Git Integration and Export Reference

Integrate solutions with Git repositories.

Reference **Solution** **DevOps** Track Changes | Git

Git Integration & Export (Reference): Comprehensive Git integration for version control, collaboration, and DevOps workflows while maintaining centralized database architecture security.

Access: Solution Export (Git Export settings)

Configuration
File Naming Convention
JSON File Format
JSON Import Behavior
TableType Reference
Object Reference Formats
Security & Sensitive Data
Git Workflow
Best Practices Checklist
Performance
Troubleshooting
See Also

Git Export Architecture

Hybrid Configuration Management

Component	Function	Location
Centralized Database	Single source of truth	.dbsln file
Automatic Exports	Version control tracking	Git folder
Selective Export	Configuration elements	JSON files
Sensitive Data	Protected via placeholders	Database only

Configuration

Enabling Git Export

Navigate to **Welcome Page**

Setting	Options	Default
Auto-save JSON files to Git folder	On/Off	On
Auto-save	Temporary / Closing Solution	When Closing Documents
Target Directory	<SolutionPath>\DesignerData\<SolutionName>-Designer\Git\	Read-only

Export Modes

Automatic Export (Auto-save):

- Monitors configuration changes
- Exports on version increment
- Batches rapid changes
- Background operation

Manual Export:

- Export All - Complete solution
- Export Module - Selected module
- Export Current - Active object

File Naming Convention

All JSON files are exported to a **single flat directory** using standardized naming:

Category	Pattern	Example
Bulk tables	`{TableType}.json`	`UnsTags.json`, `AlarmsItems.json`
Individual objects	`{TableType}-{Name}.json`	`ScriptsTasks-ServerStartup.json`
UserType members	`UserType-{Name}.json`	`UserType-LabelingMachine.json`

Examples:

`Git:`

- `AlarmsGroups.json`
- AlarmsGroups.json
- `AlarmsItems.json`
- `DevicesChannels.json`
- `DevicesNodes.json`
- `DevicesPoints.json`
- `ScriptsExpressions.json`
- `ScriptsTasks-ServerStartup.json`
- `ScriptsTasks-DataProcessor.json`
- `UnsTags.json`
- `UnsUserTypes.json`
- `UserType-LabelingMachine.json`
- `UserType-Motor.json`

Benefits:

- Simple enumeration: `ls *.json` returns all files
- Pattern matching: `ScriptsTasks-*.json` finds all script tasks
- Predictable paths for MCP/AI tools
- Alphabetical grouping by module prefix

JSON File Format

Standard Structure (Format v1.1)

json

```
{
  "ObjectIdentification": {
    "TableType": "ScriptsTasks",
    "SolutionName": "MyProject"
  },
  "ExportMetadata": {
    "ExportDate": "2025-08-23T10:30:00Z",
    "ExportUser": "john.doe",
    "ExportProduct": "FrameworX",
    "ExportProductVersion": "10.1.5.2034",
    "ExportFormat": "1.1"
  },
  "Data": [
    { /* object properties */ }
  ]
}
```

Section	Purpose	Import Behavior
`ObjectIdentification`	TableType and source solution	**Ignored** - TableType inferred from filename
`ExportMetadata`	Audit trail for export	**Ignored** - metadata only
`Data`	Array of configuration objects	**Processed** - actual data

Note: The `Data` section is always an array, even for single-object exports.

Fields Filtered from Export

The following fields are **excluded** from export to reduce file size and improve readability:

Auto-Generated Fields:

Field	Reason
`ID`	Auto-assigned unique identifier
`XRefInfo`	Cross-reference data rebuilt by system
`LockState`	Lock status reset on import
`LockOwner`	Lock owner reset on import

Compiled/Binary Data:

Field	Reason
`ContentsRun`	Compiled binary (.NET assembly)
`WebContentsRun`	Web/WASM compiled version
`BuildMessages`	Serialized build error data
`WebBuildMessages`	Web build errors

Build Results:

Field	Reason
`BuildStatus`	Build result, not configuration
`BuildErrors`	Build error count
`BuildMessage`	Build error text
`BuildRequired`	Build flag
`BuildNumber`	Build identifier

Calculated Fields:

Field	Reason
`Lines`	Calculated from Contents
`Length`	Calculated from Contents
`CanBePortable`	Calculated property
`Strings`	Extracted strings
`DriverVersion`	System-assigned

Note: Fields with empty or null values that represent configurable options (such as `Description`, `Trigger`, `NamespaceDeclarations`) are **kept** in the export for discoverability.

Export Categories

Individual Object Files (`{TableType}-{Name}.json`):

- Script Tasks and Classes
- Display Layouts
- Dataset Queries

- Report Definitions
- Symbol Definitions

Bulk Table Files (`{TableType}.json`):

- Tags (UnsTags)
- Alarm Items (AlarmsItems)
- Device Points (DevicesPoints)
- Expressions (ScriptsExpressions)
- UserType Definitions (UnsUserTypes)

JSON Import Behavior

The import function uses the **same JSON format** as export.

