USSD Flow Editor - Functional Documentation

Table of Contents

Overview

The USSD Flow Editor is a visual drag-and-drop interface for creating, configuring, and managing USSD (Unstructured Supplementary Service Data) menu flows. It provides a comprehensive solution for designing complex telecommunication service flows with multi-language support, dynamic content, and API integrations.

Key Capabilities

- **Visual Flow Design**: Drag-and-drop canvas with real-time editing
- **Multi-Language Support**: English, Spanish, French, and Arabic
- **API Integration**: Template-based API configuration with JOLT transformations
- **Dynamic Content**: Session-driven dynamic menus and conditional routing
- **Export/Import**: JSON-based flow persistence and sharing
- **Testing**: K6 load testing script generation
- **Validation**: Real-time flow validation and error checking

Node Types

1. START Node 🚀

Purpose: Entry point of the USSD flow

Configuration Options:

- **USSD Dial Code**: Configure the trigger code (e.g., `*123#`)
- **Welcome Messages**: Multi-language welcome prompts
- **Default Language**: Set primary language for the flow

Left Panel Configuration:

```
{
  "ussdCode": "*123#",
  "prompts": {
    "en": "Welcome to our service",
```

```
      "es": "Bienvenido a nuestro servicio",
      "fr": "Bienvenue dans notre service",
      "ar": "مرحباً بكم في خدمتنا"
    },
    "defaultLanguage": "en"
  }
```

Connection Rules:

- **Output**: Single connection to any node type
- **Trigger**: Uses USSD code as transition key

2. MENU Node 📋

Purpose: Static menu with fixed options

Configuration Options:

- **Composite Code**: Unique identifier for the menu
- **Menu Options**: Numbered list format (1. Option 1, 2. Option 2)
- **Option Connections**: Map each option to target nodes
- **Fallback**: Default route for invalid selections

Left Panel Configuration:

```
  {
    "compositCode": "7634",
    "prompts": {
      "en": "1. Check Balance\\n2. Send Money\\n3. Pay Bills\\n4. Exit",
      "es": "1. Consultar Saldo\\n2. Enviar Dinero\\n3. Pagar Facturas\\n4. Salir"
    },
    "transitions": {
      "1": "action_balance_123",
      "2": "input_amount_456",
      "3": "menu_bills_789",
      "4": "end_goodbye_000"
    },
    "fallback": "end_error_999"
  }
```

Canvas Display:

- Shows menu options as numbered list
- Individual output handles for each option
- Composite code displayed below title in bold

3. DYNAMIC MENU Node 🔢

Purpose: Menu generated from API responses or session data

Configuration Options:

- **Data Source**: Session variable or API endpoint
- **Response Key**: Path to array in API response (e.g., "data", "result.items")
- **Name Field**: Field containing display text (e.g., "name", "title")
- **ID Field**: Field containing unique identifier
- **Max Items**: Limit number of menu options
- **Routing Strategy**: How to route user selections

Example Use Cases:

- **Bill Payment**: List of available billers from API
- **Mobile Money**: Available mobile money providers
- **Account Selection**: User's multiple accounts

Left Panel Configuration:

```
{
  "dataSource": {
    "type": "session",
    "sessionVariable": "billers_list",
    "responseKey": "data",
    "nameField": "name",
    "idField": "id"
  },
  "maxMenuItems": 8,
  "routingStrategy": {
    "type": "conditional",
    "conditionalRules": [
      {
        "condition": "item.type === 'mobile_money'",
        "targetNode": "input_phone_123"
      },
      {
        "condition": "item.type === 'utility'",
        "targetNode": "input_account_456"
      }
    ],
    "defaultTarget": "end_error_999"
  }
}
```

4. INPUT Node ⌨️

Purpose: Collect user input with validation

Configuration Options:

- **Variable Name**: Storage attribute for collected data
- **Input Validation**: Pattern matching rules

- **Error Handling**: Invalid input responses
- **Data Types**: Support for text, numbers, PIN, etc.

