "step": 1, "default": 50, "show_value_display": true } } E. select (Выпадающий список)

Описание: Элемент, позволяющий выбрать одно значение из предопределенного списка. frontend_props: "options": array - Массив объектов { "value": "...", "label": "..." }, определяющих доступные опции. "default": string (опционально) - Значение по умолчанию (должно совпадать с одним из value). "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента. Пример: json

1 2 3 4 5 6 7 8 9 10 11 12 13 14 ⌄ ⌄ ⌄ { "type": "parameter", "param_id": "mode", "display_name": "Mode", "frontend_type": "select", "frontend_props": { "default": "auto", "options": [{ "value": "auto", "label": "Automatic" }, { "value": "manual", "label": "Manual" }, { "value": "off", "label": "Off" }] } }

F. radio (Радиокнопки)

- Описание: Набор переключателей, позволяющий выбрать одно значение из предопределенного списка.
- frontend_props:
  - "options": array - Массив объектов { "value": "...", "label": "..." }, определяющих доступные опции.
  - "default": string (опционально) - Значение по умолчанию (должно совпадать с одним из value).
  - "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента.

```
{
  "type": "parameter",
```

```
  "param_id": "color",
  "display_name": "Color",
  "frontend_type": "radio",
  "frontend_props": {
    "default": "blue",
    "options": [
      { "value": "red", "label": "Red" },
      { "value": "green", "label": "Green" },
      { "value": "blue", "label": "Blue" }
    ]
  }
}
```

2. command (Команда) Представляет собой кнопку, при нажатии на которую генерируется событие.

Поля:

- "type": "command"
- "command_id": string - Уникальный идентификатор команды.
- "display_name": string - Отображаемый текст кнопки.
- "frontend_props": object (опционально) - Объект с дополнительными свойствами для настройки кнопки.
- "style": string (опционально) - CSS-класс для неактивного (или обычного) состояния кнопки (для toggle).
- "style_active": string (опционально) - CSS-класс для активного состояния кнопки (только для toggle).
- "style_clicked": string (опционально) - CSS-класс, применяемый во время нажатия.
- "display_name_active": string (опционально) - Текст кнопки в активном состоянии (только для toggle).
- "display_name_clicked": string (опционально) - Текст кнопки во время нажатия.
- "behavior": string (опционально, по умолчанию "button") - Поведение кнопки: "button" (обычная) или "toggle" (переключатель).
- "auto_reset": boolean (опционально, по умолчанию false) - Автоматически сбрасывать style_clicked и display_name_clicked через click_duration мс.
- "click_duration": number (опционально, по умолчанию 200) - Время в мс для auto_reset.
- "tooltip": string (опционально) - Текст всплывающей подсказки при наведении.
- "style": string (опционально) - CSS-класс, применяемый к контейнеру элемента.

```
Обычная кнопка
{
  "type": "command",
  "command_id": "START",
  "display_name": "Start Motor",
  "frontend_props": {
    "style": "base",
    "style_clicked": "clicked",
    "display_name_clicked": "Starting...",
    "auto_reset": true,
```

```
    "click_duration": 500,
    "tooltip": "Starts the motor.",
    "behavior": "button"
  }
}
Toggle-кнопка
{
  "type": "command",
  "command_id": "TOGGLE_LIGHT",
  "display_name": "Turn On Light",
  "frontend_props": {
    "style": "primary",
    "style_active": "success",
    "display_name_active": "Turn Off Light",
    "style_clicked": "clicked",
    "display_name_clicked": "Toggling...",
    "auto_reset": true,
    "click_duration": 300,
    "behavior": "toggle"
  }
}
```

2. Удаление (уничтожение) UI Для корректного удаления UI и предотвращения утечек памяти (например, из-за слушателей событий) используется метод destroyUI.

```
// Уничтожить текущий UI, созданный этим generator
generator.destroyUI();
```

Этот вызов:

Вызывает метод destroy() у всех созданных TabElement и их потомков (GroupElement, ParameterElement, CommandElement). destroy() каждого элемента: Отписывает все его слушатели событий DOM (если они были добавлены через addEventListener). Отписывает его от событий EventManager. Удаляет его DOM-элемент из дерева документа. Очищает его внутренние ссылки. Удаляет корневой элемент UI из DOM. Очищает список зарегистрированных элементов в UIStateManager. После вызова destroyUI() вы можете снова вызвать generateUI с новой схемой, и старый UI будет полностью удален перед созданием нового.

