AccuTrack Designer User Guide

Version: 1.0
Last Updated: 2024
Platform: Windows, Linux

Table of Contents

Introduction

What is AccuTrack Designer?

AccuTrack Designer is a comprehensive SCADA (Supervisory Control and Data Acquisition) configuration tool that allows you to design, configure, and deploy industrial automation projects. It provides a complete environment for creating HMI screens, configuring data points (tags), setting up alarms, writing automation scripts, and managing communication with industrial devices.

Key Features

- **Visual HMI Design**: Create interactive operator screens with drag-and-drop components
- **Tag Management**: Centralized database for process variables and data points
- **Multi-Protocol Support**: Connect to devices via Modbus, OPC UA, and other industrial protocols

- **Lua Scripting**: Add custom logic and automation with embedded Lua scripting engine
- **Alarm Management**: Configure and manage alarm conditions and notifications
- **Security**: Role-based access control and user management
- **Historical Data**: Configure data collection and trending
- **Cross-Platform**: Runs on Windows and Linux

Getting Started

System Requirements

- **Operating System**: Windows 10/11 or Linux (Ubuntu 20.04+)
- **Qt Framework**: Qt 6.10.1 or later
- **Memory**: Minimum 4GB RAM (8GB recommended)
- **Disk Space**: 500MB for installation, additional space for projects
- **Display**: 1280x720 minimum resolution

First Launch

When you first launch AccuTrack Designer, you'll see the **Startup Page** with the following options:

1. **Recent Projects**: A list of recently opened projects (if any)
2. **New Project**: Create a new SCADA project
3. **Open Project**: Browse and open an existing project
4. **Remove**: Remove a project from the recent projects list

Creating Your First Project

1. Click **New Project** on the startup page, or use **File → New Project** from the menu
2. In the file dialog, navigate to your desired location
3. Enter a project name (e.g., "MyFirstProject")
4. Click **Save** (projects are saved with `.isc` extension)
5. The project will be created and automatically opened

The project structure will be created with the following default folders: - `Tags/` - Tag database files - `Screens/` - HMI screen templates - `Scripts/` - Lua script files - `Communication/` - Communication module configurations - `Alarms/` - Alarm configurations - `Security/` - User and security settings - `Historian/` - Historical data configurations - `Schedules/` - Scheduled task configurations

---

User Interface Overview

The AccuTrack Designer interface is organized into several key areas:

Main Window Layout

```
  Menu Bar: File | Build | Download | Upload | Help


 Project       Editor Area                 Components
 View          (Tabbed)                     Palette



  Bottom Panel: Output | Debug | Properties
```

Key Interface Components

1. Project View (Left Panel)

- **Project Tab**: Tree view of your project structure
    - SCADA projects
    - Tag tables
    - Screens
    - Scripts
    - Communication modules
    - Alarms
    - Schedules
    - Historian
    - Security
    - Machine Learning
    - Settings
- **Devices Tab**: Device discovery and management

2. Editor Area (Center)

- Tabbed interface for open editors
- Each editor opens in its own tab
- Tabs can be closed individually
- Supports multiple screens, tag tables, and scripts open simultaneously

3. Components Palette (Right Panel)

- Visual components available for HMI screens
- Organized by categories:
    - Basic Controls (buttons, inputs, etc.)

  - Industrial Components (pumps, valves, tanks)
  - Data Visualization (trends, gauges, tables)
  - Layout Components (tabs, containers)

4. Bottom Panel

- **Output Tab**: Build output and system messages
- **Debug Tab**: Debug console and diagnostic information
- **Properties Tab**: Property editor for selected components

Menu Bar

File Menu

- **New Project**: Create a new SCADA project
- **Open Project**: Open an existing project
- **Recent Projects**: Quick access to recently opened projects
- **Save**: Save current project
- **Save As**: Save project with a new name
- **Close Project**: Close the current project
- **Exit**: Exit the application

Build Menu

- **Build Project**: Compile and prepare project for deployment
- **Clean Project**: Remove build artifacts

Download Menu

- **Download to Device**: Deploy project to a target device

Upload Menu

- **Upload from Device**: Retrieve project from a device

Help Menu

- **Documentation**: Open user documentation
- **About**: Application information

Project Management

Project Structure

AccuTrack projects are organized in a hierarchical structure:

```
ProjectName.isc
  SCADA Projects/
     SCADA_Name/
        Tags/
        Screens/
        Scripts/
        Communication/
        Alarms/
        Schedules/
        Historian/
        Security/
        MachineLearning/
  Settings/
```

