USSD Flow Editor - API Reference

Table of Contents

Flow Export API

`exportToFlowFormat(nodes, edges)`

Converts React Flow data structure to simplified backend format.

Parameters:

- `nodes` (Array): React Flow nodes array
- `edges` (Array): React Flow edges array

Returns: Array of simplified node objects

Example:

```
import { exportToFlowFormat } from './utils/flowUtils.js';

const nodes = [
  {
    id: 'start_123',
    data: {
      type: 'START',
      config: {
        ussdCode: '*123#',
        prompts: { en: 'Welcome' }
      }
    }
  }
];

const edges = [
  {
    source: 'start_123',
    target: 'menu_456',
    sourceHandle: '*123#'
  }
];

const exported = exportToFlowFormat(nodes, edges);
// Returns:
// [
//   {
//     id: 'start_123',
//     type: 'START',
//     transitions: { '*123#': 'menu_456' },
//     nextNodeType: 'MENU',
//     nextNodePrompts: { en: 'Welcome' }
//   }
// ]
```

`exportToGraphFormat(nodes, edges)`

Exports complete graph with visual properties for import.

Parameters:

- `nodes` (Array): React Flow nodes
- `edges` (Array): React Flow edges

Returns: Object with nodes, edges, and metadata

Example:

```
const graphData = exportToGraphFormat(nodes, edges);
// Returns:
// {
//   nodes: [...],
//   edges: [...],
//   timestamp: '2024-01-15T10:00:00.000Z',
//   version: '1.0'
// }
```

`importFromGraphFormat(graphData)`

Imports flow from graph format.

Parameters:

- `graphData` (Object): Graph data with nodes and edges

Returns: Object with nodes and edges arrays

Template System API

`TemplateManager.save(template)`

Saves API template to localStorage.

Parameters:

- `template` (Object): Template configuration

Returns: Template object with generated ID

Example:

```
import { TemplateManager } from './utils/TemplateManager.js';

const template = {
  name: 'Send Money API',
  requestTemplate: {
    method: 'POST',
    url: 'https://api.bank.com/transfer',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer {{sessionToken}}'
    },
    body: {
      amount: '{{AMOUNT}}',
      recipientPhone: '{{PHONE}}'
    }
  },
  joltSpec: [
    {
      operation: 'shift',
      spec: {
        'transactionId': 'txnId',
        'status': 'result'
      }
    }
  ]
};

const saved = TemplateManager.save(template);
// Returns template with _id field
```

`TemplateManager.getAll()`

Retrieves all saved templates.

Returns: Array of template objects

`TemplateManager.getById(id)`

Retrieves specific template by ID.

Parameters:

- `id` (String): Template ID

Returns: Template object or null

`TemplateManager.delete(id)`

Deletes template by ID.

Parameters:

- `id` (String): Template ID

Returns: Boolean indicating success

`TemplateManager.validate(template)`

Validates template structure.

Parameters:

- `template` (Object): Template to validate

Returns: Validation result object

Example:

```
const validation = TemplateManager.validate(template);
// Returns:
// {
//   valid: true,
//   errors: [],
//   warnings: ['Optional field missing']
// }
```

Validation API

`validateFlow(nodes, edges)`

Validates complete flow structure.

Parameters:

- `nodes` (Array): Flow nodes
- `edges` (Array): Flow edges

Returns: Validation result with errors and warnings

Example:

```
import { validateFlow } from './utils/validation.js';
```

```
const validation = validateFlow(nodes, edges);
// Returns:
// {
//   errors: [
//     { type: 'missing_start', message: 'Flow must have START node' }
//   ],
//   warnings: [
//     { type: 'orphaned_node', message: 'Node not connected', nodeId: 'node_123'
}
//   ]
// }
```

`validateNode(node)`

Validates individual node configuration.

Parameters:

- `node` (Object): Node to validate

Returns: Node validation result

`validateTemplate(template)`

Validates API template configuration.

Parameters:

- `template` (Object): Template to validate

Returns: Template validation result

Git Workflow API

Server Endpoints

`POST /api/submit-flow`

Submits flow for maker-checker review.

Request Body:

```
{
  "flowData": {
    "id": "flow_123",
    "metadata": {
      "name": "Payment Flow",
      "version": "1.0"
    },
    "nodes": [...]
  },
  "submitter": "developer_name"
}
```

Response:

```
{
  "success": true,
  "branchName": "flow-1642234567890-developer",
  "reviewUrl": "http://git-server/review/flow-1642234567890-developer"
}
```

`POST /api/approve-flow`

Approves submitted flow.

Request Body:

```
{
  "branchName": "flow-1642234567890-developer",
  "reviewer": "senior_developer",
  "comments": "Approved with minor suggestions"
}
```

Response:

```
{
  "success": true,
  "merged": true,
  "deploymentReady": true
}
```

`GET /api/pending-reviews`

Lists flows pending review.