Validation Patterns:

- `*`: Accept any input
- `^[0-9]+$`: Numbers only
- `^[a-zA-Z]+$`: Letters only
- `^[0-9]{4}$`: 4-digit PIN
- `^[0-9]{1,2}$`: 1-2 digits

Left Panel Configuration:

```
{
  "variableName": "AMOUNT",
  "matchPattern": "^[0-9]+$",
  "prompts": {
    "en": "Enter amount to send:",
    "es": "Ingrese cantidad a enviar:"
  },
  "transitions": {
    "*": "action_validate_456"
  }
}
```

5. ACTION Node ⚡

Purpose: Execute API calls and handle responses

Configuration Options:

- **API Templates**: Multiple template support
- **Response Codes**: Handle 200, 400, 500 responses
- **Conditional Routing**: Apache Calcite SQL conditions
- **Template Creation**: AI-powered template builder
- **JOLT Transformations**: Request/response mapping

Template Structure:

```
{
  "_id": "SEND_MONEY_API",
  "name": "Send Money Template",
  "requestTemplate": {
    "method": "POST",
    "url": "https://api.bank.com/transfer",
    "headers": {
      "Content-Type": "application/json",
      "Authorization": "Bearer {{sessionToken}}"
    },
```

```
      "body": {
        "amount": "{{AMOUNT}}",
        "recipientPhone": "{{PHONE}}",
        "senderAccount": "{{ACCOUNT_ID}}"
      }
    },
    "joltSpec": [
      {
        "operation": "shift",
        "spec": {
          "transactionId": "txnId",
          "status": "result",
          "balance": "newBalance"
        }
      }
    ]
  }
```

Response Conditional Routing:

```
-- Apache Calcite SQL for conditional routing
SELECT fetchquery,
  TRIM(
    CASE
      WHEN httpCode = 200 AND status = 'SUCCESS' THEN 'success'
      WHEN httpCode = 200 AND status = 'INSUFFICIENT_FUNDS' THEN
'insufficient_funds'
      WHEN httpCode = 400 THEN 'validation_error'
      ELSE 'system_error'
    END
  ) AS matchedPath
FROM FLOWFILE
```

6. END Node 🏁

Purpose: Terminate flow and clear session

Configuration Options:

- **Composite Code**: Unique identifier for the endpoint
- **Final Messages**: Multi-language goodbye prompts
- **Variable Extraction**: Extract variables from messages using `:variableName` syntax
- **Session Cleanup**: Clear session data

Variable Extraction Example:

```
{
  "compositCode": "7633",
  "prompts": {
```

```
    "en": "Thank you :userName! Your transaction :transactionId is complete. New balance: :newBalance",
    "es": "¡Gracias :userName! Su transacción :transactionId está completa. Nuevo saldo: :newBalance"
  }
}
```

Generated promptsList:

```
{
  "promptsList": ["userName", "transactionId", "newBalance"]
}
```

Canvas Operations

Node Creation

1. **Drag from Palette**: Drag node type from left palette to canvas
2. **Auto-Positioning**: Nodes automatically position with spacing
3. **Real-time Preview**: Immediate visual feedback

Node Connection

1. **Output Handles**: Drag from output handle (right side of nodes)
2. **Input Handles**: Drop on input handle (left side of target node)
3. **Smart Handles**: Dynamic handles based on node configuration
    - MENU nodes: One handle per option (option-1, option-2, etc.)
    - ACTION nodes: One handle per response code (200, 400, 500)
    - INPUT nodes: Single handle for any input (*)

Edge Management

- **Visual Feedback**: Animated edges with color coding
- **Connection Validation**: Prevents invalid connections
- **Auto-routing**: Smart edge routing around nodes

Canvas Controls

- **Zoom**: Mouse wheel or controls panel
- **Pan**: Click and drag on empty space
- **Select**: Click nodes/edges to select
- **Multi-select**: Ctrl+click for multiple selection
- **Delete**: Delete key to remove selected items

Flow Configuration

Multi-Language Support

Each node supports 4 languages with automatic fallback:

```
{
  "prompts": {
    "en": "English text",
    "es": "Spanish text",
    "fr": "French text",
    "ar": "Arabic text"
  },
  "defaultLanguage": "en"
}
```

Connection Configuration

Connections are established both visually and through configuration:

Visual Connections:

- Drag from output handle to input handle
- Creates edge in React Flow graph

Configuration Connections:

- Stored in node's `transitions` object
- Maps trigger values to target node IDs

Example MENU node transitions:

```
{
  "transitions": {
    "1": "action_balance_check",
    "2": "input_transfer_amount",
    "3": "menu_bill_payment",
    "fallback": "end_invalid_option"
  }
}
```