Creating a New Project

1. **File → New Project** (or `Ctrl+N`)
2. Choose a location and enter project name
3. Project is created with default structure
4. Project automatically opens in the Project View

Opening a Project

1. **File → Open Project** (or `Ctrl+O`)
2. Browse to the project file (`.isc`)
3. Select and open
4. Project loads in the Project View

Saving a Project

- **File → Save** (or `Ctrl+S`): Save all changes
- **File → Save As**: Save with a new name/location
- The system automatically saves:
  - Tag tables
  - Screen configurations
  - Script files
  - All module configurations

Closing a Project

1. **File → Close Project**
2. Any unsaved changes will prompt for confirmation
3. All open editors are closed

Recent Projects

The application maintains a list of recently opened projects accessible via: - **File → Recent Projects** menu - Startup page recent projects list

---

Working with Tags

Tags are the fundamental data points in your SCADA system. They represent process variables, I/O points, and computed values.

Understanding Tags

A tag is a named data point that can represent: - **Input Tags (I)**: Values read from external devices (PLCs, sensors) - **Output Tags (O)**: Values written to external devices - **Memory Tags (M)**: Internal variables for calculations and logic

Tag Data Types

Supported data types: - **Boolean**: True/False values - **Integer**: 16-bit, 32-bit, 64-bit (signed/unsigned) - **Float**: 32-bit and 64-bit floating point - **String**: Text data - **Enumeration**: Predefined value sets - **Array**: Collections of values - **Structure**: Composite data types

Creating Tags

1. In **Project View**, expand your SCADA project
2. Right-click on **Tags** → **New Tag Table** (or double-click Tags)
3. The Tag Table Editor opens
4. Click **Add Tag** or press `Insert`
5. Configure tag properties:
    - **Name**: Unique identifier (required)
    - **Description**: Human-readable description
    - **Type**: I/O type (Input/Output/Memory)
    - **Data Type**: Boolean, Integer, Float, etc.
    - **Address**: Device address (for I/O tags)
    - **Value**: Initial/default value
    - **Min/Max**: Valid range
    - **Units**: Engineering units
    - **Scaling**: Linear scaling factors
    - **Access Level**: Security access level

Tag Table Editor

The Tag Table Editor provides: - **Table View**: Spreadsheet-like interface for bulk editing - **Search/Filter**: Find tags quickly - **Validation**: Real-time validation of tag names and addresses - **Duplicate Detection**: Prevents duplicate names/addresses - **Bulk Operations**: Edit multiple tags at once

Tag Addressing

For I/O tags, addresses specify where to read/write data:

Modbus Examples: - `40001` - Holding Register 1 - `30001` - Input Register 1 - `10001` - Coil 1 - `00001` - Discrete Input 1

Bit Addressing: - `40001.0` - Bit 0 of Holding Register 1 - `40001.15` - Bit 15 of Holding Register 1

Tag Validation

The system validates: - Unique tag names within a tag table - Valid data types for addresses - Range constraints (min/max) - Address format correctness

Using Tags in Screens

Tags can be bound to screen components: 1. Select a component on a screen 2. In the **Properties** panel, find tag-related properties 3. Click the tag selector (usually a `...` button) 4. Browse and select the desired tag 5. The component will display/control the tag value

---

Creating HMI Screens

HMI (Human-Machine Interface) screens are the visual interface operators interact with.

Screen Editor

The Screen Editor provides a visual canvas for designing operator interfaces.

Opening the Screen Editor

1. In **Project View**, expand **Screens**
2. Double-click a screen, or right-click → **Open**
3. Screen opens in a new editor tab

Creating a New Screen

1. In **Project View**, right-click **Screens** → **New Screen**
2. Enter screen name and properties
3. Screen is created and opened in editor

Screen Canvas

The canvas provides: - **Grid**: Visual alignment guide (can be toggled) - **Snap to Grid**: Automatic alignment - **Zoom Controls**: Zoom in/out/fit - **Selection Tools**: Select, move, resize components - **Layers**: Z-order management for overlapping components

Adding Components

1. **From Components Palette**:
    - Click a component in the palette
    - Click on the canvas to place it
    - Component appears at click location
2. **Drag and Drop**:
    - Drag component from palette
    - Drop onto canvas
3. **Copy/Paste**:
    - Select component(s)
    - `Ctrl+C` to copy
    - `Ctrl+V` to paste

Component Properties

When a component is selected: - **Properties Panel** (bottom right) shows all properties - Properties are organized by category: - **General**: Name, position, size, visibility - **Appearance**: Colors, fonts, borders, styles - **Data**: Tag bindings, expressions - **Behavior**: Events, animations, interactions - **Security**: Access levels, permissions