Response:

```
{
  "reviews": [
    {
      "branchName": "flow-1642234567890-developer",
      "submitter": "developer_name",
      "submittedAt": "2024-01-15T10:00:00Z",
      "flowMetadata": {
        "name": "Payment Flow",
        "nodeCount": 5
      }
    }
  ]
}
```

Client Functions

`submitFlowForReview(flowData, submitter)`

Submits flow to git workflow server.

Parameters:

- `flowData` (Object): Complete flow data
- `submitter` (String): Developer name

Returns: Promise with submission result

K6 Test Generation API

`generateK6Test(flowData, testConfig)`

Generates K6 load testing script from flow.

Parameters:

- `flowData` (Object): Flow configuration
- `testConfig` (Object): Test parameters

Returns: K6 test script string

Example:

```
import { generateK6Test } from './utils/k6Generator.js';

const testConfig = {
  baseUrl: 'http://localhost:8080',
  stages: [
    { duration: '1m', target: 10 },
    { duration: '3m', target: 10 },
    { duration: '1m', target: 0 }
  ],
  phoneNumberPattern: '123456789{####}',
  sessionIdPattern: 'sess_{uuid}'
};

const k6Script = generateK6Test(flowData, testConfig);
```

`generateTestScenario(nodeSequence, config)`

Generates specific test scenario for node sequence.

Parameters:

- `nodeSequence` (Array): Ordered array of nodes
- `config` (Object): Scenario configuration

Returns: Test scenario object

Utility Functions API

`generateUniqueId(prefix)`

Generates unique identifier with optional prefix.

Parameters:

- `prefix` (String, optional): ID prefix

Returns: Unique string ID

Example:

```javascript
import { generateUniqueId } from './utils/flowUtils.js';

const nodeId = generateUniqueId('menu'); // 'menu_1642234567890_123'
const edgeId = generateUniqueId(); // '1642234567890_456'
```

`extractVariablesFromPrompt(prompt)`

Extracts variable placeholders from prompt text.

Parameters:

- `prompt` (String): Prompt text with :variableName syntax

Returns: Array of variable names

Example:

```javascript
const prompt = "Hello :userName, your balance is :balance";
const variables = extractVariablesFromPrompt(prompt);
// Returns: ['userName', 'balance']
```

`substituteVariables(text, variables)`

Replaces variables in text with values.

Parameters:

- `text` (String): Text with {{variableName}} placeholders
- `variables` (Object): Variable name-value pairs

Returns: Text with substituted values

Example:

```
const text = "Amount: {{AMOUNT}}, Phone: {{PHONE}}";
const variables = { AMOUNT: '100', PHONE: '1234567890' };
const result = substituteVariables(text, variables);
// Returns: "Amount: 100, Phone: 1234567890"
```

`debounce(func, delay)`

Creates debounced version of function.

Parameters:

- `func` (Function): Function to debounce
- `delay` (Number): Delay in milliseconds

Returns: Debounced function

Component API

NodeConfigPanel

Props

- `selectedNode` (Object): Currently selected node
- `onNodeConfigChange` (Function): Configuration change handler

Methods

- `updateConfig(field, value)`: Updates specific configuration field
- `resetConfig()`: Resets configuration to defaults
- `validateConfig()`: Validates current configuration

Custom Node Components

Base Props (All Node Types)

- `data` (Object): Node data containing config and metadata
- `selected` (Boolean): Selection state
- `id` (String): Unique node identifier

StartNode

- `data.config.ussdCode` (String): USSD trigger code
- `data.config.prompts` (Object): Multi-language prompts
- `data.config.defaultLanguage` (String): Default language

MenuNode

- `data.config.compositCode` (String): Menu identifier
- `data.config.prompts` (Object): Menu options by language
- `data.config.transitions` (Object): Option-to-node mappings
- `data.config.fallback` (String): Default route for invalid input

DynamicMenuNode

- `data.config.dataSource` (Object): Data source configuration
- `data.config.maxMenuItems` (Number): Maximum menu items
- `data.config.routingStrategy` (Object): Routing configuration

InputNode

- `data.config.variableName` (String): Storage variable name
- `data.config.matchPattern` (String): Input validation pattern
- `data.config.prompts` (Object): Input prompts by language

ActionNode

- `data.config.templates` (Array): API template configurations
- `data.config.responseRouting` (Object): Response-based routing
- `data.config.conditionalSQL` (String): Apache Calcite SQL for routing

EndNode

- `data.config.compositCode` (String): End point identifier
- `data.config.prompts` (Object): Final messages by language
- `data.config.promptsList` (Array): Extracted variables from prompts

Configuration Schema

Node Configuration Schema

START Node

```
interface StartConfig {
  ussdCode: string;                    // USSD trigger code (e.g., '*123#')
  prompts: {                          // Multi-language prompts
    en: string;
    es: string;
    fr: string;
    ar: string;
  };
  defaultLanguage: 'en' | 'es' | 'fr' | 'ar';
  transitions: {                    // Trigger-to-node mappings
    [trigger: string]: string;      // Node ID
  };
}
```

