

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.

```
{
  "schema_version": "string", // Версия схемы (опционально, для валидации)
  "controller": {
    "name": "string",           // Уникальный идентификатор контроллера
    "display_name": "string",   // Отображаемое имя контроллера
    "layout": "row" или "column", // Ориентация размещения items (по
    // умолчанию "row")
    "visibility_head": boolean, // Видимость заголовка (h1) (по умолчанию
    // true)
    "items": [                  // Массив элементов UI
      // ... элементы (tabs, graphics) ...
    ]
  }
}
```

Изменения в **controller**:

- **visibility_head**: **boolean** - Если **false**, заголовок контроллера (**h1**) будет скрыт при генерации UI. По умолчанию **true**.

Элементы **items**

Массив **items** в **controller** может содержать элементы типа **tabs** или **graphics**.

1. **tabs** (Вкладки)

Определяет блок вкладок.

```
{
  "type": "tabs",
  "id": "string",           // Уникальный идентификатор блока вкладок
  "visibility": boolean,    // Видимость блока вкладок (по умолчанию true)
  "width": "percentage",   // Ширина блока в процентах (например, "50%")
  // опционально
  "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 (parameter, command) ...
        ]
    }
  ]
}
]
}

```

- **visibility (вкладки):** Если `false`, кнопка вкладки **не отображается**, и её содержимое недоступно для пользователя.
- **width:** Определяет **ширину этого tabs** элемента в `main-container`, если `main-container` использует `flex`.

2. graphics (Графики)

Определяет блок для отображения графиков.

```

{
  "type": "graphics",
  "id": "string",                        // Уникальный идентификатор блока графики
  "visibility": boolean,                // Видимость блока графики (по умолчанию true)
  "width": "percentage",                // Ширина блока в процентах (например, "40%")
  (опционально)
  "graphics": [                         // Массив элементов графики (обычно один элемент -
  сам график)
  {
    "id": "string",                      // Уникальный идентификатор конкретного
    графика
    "title": "string",                  // Заголовок графика
    "visibility_title": boolean,         // Видимость заголовка графика (по
    умолчанию true)
    "frontend_props": {
      "type": "line",                   // Тип диаграммы (пока поддерживается только
      "line")
      "x_axis_label": "string",          // Подпись оси X
      "y_axis_label": "string",          // Подпись оси Y
      "x_range": { "min": number, "max": number }, // Диапазон по оси X
      "y_range": { "min": number, "max": number } или "auto", // Диапазон
      по оси Y или авто-масштабирование
      "lines": [                         // Массив линий (параметров/команд),
      отображаемых на графике
        // ... элементы линий (x_constant, user_formula, real_time_data)
        ...
      ]
    }
  }
}

```

```
}  
]  
}
```

- **visibility (графики):** Если `false`, весь блок `graphics-container-wrapper` (включая заголовок и сам `canvas`) **скрывается**.
- **width:** Определяет **ширину** *этого* `graphics` элемента в `main-container`.
- **y_range: "auto"** означает, что `Chart.js` автоматически масштабирует ось Y. Объект `{ "min": ..., "max": ... }` задаёт фиксированный диапазон.
- **lines:** Определяет линии, отображаемые на графике. См. ниже.

Типы элементов в `items` (вкладки)

Элементы внутри `items` вкладок определяются полем `"type"`.

`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:

```
{  
  "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:

```
{
  "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:

```
{
  "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:

```
{
  "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:

```
{
  "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.

Пример:

```
{
  "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" }
    ]
  }
}
```

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 команды (`div.item`).

Примеры:

- 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"
  }
}
```

Типы элементов в `lines` (графики)

Элементы внутри `lines` определяют **линии**, отображаемые **на графике**.

Общие поля:

- `"id": string` - Уникальный идентификатор линии.
- `"visible": boolean` - Видима ли линия (по умолчанию `true`).
- `"type": string` - Тип линии (см. ниже).
- `"params": object` - Параметры, специфичные для типа линии.
- `"frontend_props": object` (опционально) - Объект с дополнительными свойствами.
 - `"round_precision": number` (опционально) - Для линий `real_time_data` и `user_formula`. Определяет, до какого значения округляется `x` при обновлении. Ограничивает количество *уникальных* точек в буфере. По умолчанию `1` (`Math.round`).
- `"style": object` - Стили для `Chart.js` dataset.
 - `"label": string` - Метка линии в легенде.
 - `"borderColor": string` - Цвет линии.
 - `"backgroundColor": string` - Цвет фона (для заливки).
 - `"borderWidth": number` - Толщина линии.
 - `"borderDash": array` - Паттерн пунктира [`длина_линии`, `длина_промежутка`].
 - `"pointRadius": number` - Размер точек данных.
 - `"fill": boolean` - Заливать ли область под линией.
 - `"order": number` - Порядок отрисовки (меньше - ниже).

Поддерживаемые `type`:

А. `x_constant` (Вертикальная линия)

- **Описание:** Отображает вертикальную линию на заданной позиции `x`.
- **params:**
 - `"a": number` - Координата `x`, где рисуется линия.
- **frontend_props:** Не используется.
- **style:** См. общие стили.