Common Component Operations

Moving Components

- Click and drag to move
- Use arrow keys for fine positioning
- Hold `Shift` for faster movement

Resizing Components

- Click and drag corner/edge handles
- Hold `Shift` to maintain aspect ratio
- Use Properties panel for precise dimensions

Aligning Components

- Select multiple components (`Ctrl+Click`)
- Use alignment tools (if available)
- Use grid snap for manual alignment

Deleting Components

- Select component(s)
- Press `Delete` key or right-click → Delete

Screen Properties

Each screen has properties: - **Name**: Screen identifier - **Size**: Width and height in pixels - **Background**: Background color or image - **Security**: Required access level to view screen - **Navigation**: Screen navigation settings

Screen Templates

Screens can be used as templates: - Create base screens with common elements - Inherit from templates - Override template properties as needed

Saving Screens

- Screens are saved automatically when you save the project
- **File → Save** saves all open screens
- Individual screens can be saved via right-click menu

Using Components

Components are the building blocks of HMI screens. They provide visual representation and interaction capabilities.

Component Categories

1. Basic Controls **Buttons** - Standard push buttons - Toggle buttons - Radio buttons - Checkboxes - Properties: Text, icon, colors, events

Text Input - Single-line text input - Multi-line text area - Numeric input (with validation) - Password input - Properties: Placeholder, validation, format

Selection Controls - ComboBox (dropdown) - ListBox - RadioButton groups - CheckBox groups - Properties: Options, default selection

Sliders and Spinners - Horizontal/vertical sliders - Numeric spinners - Properties: Min/max, step, orientation

2. Industrial Components **Pumps** - Visual pump representation - States: Running, Stopped, Fault - Properties: Colors, animations, tag bindings

Motors - Motor visualization - Speed indication - Status indicators

Tanks - Tank level visualization - Fill percentage - High/low level indicators - Properties: Capacity, units, colors

Valves - Valve position visualization - Open/closed states - Flow direction indicators

Conveyors - Conveyor belt visualization - Direction and speed - Status indicators

3. Data Visualization **TrendView** - Real-time trending - Historical trending - Multiple tag support - Zoom, pan, cursor inspection - Properties: Tags, time range, colors, axes

GaugeView - Circular gauges - Linear gauges - Digital displays - Properties: Min/max, units, colors, ranges

AlarmView - Alarm list display - Alarm status indicators - Alarm history - Properties: Filter, sort, acknowledge actions

TableView - Data grid display - Tag value tables - Sortable columns - Properties: Columns, data source, formatting

4. Layout Components **TabComponent** - Multi-page layouts - Tab navigation - Properties: Tab labels, pages, styling

PopupComponent - Modal dialogs - Overlay windows - Properties: Size, position, modal behavior

Container Components - Group boxes - Panels - Scroll areas - Properties: Layout, borders, backgrounds

Adding Components to Screens

1. **Select Component**: Click component in Components Palette
2. **Place on Canvas**: Click where you want to place it
3. **Configure Properties**: Use Properties panel
4. **Bind to Tags**: Set tag bindings in Data properties
5. **Test**: Use preview mode (if available)

Component Tag Binding

Most components can be bound to tags:

1. Select component
2. In Properties panel, find **Tag** or **Data Source** property
3. Click tag selector (... button)
4. Browse tag tables and select tag
5. Component will display/control tag value

Binding Types: - **Read**: Display tag value (read-only) - **Write**: Control tag value (write-only) - **Read/Write**: Both display and control

Component Events

Components can trigger events: - **Click**: Button press, component click - **Value Changed**: Input value change - **Mouse Enter/Leave**: Hover events - **Focus**: Focus gained/lost

Events can be connected to: - Scripts - Screen navigation - Tag writes - Other component actions

Component Styling

Components support extensive styling: - **Colors**: Background, foreground, border - **Fonts**: Font family, size, style - **Borders**: Width, style, radius - **Shadows**: Shadow effects - **Animations**: Transitions, state changes

Component Libraries

Components are organized in libraries: - Access via Components Palette - Search functionality - Category filtering - Custom components can be added

---

Scripting with Lua

AccuTrack Designer includes an embedded Lua scripting engine (v5.4.7) for custom logic and automation.