Export & Import

Flow Export Format

Simplified format for backend processing:

```
[
  {
    "id": "start_1759210830016_123",
    "type": "START",
    "transitions": {
```

```
        "*123#": "menu_main_456"
      },
      "nextNodeType": "MENU",
      "nextNodePrompts": {
        "en": "1. Balance\\n2. Transfer",
        "es": "1. Saldo\\n2. Transferir"
      }
    },
    {
      "id": "menu_main_456",
      "type": "MENU",
      "compositCode": "7634",
      "transitions": {
        "1": "action_balance_789",
        "2": "input_amount_012"
      },
      "nextNodesMetadata": {
        "1": {
          "nextNodeType": "ACTION",
          "nextNodeTemplateId": "CHECK_BALANCE_API"
        },
        "2": {
          "nextNodeType": "INPUT",
          "nextNodeStoreAttribute": "AMOUNT"
        }
      }
    },
    {
      "id": "end_success_345",
      "type": "END",
      "compositCode": "7633",
      "transitions": {},
      "promptsList": ["userName", "balance", "timestamp"]
    }
  ]
```

Graph Export Format

Complete format with visual properties:

```
  {
    "nodes": [
      {
        "id": "start_123",
        "type": "start",
        "position": { "x": 100, "y": 100 },
        "data": {
          "label": "Welcome",
          "type": "START",
          "config": { /* full configuration */ }
        },
```

```
        "measured": { "width": 200, "height": 120 }
      }
    ],
    "edges": [
      {
        "id": "edge_123_456",
        "source": "start_123",
        "target": "menu_456",
        "sourceHandle": "*123#",
        "type": "smoothstep",
        "animated": true
      }
    ],
    "timestamp": "2025-09-30T10:00:00.000Z"
  }
```

Export Options

- 📤 **Export Flow**: Simplified backend format
- 🎨 **Export Graph**: Complete visual format
- 📋 **Copy to Clipboard**: Direct clipboard copy
- 💾 **Download JSON**: Save to file

Testing & Validation

Flow Validation

Automatic validation checks:

Errors ❌:

- Missing START node
- Invalid node connections
- Circular dependencies
- Orphaned nodes

Warnings ⚠️:

- Multiple START nodes
- Missing END nodes
- Unconfigured transitions

K6 Load Testing

Generate K6 scripts for flow testing:

```
// Generated K6 test script
import http from 'k6/http';
import { check, sleep } from 'k6';
```

```
export const options = {
  scenarios: {
    ussd_flow_test: {
      executor: 'ramping-vus',
      startVUs: 0,
      stages: [
        { duration: '1m', target: 10 },
        { duration: '3m', target: 10 },
        { duration: '1m', target: 0 },
      ],
    },
  },
};

const BASE_URL = 'http://localhost:8080';

export default function () {
  // Test scenario: Balance inquiry flow
  const phoneNumber = generatePhoneNumber();
  const sessionId = generateSessionId();

  // Step 1: Initiate USSD session
  const startResponse = http.post(`${BASE_URL}/ussd/session/start`,
    JSON.stringify({
      sessionId: sessionId,
      phoneNumber: phoneNumber,
      ussdCode: '*123#',
      text: ''
    }),
    { headers: { 'Content-Type': 'application/json' } }
  );

  check(startResponse, {
    'session started': (r) => r.status === 200,
  });

  sleep(1);

  // Step 2: Select balance option
  const balanceResponse = http.post(`${BASE_URL}/ussd/session/continue`,
    JSON.stringify({
      sessionId: sessionId,
      phoneNumber: phoneNumber,
      text: '1'
    }),
    { headers: { 'Content-Type': 'application/json' } }
  );

  check(balanceResponse, {
    'balance retrieved': (r) => r.status === 200 && r.body.includes('balance'),
  });
}
```

Advanced Features

AI Flow Generation

Generate flows from natural language:

Input: "Login with PIN → validate → main menu (balance/transfer) → actions → end"

Generated Flow: Complete node structure with connections

Template Creator

AI-powered API template builder:

- **cURL Import**: Parse cURL commands automatically
- **Field Extraction**: Smart field detection
- **JOLT Generation**: Auto-generate transformations
- **Validation**: Template validation and testing

Maker-Checker Workflow

Git-based approval workflow:

- **Draft Creation**: Save flows as drafts
- **Review Process**: Submit for approval
- **Version Control**: Track changes and approvals
- **Deployment**: Deploy approved flows

Auto Layout

Intelligent node positioning:

- **Hierarchical Layout**: Organize by flow levels
- **Force-Directed**: Physics-based positioning
- **Compact Layout**: Optimize for small flows
- **Custom Spacing**: Configurable node spacing

Composite Code Management

Unique identifiers for MENU and END nodes:

- **MENU Nodes**: Identify specific menu screens
- **END Nodes**: Track termination points
- **Export Integration**: Included in flow exports
- **Visual Display**: Shown on canvas in bold

This functional documentation provides a comprehensive overview of all USSD Flow Editor capabilities, from basic node operations to advanced features like AI generation and testing integration.