Import Behavior Rules

Scenario	Behavior
New object (Name doesn't exist)	**Created** with specified properties
Existing object (Name matches)	**Skipped** - no update, no error
Pre-defined system object	**Skipped** - cannot modify
Missing required field	**Error** - import fails for that object
Invalid reference	**Error** - referenced object must exist
Import file older than system object	**Warning** logged, object skipped

Conflict Detection

Import uses `DateModified` to detect potential conflicts:

- If system object is **newer** than import file Warn and skip
- If import file is **newer** than system object Safe to update (future feature)
- If dates are **equal** Skip (no changes)

Import Order Dependencies

When importing related objects, order matters:

1. **Independent objects first:** Tags, Channels, AlarmGroups, StorageLocations, DatasetDBs
2. **Dependent objects after:** NodesChannels, PointsNodes+Tags, AlarmItemsTags+Groups

---

TableType Reference

Valid `TableType` values:

TableType	Object	Module
`UnsTags`	Tag	UNS
`UnsUserTypes`	UserType	UNS
`UnsEnumerations`	Enumeration	UNS
`UnsTagProviders`	TagProvider	UNS
`UnsAssetTree`	AssetFolder	UNS
`DevicesChannels`	Channel	Devices
`DevicesNodes`	Node	Devices
`DevicesPoints`	Point	Devices
`AlarmsGroups`	AlarmGroup	Alarms

`AlarmsItems`	AlarmItem	Alarms
`AlarmsAreas`	AlarmArea	Alarms
`HistorianStorageLocations`	StorageLocation	Historian
`HistorianTables`	HistorianTable	Historian
`HistorianTags`	HistorianTag	Historian
`DatasetsDBs`	DatasetDB	Datasets
`DatasetsQueries`	DatasetQuery	Datasets
`DatasetsTables`	DatasetTable	Datasets
`ScriptsTasks`	ScriptTask	Scripts
`ScriptsClasses`	ScriptClass	Scripts
`ScriptsExpressions`	ScriptExpression	Scripts
`SecurityUsers`	SecurityUser	Security
`SecurityPermissions`	SecurityPermission	Security

Object Reference Formats

When JSON fields reference other objects, use these formats:

Reference To	Format	Example
Tag (root)	`Tag.{name}`	`"Tag.Temperature"`
Tag (nested)	`Tag.{path}.{name}`	`"Tag.Line1.Tank1.Level"`
Channel	`Device.Channel.{name}`	`"Device.Channel.ModbusPLC1"`
Node	`Device.Node.{name}`	`"Device.Node.PLC1"`
AccessType	`Device.AccessType.{name}`	`"Device.AccessType.Read"`
AlarmGroup	`Alarm.Group.{name}`	`"Alarm.Group.Critical"`
AlarmArea	`Alarm.Area.{name}`	`"Alarm.Area.Plant1"`
HistorianTable	`Historian.Table.{name}`	`"Historian.Table.Table1"`
DatasetDB	`DB.{name}`	`"Dataset.DB.TagHistorian"`

Security & Sensitive Data

Data Protection

Data Type	Handling	Storage
Passwords	Vault references	Secure storage
Connection Strings	Environment placeholders	`${DB_CONNECTION}`
API Keys	Vault references	Secure storage
Encrypted Values and Secrets	Not exported	Database only

Environment Variables

json

```
{
  "DatabaseConnection": "${DB_CONNECTION}",
  "EmailServer": "${SMTP_SERVER}",
  "HistorianPath": "${HISTORIAN_PATH}"
}
```

Variables resolved at runtime for environment-specific deployment.

Git Workflow

Recommended Process

Initialize Repository

```
cd /path/to/solution
git init
git remote add origin https://github.com/company/scada-config.git
```

Development Branch

```
git checkout -b feature/new-alarm-logic
# Make changes in FrameworX
git add .
git commit -m "Add temperature monitoring for Reactor-3"
git push origin feature/new-alarm-logic
```

Best Practices Checklist

What to Export

- **Export to Git:**
  - Application configuration
  - Display definitions
  - Script code
  - Alarm rules
  - Report templates
  - Static settings
- **Keep in Database Only:**
  - Runtime tag values
  - Historical data
  - User passwords
  - Session information
  - Temporary data
  - High-frequency updates

Performance

Export Performance

Configuration Size	Export Time	File Count	Total Size
Small (<100 objects)	<1 sec	20-50	<500 KB
Medium (1,000 objects)	2-4 sec	50-150	2-5 MB
Large (10,000 objects)	10-15 sec	200-500	20-50 MB
Enterprise (50,000+)	30-60 sec	1,000+	100-300 MB

Note: File counts and sizes reduced compared to previous format due to field filtering and flat structure.

CI/CD Integration

Example GitHub Actions

```
name: Validate FrameworX Configuration

on:
  pull_request:
    branches: [ main, develop ]

jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    - name: Validate JSON Schema
      run: |
        for file in $(find . -name "*.json"); do
          python validate_schema.py $file
        done

    - name: Check Tag Naming Convention
      run: python check_naming.py ./UnsTags.json

    - name: Verify No Credentials
      run: |
        ! grep -r "password\|credential\|secret" --include="*.json"
```

Note: CI/CD scripts simplified due to flat directory structure - no need to traverse nested folders.

Troubleshooting

Issue	Cause	Solution
Export not triggering	Automatic export disabled	Enable in settings
Merge conflicts	Concurrent edits	Use branches
Missing exports	Permission issues	Check directory access
Import skips object	Object already exists	Expected behavior (check logs)
Import error on reference	Referenced object missing	Import dependencies first
DateModified warning	System has newer version	Review changes before force import

See Also

- Object Model Quick Reference — Complete object reference
- JSON Import/Export Specification — Detailed field reference
- Track Changes Designer UI Reference — Internal change tracking

---

In this section...