Script Editor

The Script Editor provides: - **Syntax Highlighting**: Lua syntax coloring - **Code Completion**: Auto-completion support - **Error Marking**: Visual error indicators - **Line Numbers**: Easy navigation - **Search/Replace**: Find and replace functionality

Creating Scripts

1. In **Project View**, expand **Scripts**
2. Right-click → **New Script**
3. Enter script name
4. Script Editor opens
5. Write your Lua code
6. Save script (automatically saved with project)

Lua Script Structure

```
-- Script: MyScript
-- Description: Example script

-- Global variables
```

```lua
local tagValue = 0

-- Function definitions
function onStartup()
    -- Code executed at startup
    print("Script started")
end

function onTagChanged(tagName, newValue)
    -- Code executed when tag changes
    tagValue = newValue
    print("Tag " .. tagName .. " changed to " .. newValue)
end

function onTimer()
    -- Code executed periodically
    -- Access tags, perform calculations
end
```

Accessing Tags from Scripts

The Lua bridge provides access to tags:

```lua
-- Read tag value
local value = getTag("TagName")

-- Write tag value
setTag("TagName", 100)

-- Check tag quality
local quality = getTagQuality("TagName")
```

Script Execution Contexts

Scripts can execute in different contexts:

1. **Event-Triggered**: Execute when specific events occur
    - Tag value changes
    - Button clicks
    - Timer events
    - Alarm conditions
2. **Periodic**: Execute at regular intervals
    - Configure scan rate
    - Continuous monitoring
    - Background processing
3. **Startup/Shutdown**: Execute once
    - System initialization

- Cleanup operations

Script Functions

Common script functions available:

```
-- Tag operations
getTag(name)                -- Read tag value
setTag(name, value)         -- Write tag value
getTagQuality(name)         -- Get tag quality

-- Alarm operations
acknowledgeAlarm(id)        -- Acknowledge alarm
shelveAlarm(id)             -- Shelve alarm

-- System functions
print(message)              -- Print to console
log(message, level)         -- Log message
sleep(ms)                   -- Sleep for milliseconds

-- Math and utilities
math.sin(x)                 -- Standard Lua math
math.cos(x)
string.format(...)          -- String formatting
```

Script Scheduling

Scripts can be scheduled via the Schedules module: 1. Create a schedule 2. Select script to execute 3. Configure timing (daily, weekly, etc.) 4. Set execution parameters

Script Debugging

Debugging features: - **Console Output**: Use `print()` to output messages - **Error Messages**: Syntax and runtime errors displayed - **Breakpoints**: (If supported) Set breakpoints - **Step Through**: (If supported) Step through execution

Script Best Practices

1. **Error Handling**: Always handle errors

   ```
   local success, result = pcall(function()
       -- Risky code
   end)
   ```

2. **Performance**: Keep scripts efficient

   - Avoid tight loops

    - Use appropriate scan rates
    - Minimize tag reads/writes

3. **Documentation**: Comment your code

   ```
   -- Purpose: Calculate average temperature
   -- Author: Your Name
   -- Date: 2024-01-01
   ```

4. **Modularity**: Break complex logic into functions

5. **Testing**: Test scripts thoroughly before deployment

Configuring Alarms

Alarms notify operators of abnormal conditions and events in the process.

Alarm Types

Digital Alarms

- Trigger on Boolean tag state changes
- ON alarm: Triggers when tag becomes true
- OFF alarm: Triggers when tag becomes false

Analog Alarms

- Trigger on numeric tag threshold violations
- **High**: Tag exceeds high limit
- **Low**: Tag falls below low limit
- **High-High**: Critical high condition
- **Low-Low**: Critical low condition

Deviation Alarms

- Trigger when tag deviates from setpoint
- Deadband to prevent oscillation

Rate-of-Change Alarms

- Trigger on rapid value changes
- Prevents sudden spikes/drops

Creating Alarms

1. In **Project View**, expand **Alarms**
2. Right-click → **Open Alarms Editor** (or double-click)
3. Click **Add Alarm**

4. Configure alarm properties:

Basic Properties: - **Name**: Unique alarm identifier - **Description**: Human-readable description - **Tag**: Tag to monitor - **Type**: Digital, Analog, etc.