MENU Node

```
interface MenuConfig {
  compositCode: string;             // Unique menu identifier
  prompts: {                       // Menu options by language
    en: string;                    // Format: "1. Option\\n2. Option"
    es: string;
    fr: string;
    ar: string;
  };
  transitions: {                   // Option-to-node mappings
    [option: string]: string;      // Node ID
    fallback?: string;             // Default route
  };
}
```

DYNAMIC-MENU Node

```
interface DynamicMenuConfig {
  dataSource: {
    type: 'session' | 'api';
    sessionVariable?: string;       // Session variable name
    apiEndpoint?: string;          // API URL
    responseKey: string;           // Path to array in response
    nameField: string;             // Display field name
    idField: string;               // Unique identifier field
  };
  maxMenuItems: number;            // Maximum items to display
  routingStrategy: {
    type: 'simple' | 'conditional';
    defaultTarget?: string;        // Default node ID
    conditionalRules?: Array<{     // Conditional routing rules
      condition: string;           // JavaScript condition
      targetNode: string;          // Target node ID
    }>;
  };
}
```

INPUT Node

```
interface InputConfig {
  variableName: string;             // Session variable name
  matchPattern: string;             // Validation regex pattern
  prompts: {                       // Input prompts
    en: string;
    es: string;
    fr: string;
    ar: string;
  };
  transitions: {
    '*': string;                    // Target node for any input
  };
  validation?: {
    minLength?: number;
    maxLength?: number;
    errorMessage?: string;
  };
}
```

ACTION Node

```
interface ActionConfig {
  templates: Array<{
    _id: string;                    // Template ID
    name: string;                   // Template name
    requestTemplate: {
      method: 'GET' | 'POST' | 'PUT' | 'DELETE';
      url: string;                  // API endpoint
      headers: Record<string, string>;
      body?: any;                   // Request body
      queryParams?: Record<string, string>;
    };
    joltSpec?: Array<any>;          // JOLT transformation
  }>;
  responseRouting: {
    '200': string;                  // Success route
    '400': string;                  // Client error route
    '500': string;                  // Server error route
  };
  conditionalSQL?: string;          // Apache Calcite SQL
}
```

END Node

```
interface EndConfig {
  compositCode: string;             // Unique end identifier
  prompts: {                       // Final messages
    en: string;                    // May contain :variableName
    es: string;
    fr: string;
    ar: string;
  };
  promptsList: string[];            // Extracted variables
  sessionCleanup?: boolean;         // Clear session data
}
```

Template Schema

API Template

```
interface ApiTemplate {
  _id: string;                      // Unique identifier
  name: string;                     // Display name
  description?: string;             // Template description
  requestTemplate: {
    method: 'GET' | 'POST' | 'PUT' | 'DELETE';
    url: string;                    // API endpoint with variables
    headers: Record<string, string>; // HTTP headers
    body?: any;                     // Request body (POST/PUT)
    queryParams?: Record<string, string>;
  };
  joltSpec?: Array<{               // JOLT transformation rules
    operation: 'shift' | 'default' | 'remove' | 'cardinality';
    spec: any;                     // Operation-specific spec
  }>;
  testData?: {                     // Test configuration
    sampleRequest: any;
    expectedResponse: any;
  };
  validation?: {
    requiredFields: string[];
    optionalFields: string[];
  };
}
```

Export Format Schema

Flow Export Format

```
interface FlowExport {
  id: string;                       // Node ID
  type: 'START' | 'MENU' | 'DYNAMIC-MENU' | 'INPUT' | 'ACTION' | 'END';
  transitions: Record<string, string>; // Trigger-to-node mappings

  // Node-specific fields
  ussdCode?: string;                // START only
  compositCode?: string;            // MENU, END only
  prompts?: Record<string, string>; // All types
  storeAttribute?: string;          // INPUT only
  templateId?: string;              // ACTION only
  promptsList?: string[];           // END only

  // Metadata for connected nodes
  nextNodeType?: string;
  nextNodePrompts?: Record<string, string>;
  nextNodesMetadata?: Record<string, {
    nextNodeType: string;
    nextNodeTemplateId?: string;
    nextNodeStoreAttribute?: string;
  }>;
}
```

Graph Export Format

```
interface GraphExport {
  nodes: Array<{
    id: string;
    type: string;
    position: { x: number; y: number };
    data: {
      label: string;
      type: string;
      config: any;
    };
    measured?: { width: number; height: number };
  }>;
  edges: Array<{
    id: string;
    source: string;
    target: string;
    sourceHandle?: string;
    targetHandle?: string;
    type?: string;
    animated?: boolean;
  }>;
  timestamp: string;
  version: string;
  metadata?: {
    name: string;
    description: string;
    author: string;
  };
}
```

This API reference provides comprehensive documentation for all programmatic interfaces in the USSD Flow Editor, enabling developers to integrate with and extend the system effectively.