3. Изменение схемы (частичное обновление) Создание/удаление элементов: Текущая архитектура не поддерживает динамическое добавление или удаление отдельных параметров, команд, групп или вкладок после первого вызова generateUI. Для этого всё равно нужно будет вызвать destroyUI() и затем generateUI() с новой схемой, которая включает или исключает нужные элементы. Обновление свойств существующих элементов: Текущая архитектура не поддерживает обновление frontend_props (например, min, max, options, style) уже созданных элементов через EventManager. Изменение visibility и enabled возможно через updateElementVisibility и UIStateManager (если добавить updateElementEnabled). Полная замена схемы: Вы можете получить новую схему с сервера и вызвать destroyUI(), а затем generateUI(новая_схема) для полной перезагрузки интерфейса.

Взаимодействие с UI

Создание и удаление

1. Создание UI Создание UI происходит при вызове метода generateUI у экземпляра UIGenerator.

```
// 1. Подготовьте схему (например, получите с сервера)
const stateSchema = {
    // ... ваша схема ...
};

// 2. Подготовьте обработчики (опционально)
const handlers = {
    onParameterChange: (paramId, value) => {
        console.log(`Параметр ${paramId} изменен на ${value}`);
    },
    onCommand: (commandId) => {
        console.log(`Команда ${commandId} выполнена`);
    }
};

// 3. Создайте экземпляр UIGenerator
const generator = new UIGenerator(stateSchema, handlers);

// 4. (Опционально) Подключите DataConnector
// const dataConnector = new DataConnector(generator.getEventManager());
// dataConnector.connect('ws://...');
// generator.setDataConnector(dataConnector);

// 5. Вызовите generateUI, передав ID DOM-элемента, в который вставлять UI
generator.generateUI('app'); // 'app' - ID элемента в HTML
```

Этот вызов:

Создает DOM-элементы для всех вкладок, групп, параметров и команд, описанных в stateSchema. Применяет логику переключения вкладок. Подписывает элементы на внутренние события (например, change для параметров, click для команд). Регистрирует все параметры и команды в UIStateManager для последующего обновления.

После создания UIGenerator, вы можете:

1. Обновить значение одного параметра

```
generator.updateParameter(paramId, value);
// Пример:
generator.updateParameter('position', 12345);
```

2. Обновить значения нескольких параметров

```
generator.updateMultipleParameters(paramsObject);
// Пример:
const data = {
    "speed": 300,
    "status_flags": 1,
    "position": 789
};
generator.updateMultipleParameters(data);
```

3. Прочитать текущее значение параметра

```
const currentValue = generator.getParameterValue(paramId);
// Пример:
const speedValue = generator.getParameterValue('speed');
console.log('Текущее значение speed:', speedValue);
```

4. Изменить видимость элемента (вкладки, группы, параметра, команды)

```
generator.updateElementVisibility(elementId, isVisible);
// Пример:
generator.updateElementVisibility('config', false); // Скрыть группу
'config'
```

Подписка на события обновления

Компонент DataConnector автоматически подписывается на события PARAMETER_VALUE_CHANGED и COMMAND_CLICKED, отправляя их на сервер по WebSocket. Вы можете реализовать аналогичную логику вручную, подписавшись на EventManager генератора:

```
const eventManager = generator.getEventManager();
eventManager.subscribe('PARAMETER_VALUE_CHANGED', (data) => {
    console.log('Параметр изменен:', data.paramId, '=', data.value);
    // Отправить на сервер
});
eventManager.subscribe('COMMAND_CLICKED', (data) => {
    console.log('Команда выполнена:', data.commandId);
    // Отправить на сервер
});
```