Condition Properties: - **Condition**: Trigger condition - **Threshold**: Threshold value (for analog) - **Deadband**: Hysteresis to prevent oscillation - **Delay**: Time delay before triggering

Priority and Severity: - **Priority**: 1-100 (higher = more important) - **Severity**: Critical, High, Medium, Low - **Category**: Alarm category for grouping

Behavior: - **Latching**: Alarm remains active until acknowledged - **Auto-Acknowledge**: Automatically acknowledge when condition clears - **Shelving**: Allow temporary suppression

Alarm Editor

The Alarm Editor provides: - **Alarm List**: Table of all alarms - **Filtering**: Filter by type, priority, status - **Search**: Find alarms quickly - **Bulk Edit**: Edit multiple alarms - **Validation**: Verify alarm configuration

Alarm States

Alarms can be in various states: - **Normal**: Condition not met, no alarm - **Active**: Condition met, alarm active - **Acknowledged**: Alarm acknowledged by operator - **Shelved**: Temporarily suppressed - **Cleared**: Condition cleared, alarm inactive

Alarm Actions

Alarms can trigger actions: - **Notifications**: Display messages, sounds - **Scripts**: Execute Lua scripts - **Tag Writes**: Set tag values - **Screen Navigation**: Navigate to specific screens - **Logging**: Log to historian

Alarm History

Alarm events are logged: - **Event Log**: All alarm state changes - **Timestamp**: When events occurred - **Operator**: Who acknowledged (if applicable) - **Retention**: Configurable retention period

Alarm Display

Alarms are displayed in: - **Alarm View Component**: On HMI screens - **Alarm List**: Tabular display - **Alarm Banner**: Top-of-screen summary - **Alarm Summary**: Summary screen

---

Communication Setup

The Communication module configures connections to external devices (PLCs, sensors, etc.).

Supported Protocols

- **Modbus TCP/RTU**: Industrial standard protocol
- **OPC UA/DA**: OPC Foundation protocols
- **Custom Protocols**: Extensible protocol support

Communication Module Editor

1. In **Project View**, expand **Communication**
2. Double-click **Communication Modules** to open editor
3. Configure communication settings

Modbus Configuration

Creating a Modbus Connection

1. Click **Add Connection**
2. Select **Modbus TCP** or **Modbus RTU**
3. Configure connection properties:

Modbus TCP: - **Name**: Connection identifier - **IP Address**: Device IP address - **Port**: TCP port (default 502) - **Unit ID**: Modbus unit identifier - **Timeout**: Communication timeout (ms) - **Retry Count**: Number of retry attempts

Modbus RTU: - **Name**: Connection identifier - **Port**: Serial port (COM1, /dev/ttyUSB0, etc.) - **Baud Rate**: 9600, 19200, 38400, etc. - **Data Bits**: 7 or 8 - **Parity**: None, Even, Odd - **Stop Bits**: 1 or 2 - **Unit ID**: Modbus unit identifier

Modbus Scanner The Modbus Scanner helps discover devices: 1. Open **Device Manager** tab 2. Click **Scan for Devices** 3. Configure scan parameters 4. Start scan 5. Discovered devices appear in list 6. Add to communication configuration

OPC UA Configuration

Creating an OPC UA Connection

1. Click **Add Connection**
2. Select **OPC UA**
3. Configure connection:

Connection Properties: - **Name**: Connection identifier - **Endpoint URL**: OPC UA server endpoint - **Security Policy**: Security policy (None, Basic128Rsa15, etc.) - **Security Mode**: Security mode (None, Sign, SignAndEncrypt) - **Username/Password**: Authentication credentials - **Certificate**: Client certificate (if required)

Node Browsing: 1. Click **Browse Nodes** 2. Connect to OPC UA server 3. Browse server address space 4. Select nodes to map to tags 5. Create tag mappings

Tag Mapping

After configuring communication: 1. Map device addresses to tags 2. In Tag Editor, set tag addresses 3. Addresses reference communication connections 4. Example: `ModbusConnection1:40001`

Connection Status

Monitor connection status: - **Connected**: Active connection - **Disconnected**: No connection - **Error**: Connection error - **Retrying**: Attempting to reconnect

Communication Diagnostics

Diagnostic tools: - **Traffic Monitor**: View communication traffic - **Error Log**: Communication errors - **Performance Metrics**: Response times, throughput - **Status Indicators**: Visual connection status

---

Security Configuration

The Security module manages user accounts, roles, and permissions.