Пример:

```
{
  "id": "line_PLW",
  "visible": true,
  "type": "x_constant",
  "params": { "a": 20 },
}
```



```

"style": {
  "borderColor": "rgba(0, 0, 255, 0.7)",
  "borderWidth": 3,
  "borderDash": [5, 5],
  "pointRadius": 0,
  "fill": false
}
}

```

B. `user_formula` (Теоретическая кривая)

- **Описание:** Отображает кривую, вычисленную по формуле $y = f(x)$.
- **formula:** `string` - Математическое выражение, например, `"a * x * x + b * x + c"`.
Используется `math.js` для вычисления.
- **params:** `object` - Параметры, используемые в формуле (например, `{ "a": 0.1, "b": 0, "c": 10 }`).
- **frontend_props:**
 - **"round_precision":** `number` (опционально) - Точность округления `x` при вычислении точек формулы. Влияет на количество точек кривой. По умолчанию `1`.
- **style:** См. общие стили.

Пример:

```

{
  "id": "teor",
  "visible": true,
  "type": "user_formula",
  "formula": "a * x * x + b * x + c",
  "params": { "a": 0.1, "b": 0, "c": 10 },
  "frontend_props": {
    "round_precision": 0.1 // Более плавная кривая
  },
  "style": {
    "label": "Theoretical Curve",
    "borderColor": "rgba(255, 99, 132, 0.5)",
    "backgroundColor": "rgba(255, 99, 132, 0.1)",
    "borderWidth": 2,
    "borderDash": [5, 5],
    "pointRadius": 0,
    "fill": false,
    "order": 0
  }
}

```

C. `real_time_data` (Практические данные)

- **Описание:** Отображает точки данных, обновляемые в реальном времени.
- **params:** Не используется.
- **frontend_props:**

- `"round_precision": number` (опционально) - Точность округления `x` при добавлении новых точек через `updateLineData`. Ограничивает размер буфера. По умолчанию `1`.
- **style**: См. общие стили.

Пример:

```
{
  "id": "prakt",
  "visible": true,
  "type": "real_time_data",
  "params": {},
  "frontend_props": {
    "round_precision": 0.1 // Ограничивает количество точек
  },
  "style": {
    "label": "Practical Data",
    "borderColor": "rgb(75, 192, 192)",
    "backgroundColor": "rgba(75, 192, 192, 0.2)",
    "borderWidth": 2,
    "pointRadius": 3,
    "fill": false,
    "order": 1
  }
}
```

Управление UI

1. Создание UI

```
const generator = new UIGenerator(exampleSchema, handlers);
generator.generateUI('app');
```

2. Обновление параметров

- **Один параметр:**

```
generator.updateParameter('position', 12345);
```

- **Несколько параметров:**

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

- **Данные линии графика:**

```
// Добавить точку к буферу prakt
generator.updateLineData('graph', 'prakt', {x: 10, y: 25});
// Обновить весь буфер prakt
generator.updateLineData('graph', 'prakt', [{x: 5, y: 12}, {x: 15, y: 15}]);
```

- **Несколько параметров графика (пакетно):**

```
const graphUpdates = {
  "prakt": [{x: 5, y: 12}, {x: 15, y: 15}],
  "teor_params": { "a": 0.15 } // Если бы был метод
updateMultipleLineData с параметрами формул
};
// generator.updateMultipleLineData('graph', graphUpdates); // Пока не
реализовано в примере
```

3. Чтение значений

- **Значение параметра:**

```
const currentValue = generator.getParameterValue('speed');
```

- **Значение линии графика (последнее или текущее состояние буфера):**

```
const praktData = generator.getLineValue('graph', 'prakt'); // Требуется
реализации в generator и ChartRenderer
```

4. Управление видимостью

- **Элемент вкладки:**

```
generator.updateItemVisibility('config', false); // Скрыть группу 'config'
```

- **Вся вкладка:**

```
generator.updateItemVisibility('tabs_main', false); // Скрыть вкладки
```

- **Весь блок графики:**

```
generator.updateItemVisibility('graphics_main', false); // Скрыть график
```

- **Отдельная линия на графике:**

```
generator.updateLineVisibility('graph', 'teor', false); // Скрыть линию  
'teor'
```

5. Управление параметрами формулы

```
generator.updateFormulaParams('graph', 'teor', { a: 0.6, b: 12, c: 4 });
```

6. Управление заголовком контроллера

```
generator.setControllerHeaderVisibility(false); // Скрыть заголовок  
контроллера
```

7. Уничтожение UI

```
generator.destroyUI();
```

Взаимодействие с обработчиками

Обработчики (**handlers**), переданные в **UIGenerator**, вызываются при взаимодействии пользователя с UI:

- **onParameterChange(paramId, value)**: Вызывается, когда пользователь **изменяет** значение параметра (например, в **number** или **slider**).
- **onCommand(commandId)**: Вызывается, когда пользователь **нажимает** кнопку команды.

```
const handlers = {  
  onParameterChange: (paramId, value) => {  
    console.log(`Handler: Parameter ${paramId} changed to ${value}`);  
    // Отправить на сервер, обновить локальное состояние и т.д.  
  },  
  onCommand: (commandId) => {  
    console.log(`Handler: Command ${commandId} executed`);  
    // Отправить на сервер, выполнить действие и т.д.  
  }  
};
```

Подписка на внутренние события

Вы можете подписаться на события, генерируемые `EventManager` (например, `PARAMETER_VALUE_CHANGED`, `COMMAND_CLICKED`, `ELEMENT_VISIBILITY_CHANGED`), чтобы реагировать на изменения извне или от других компонентов (например, `DataConnector`).

```
const eventManager = generator.getEventManager();
eventManager.subscribe('PARAMETER_VALUE_CHANGED', (data) => {
  console.log('Event: PARAMETER_VALUE_CHANGED', data);
  // data: { paramId, value, controllerName }
});

eventManager.subscribe('COMMAND_CLICKED', (data) => {
  console.log('Event: COMMAND_CLICKED', data);
  // data: { commandId, controllerName }
});
```

Управление графиком через слайдер (Пример)

Чтобы "привязать" слайдер к обновлению данных на графике `prakt`:

1. **Создайте слайдер** (например, в HTML или динамически в `example_usage.js`).
2. **Подпишитесь** на его событие `change` или `input`.
3. **Вычислите x** (из значения слайдера) и **y** (например, по формуле или случайно).
4. **Вызовите** `generator.updateLineData('graph', 'prakt', {x: computedX, y: computedY})`.

Пример в `example_usage.js`:

```
// ... (ваш существующий код) ...

document.addEventListener('DOMContentLoaded', () => {
  // ... (ваш существующий код для чекбоксов) ...

  // --- НОВОЕ: Создаём слайдер для управления prakt ---
  const controlPanel = document.querySelector('.control-panel'); // Или
  // другой элемент

  const praktXSliderLabel = document.createElement('label');
  praktXSliderLabel.textContent = 'Prakt X Position: ';
  const praktXSlider = document.createElement('input');
  praktXSlider.type = 'range';
  praktXSlider.id = 'prakt-x-slider';
  praktXSlider.min = -30; // Используем диапазон из схемы
  praktXSlider.max = 30;
  praktXSlider.step = 1;
  praktXSlider.value = 0;
```

```
const praktXValueDisplay = document.createElement('span');
praktXValueDisplay.textContent = praktXSlider.value;

const praktYValueDisplayLabel = document.createElement('label');
praktYValueDisplayLabel.textContent = ' Prakt Y Value: ';
const praktYValueDisplay = document.createElement('span');
praktYValueDisplay.textContent = '0';

// Добавляем элементы в DOM
controlPanel.appendChild(document.createElement('br'));
controlPanel.appendChild(praktXSliderLabel);
controlPanel.appendChild(praktXSlider);
controlPanel.appendChild(praktXValueDisplay);
controlPanel.appendChild(praktYValueDisplayLabel);
controlPanel.appendChild(praktYValueDisplay);

// Обработчик изменения слайдера
praktXSlider.addEventListener('input', (e) => {
    const xValue = parseFloat(e.target.value);
    praktXValueDisplay.textContent = xValue;

    // --- ВЫЧИСЛЕНИЕ Y ---
    // Пример 1: Случайное значение в диапазоне Y графика
    const minY = exampleSchema.controller.items.find(item => item.id
=== 'graphics_main').graphics[0].frontend_props.y_range.min;
    const maxY = exampleSchema.controller.items.find(item => item.id
=== 'graphics_main').graphics[0].frontend_props.y_range.max;
    const yValue = Math.random() * (maxY - minY) + minY;

    // Пример 2: Значение по формуле, связанной с X
    // const amplitude = (maxY - minY) / 2;
    // const baseY = minY + amplitude;
    // const yValue = baseY + amplitude * Math.sin(xValue * Math.PI /
30);

    praktYValueDisplay.textContent = yValue.toFixed(2);
    // ---

    // --- ОТПРАВКА ДАННЫХ НА ГРАФИК ---
    // Обновляем *весь* буфер prakt новой точкой
    // (В реальности, возможно, вы захотите добавлять точку к
существующему буферу)
    generator.updateLineData('graph', 'prakt', {x: xValue, y: yValue});
    // ---
});

// ... (остальной код DOMContentLoaded как есть) ...
});

// ... (остальной код example_usage.js как есть) ...
```

Этот код:

1. Создаёт слайдер в `control-panel`.
2. При изменении значения слайдера:
 1. Читает `x` из слайдера.
 2. Вычисляет `y` (в примере — случайное значение в диапазоне графика).
 3. Обновляет отображение `y`.
 4. Вызывает `generator.updateLineData('graph', 'prakt', {x: xValue, y: yValue})`, что **добавляет точку** к линии `prakt` на графике `graph`.

Теперь вы можете "управлять" графиком `prakt` через слайдер, отправляя нормализованные (или вычисленные) значения `x` и `y`.