Security Editor

1. In **Project View**, expand **Security**
2. Double-click **Security** to open editor
3. Configure users, groups, and roles

User Management

Creating Users

1. Click **Users** tab
2. Click **Add User**
3. Configure user properties:

User Properties: - **Username**: Unique username (required) - **Full Name**: User's full name - **Email**: Email address - **Password**: User password (SHA-256 hashed) - **Active**: Enable/disable user account - **Roles**: Assigned roles - **Groups**: Group memberships

User Roles Roles define permission sets: - **Administrator**: Full system access - **Operator**: Operational access, limited configuration - **Viewer**: Read-only access - **Engineer**: Configuration access - **Custom Roles**: User-defined roles

Group Management

Groups organize users and inherit permissions:

1. Click **Groups** tab
2. Click **Add Group**
3. Configure group:
    - **Name**: Group identifier
    - **Description**: Group description
    - **Parent Group**: Hierarchical grouping
    - **Members**: Users in group
    - **Roles**: Roles assigned to group

Permission System

Permissions control access to resources:

Permission Types: - **View**: Read-only access - **Control**: Write/control access - **Configure**: Configuration access - **Admin**: Administrative access

Resource Types: - Tags - Screens - Scripts - Alarms - Communication - System settings

Access Levels

Tags and components can have access levels: - **Public**: No authentication required - **Operator**: Operator role or higher - **Engineer**: Engineer role or higher - **Administrator**: Administrator only

Authentication

Users authenticate with: - **Username/Password**: Standard authentication - **Session Management**: Active session tracking - **Timeout**: Automatic logout after inactivity

Audit Trail

Security events are logged: - **Login/Logout**: User authentication events - **Permission Denials**: Access denied events - **Configuration Changes**: Security configuration changes - **User Actions**: Significant user actions

---

Schedules and Automation

Schedules automate script execution and tasks.

Schedule Editor

1. In **Project View**, expand **Schedules**
2. Double-click **Schedules** to open editor
3. Create and manage schedules

Schedule Types

One-Time Schedule

- Execute once at specified time
- Use for: Initialization, one-time tasks

Daily Schedule

- Execute daily at specified time(s)
- Use for: Daily reports, maintenance tasks

Weekly Schedule

- Execute on specific days of week
- Use for: Weekly summaries, recurring tasks

Monthly Schedule

- Execute on specific days of month
- Use for: Monthly reports, billing cycles

Creating Schedules

1. Click **Add Schedule**
2. Configure schedule properties:

Basic Properties: - **Name**: Schedule identifier - **Description**: Schedule description - **Type**: One-Time, Daily, Weekly, Monthly - **Enabled**: Enable/disable schedule

Timing Properties: - **Start Time**: When to start execution - **End Time**: When to stop (if applicable) - **Scan Cycle**: Execution interval (ms) - **Days**: Days of week/month (for weekly/monthly)

Execution Properties: - **Script**: Lua script to execute - **Parameters**: Script parameters - **Priority**: Execution priority (1-100) - **Retry**: Retry on failure - **Max Execution Time**: Timeout limit

Script Selection

When creating a schedule: 1. Click **Select Script** 2. Browse available scripts 3. Select script 4. Configure script parameters (if any) 5. Script executes at scheduled times

Schedule Status

Monitor schedule status: - **Active**: Schedule is running - **Inactive**: Schedule is disabled - **Executing**: Currently running - **Error**: Execution error occurred - **Completed**: Execution finished

Schedule History

View execution history: - **Execution Log**: Past executions - **Success/Failure**: Execution results - **Execution Time**: When executed - **Duration**: How long execution took

---

Historical Data (Historian)

The Historian module configures data collection and storage for trending and analysis.

Historian Editor

1. In **Project View**, expand **Historian**
2. Double-click **Historian** to open editor
3. Configure data collection

Data Collection Configuration

Tag Selection

1. Select tags to collect
2. Tags can be added individually or in groups
3. Configure collection settings per tag

Collection Settings **Sampling Rate:** - **Periodic**: Sample at fixed interval - **On Change**: Sample when value changes - **On Change with Deadband**: Sample on significant change - **Rate of Change**: Sample based on rate of change

Storage Settings: - **In-Memory Buffer**: Circular buffer size - **Persistent Storage**: Archive to disk - **Compression**: Data compression settings - **Retention**: How long to keep data

Deadband Configuration

Deadband prevents excessive data collection: - **Absolute Deadband**: Value must change by fixed amount - **Percent Deadband**: Value must change by percentage - **Rate Deadband**: Rate of change threshold

Data Storage

Storage Options: - **Circular Buffer**: In-memory ring buffer - **File Archive**: Persistent file storage - **Database**: Database storage (if configured) - **Compression**: Compress archived data

Retention Policy: - **Time-Based**: Keep data for X days - **Size-Based**: Keep until storage limit - **Automatic Cleanup**: Remove old data

Data Retrieval

Retrieve historical data for: - **Trending**: Display in TrendView components - **Reports**: Generate reports - **Analysis**: Data analysis - **Export**: Export to external formats

Integration with Trends

Historical data is used by TrendView components: 1. Configure TrendView on screen 2. Select tags to trend 3. Set time range (real-time or historical) 4. TrendView displays data from historian

---

Building and Deploying

Before deploying to Runtime, projects must be built (compiled).

Building a Project

1. **Build → Build Project** (or `Ctrl+B`)
2. Build process:
    - Validates project configuration
    - Compiles resources
    - Generates deployment package
    - Creates QML files for Runtime
    - Packages all assets
3. Monitor build progress in **Output** tab
4. Build errors displayed in **Output** tab

Build Output

Build creates: - **Deployment Package**: Complete project package - **QML Files**: Runtime visualization files - **JSON Configuration**: Project configuration - **Resource Files**: Images, fonts, etc.

Build Validation

Build process validates: - **Tag References**: All tag references valid - **Screen References**: All screen references valid - **Script Syntax**: Lua scripts compile - **Communication Config**: Valid communication settings - **Security Config**: Valid security settings

Build Errors

If build fails: 1. Check **Output** tab for errors 2. Fix reported issues 3. Rebuild project 4. Common issues: - Invalid tag references - Missing files - Syntax errors in scripts - Invalid addresses

Cleaning a Project

Build → Clean Project: - Removes build artifacts - Cleans temporary files - Prepares for fresh build

Deployment to Device

Download to Device

1. **Download → Download to Device**
2. Select target device
3. Configure deployment settings
4. Transfer project package
5. Verify deployment

Device Requirements Target device must have: - AccuTrack Runtime installed - Network connectivity (for network deployment) - Sufficient storage space - Required permissions

Upload from Device

Upload → Upload from Device: 1. Connect to device 2. Select project to upload 3. Download project package 4. Open in Designer

---

Device Management

The Device Manager helps discover and manage industrial devices.

Device Manager Tab

Access via **Project View → Devices** tab.

Device Discovery

Scanning for Devices

1. Click **Scan for Devices**
2. Configure scan parameters:
    - **Protocol**: Modbus, OPC UA, etc.
    - **IP Range**: Network range to scan
    - **Port**: Communication port
    - **Timeout**: Scan timeout
3. Start scan
4. Discovered devices appear in list

Modbus Scanner For Modbus devices: 1. Select **Modbus** protocol 2. Configure network settings 3. Scan discovers: - Device addresses - Available registers - Device information

Device Configuration

Adding Devices Manually

1. Click **Add Device**
2. Configure device:
    - **Name**: Device identifier
    - **Type**: Device type
    - **Protocol**: Communication protocol
    - **Address**: Network address
    - **Port**: Communication port
3. Test connection
4. Save device

Device Status

Monitor device status: - **Online**: Device is connected - **Offline**: Device is disconnected - **Error**: Communication error - **Unknown**: Status unknown

Device Information

View device details: - **Device ID**: Unique identifier - **Firmware Version**: Device firmware - **Capabilities**: Supported features - **Registers**: Available data points

---

Tips and Best Practices

Project Organization

1. **Naming Conventions**: Use consistent naming

    - Tags: `Tank1_Level`, `Pump1_Status`
    - Screens: `MainScreen`, `AlarmScreen`
    - Scripts: `StartupScript`, `AlarmHandler`
2. **Folder Structure**: Organize by function
    - Group related tags
    - Organize screens by area
    - Separate scripts by purpose
3. **Documentation**: Document complex configurations
    - Add descriptions to tags
    - Comment scripts
    - Document screen purposes

Tag Management

1. **Tag Naming**: Use descriptive names
    - Include location/equipment
    - Include measurement type
    - Use consistent abbreviations
2. **Tag Organization**: Group related tags
    - Use tag tables for logical groups
    - Organize by process area
    - Separate I/O from memory tags
3. **Tag Validation**: Validate early
    - Check addresses before deployment
    - Verify data types
    - Test tag connections

Screen Design

1. **Layout**: Design for clarity
    - Group related information
    - Use consistent positioning
    - Follow operator workflow
2. **Colors**: Use standard conventions
    - Red: Alarm, Stop
    - Green: Normal, Running
    - Yellow: Warning, Caution
    - Blue: Information
3. **Components**: Use appropriate components
    - Match component to data type
    - Use industrial components for equipment
    - Provide clear visual feedback
4. **Navigation**: Make navigation intuitive
    - Clear navigation paths
    - Breadcrumbs or menu bars
    - Quick access to critical screens

Scripting

1. **Performance**: Write efficient scripts
    - Minimize tag reads/writes
    - Use appropriate scan rates
    - Avoid blocking operations
2. **Error Handling**: Always handle errors
    - Use pcall for risky operations
    - Log errors appropriately
    - Provide fallback behavior
3. **Modularity**: Break into functions
    - Reusable functions
    - Clear function names
    - Single responsibility

Security

1. **Access Levels**: Set appropriate levels
    - Public for read-only displays
    - Operator for control actions
    - Engineer for configuration
2. **User Management**: Follow best practices
    - Strong passwords
    - Regular password changes
    - Disable unused accounts
3. **Audit Trail**: Enable logging
    - Track important actions
    - Monitor access attempts
    - Review logs regularly

Communication

1. **Connection Management**: Configure properly
    - Appropriate timeouts
    - Retry settings
    - Error handling
2. **Address Mapping**: Map addresses correctly
    - Verify device addresses
    - Test connections
    - Document mappings
3. **Performance**: Optimize communication
    - Group tag reads
    - Use appropriate scan rates
    - Monitor traffic

Testing

1. **Test Before Deployment**: Always test
   - Test tag connections
   - Test screen navigation
   - Test scripts
   - Test alarms
2. **Incremental Development**: Build incrementally
   - Add features one at a time
   - Test each addition
   - Verify before proceeding
3. **Backup**: Keep backups
   - Regular project backups
   - Version control (if possible)
   - Archive old versions

Troubleshooting

Common Issues

Project Won't Open **Symptoms**: Error opening project file

Solutions: 1. Verify file is valid `.isc` file 2. Check file permissions 3. Verify project structure is intact 4. Try opening backup version

Tags Not Updating **Symptoms**: Tag values not changing

Solutions: 1. Check communication connection status 2. Verify tag addresses are correct 3. Check device is online 4. Verify tag data types match 5. Check communication scan rate

Screen Components Not Displaying **Symptoms**: Components missing or not visible

Solutions: 1. Check component visibility property 2. Verify component is not behind another 3. Check Z-order/layering 4. Verify screen size is correct 5. Check component position is on screen

Script Errors **Symptoms**: Scripts not executing or errors

Solutions: 1. Check script syntax in editor 2. Verify tag names are correct 3. Check script execution context 4. Review console output for errors 5. Test script functions individually

Build Failures **Symptoms**: Build process fails

Solutions: 1. Check Output tab for specific errors 2. Verify all referenced files exist 3. Check tag references are valid 4. Verify script syntax 5. Clean project and rebuild

Communication Errors **Symptoms**: Cannot connect to devices

Solutions: 1. Verify network connectivity 2. Check IP addresses and ports 3. Verify device is powered and online 4. Check firewall settings 5. Verify protocol settings match device 6. Test with device manufacturer's tools

Alarm Not Triggering **Symptoms**: Alarms not activating when expected

Solutions: 1. Verify alarm condition is correct 2. Check tag value is updating 3. Verify threshold values 4. Check deadband settings 5. Verify alarm is enabled 6. Check alarm priority/severity

Getting Help

1. **Documentation**: Review this guide
2. **Console Output**: Check Output and Debug tabs
3. **Error Messages**: Read error messages carefully
4. **Logs**: Review application logs
5. **Support**: Contact support with:
   - Error messages
   - Steps to reproduce
   - Project configuration
   - System information

Debugging Tips

1. **Use Console**: Print debug information

   ```
   print("Debug: Tag value = " .. tagValue)
   ```

2. **Check Properties**: Verify component properties
3. **Test Incrementally**: Test one feature at a time
4. **Use Breakpoints**: (If available) Set breakpoints in scripts
5. **Monitor Tags**: Use tag monitor to verify values
6. **Check Connections**: Verify all connections are active

Appendix

Keyboard Shortcuts

Action	Shortcut
New Project	`Ctrl+N`
Open Project	`Ctrl+O`
Save Project	`Ctrl+S`
Build Project	`Ctrl+B`
Close Tab	`Ctrl+W`
Copy	`Ctrl+C`
Paste	`Ctrl+V`
Delete	`Delete`
Undo	`Ctrl+Z`
Redo	`Ctrl+Y`

File Extensions

- `.isc` - AccuTrack project file
- `.lua` - Lua script file
- `.json` - Configuration files
- `.qml` - QML screen files

Project File Locations

Projects are typically stored in: - **Windows**: `%USERPROFILE%\Documents\AccuTrack\Projects\`
- **Linux**: `~/Documents/AccuTrack/Projects/`

Support Resources

- **User Guide**: This document
- **Requirements Specification**: See SRS document
- **Module Documentation**: See ReadMe files
- **Code Documentation**: Inline code comments

End of User Guide

For technical specifications and architecture details, refer to the Requirements Specification (SRS) document.