JSON Import Export Specification

Complete field reference for JSON-based configuration import and export.
Reference **Solution** **DevOps** Git Integration | JSON Specification

> JSON Import/Export Specification (Reference): Detailed field-level documentation for programmatic configuration of all object types via JSON files. Essential for MCP tools, CI/CD pipelines, and external integrations. **Format version 1.1.**

Overview

This specification documents the JSON format used for:

- Git export/import operations
- MCP (Model Context Protocol) tool integration
- CI/CD pipeline automation
- External system integration
- Bulk configuration management

Key Principle: Import and Export use the **same JSON format**. The only difference is that import ignores metadata sections.

JSON Structure

Standard Format (v1.1)

Every JSON file follows this three-section structure:

json

```
{
  "ObjectIdentification": {
    "TableType": "UnsTags",
    "SolutionName": "MyProject"
  },
  "ExportMetadata": {
    "ExportDate": "2026-01-16T10:30:00Z",
    "ExportUser": "john.doe",
    "ExportProduct": "FrameworX",
    "ExportProductVersion": "10.1.1.1011",
    "ExportFormat": "1.1"
  },
  "Data": [
    { /* object 1 */ },
    { /* object 2 */ }
  ]
}
```

Section Details

Section	Required for Import	Description
`ObjectIdentification`	No	TableType and source solution
`ExportMetadata`	No	Export audit information
`Data`	**Yes**	Array of configuration objects

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

Minimal Import Format

For import operations, only the `Data` array is required:

json

```
{
  "Data": [
    { "Name": "Tag1", "Type": "Double" }
  ]
}
```

Import Behavior

Fields Ignored on Import

These fields are auto-generated and ignored during import:

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

Fields Filtered from Export

These fields are excluded from export files to reduce size:

Compiled/Binary Data:

Field	Reason
`ContentsRun`	Compiled .NET assembly
`WebContentsRun`	Web/WASM compiled version
`BuildMessages`	Serialized build errors
`WebBuildMessages`	Web build errors

Build Results:

Field	Reason
`BuildStatus`	Build result
`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

Conditional:

Field	Rule
`Type`	Kept only for `UnsTags` and `UserType` tables

Import Rules

Scenario	Behavior
Name doesn't exist	Object created
Name already exists	**Skipped** (no error)
Pre-defined object	**Skipped** (cannot modify)
Missing required field	**Error**
Invalid reference	**Error**
Invalid enum value	**Error**

Pre-defined Objects (Cannot Import)

These system objects cannot be created or modified via import:

Module	Pre-defined Objects
Devices	All Protocols, AccessTypes (Read, Write, ReadWrite)
Alarms	Groups (Critical, Warning, AuditTrail)
Historian	StorageLocation (Default), HistorianTable (Table1)
Datasets	DBs (TagHistorian, AlarmHistorian, RuntimeUsers, Retentive)
Security	Users (Administrator, Guest), Permissions (24 built-in)

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, HistorianTagsTags+Tables

---

Import Modes

FrameworX 10.1.5 introduces three explicit modes that govern how an imported JSON file interacts with rows that already exist in the live solution. Earlier releases supported only a single behavior — an upsert that never deleted — which is preserved as the default **Merge** mode. The two additional modes give DevOps pipelines and source-of-truth workflows finer control over the live solution's drift relative to the JSON artifact.

The mode is selected **per table**, not for the import as a whole. A single import operation can run UnsTags in Mirror Sync while leaving DeviceChannels in Merge — useful when only part of the configuration is authoritatively tracked in Git.

Merge (default)

Merge is the historical and default behavior. Rows present in the JSON are **inserted** when their `Name` does not yet exist in the live solution, and **updated** when it does. Rows present in the live solution but **absent from the JSON are left in place**. Merge never deletes.

Merge is the right choice when the JSON file represents a partial contribution — for example, a new set of tags being added to an existing project, or a third-party module shipping a bundle of Channels and Nodes meant to coexist with the customer's own configuration.

Append

Append **inserts** rows whose `Name` does not yet exist and **skips** rows whose `Name` already exists. There is no update and no delete. Append guarantees the import will never modify an existing live row.

Append is the right choice when the JSON file is a one-shot seed — for example, importing a vendor-supplied tag library into a project that may have already customized some of those tags — and the operator wants the customer's edits preserved against any newer copy of the seed file.

Mirror Sync (10.1.5)

Mirror Sync makes the imported JSON the **authoritative state** of the selected table. Rows present in the JSON are inserted or updated exactly as in Merge mode. In addition, rows present in the live solution but **absent from the JSON are deleted**. After a successful Mirror Sync import, the live table mirrors the JSON exactly, row-for-row.

Mirror Sync targets three classes of workflow:

- **Authoritative Git-tracked configuration:** the JSON file in the repository is the single source of truth, and a row removed from the JSON must propagate to every environment on the next deployment.
- **CI/CD deployment pipelines:** a stale row left behind in the live solution from a prior build can corrupt the new deployment — for example, an obsolete Device Point that still polls a decommissioned address. Mirror Sync guarantees deployment-time hygiene.
- **Test environment resets:** a known-good JSON snapshot can reset a test solution to a clean baseline state without manual cleanup of leftover rows from prior test runs.

Opt-in mechanism. Mirror Sync is never the default. It is selected explicitly per table on two surfaces:

- The Designer import dialog presents a checkbox column labeled **Mirror sync** next to each table in the import. The operator ticks the box only for tables that should run in Mirror Sync; unticked tables run in Merge.
- The JSON import API and MCP-side tooling expose a parameter that accepts the list of `TableType` values to run in Mirror Sync. Tables not listed run in Merge.

MCP parameter (10.1.5). The ConsoleMCP `import_solution_json` tool and its DesignerMCP equivalent expose a `mirror_sync` parameter on the import call. The parameter accepts either a boolean — `true` to enable Mirror Sync for every table in the payload, `false` (default) to keep every table in Merge — or a per-table opt-in list of `TableType` values for finer control. Example invocations: `{"mirror_sync": true}` applies Mirror Sync to all imported tables; `{"mirror_sync": ["UnsTags", "UnsUserTypes"]}` runs Mirror Sync only for the two listed tables while every other table in the same import runs in Merge. Tool naming and signatures may evolve across MCP server versions — consult the MCP tool reference for the exact verb name and parameter shape in your installed FrameworX build.

Built-in row protection. Mirror Sync deletion never touches pre-defined system objects or rows marked as built-in. The `LockState` and `LockOwner` mechanism that already protects these rows from manual edit and from import overwrite also protects them from Mirror Sync deletion. A Mirror Sync import that omits, for example, the `Critical` alarm group or the `Default` storage location will **not** delete them — they are silently retained. This matches the behavior of the pre-defined objects table in the Import Behavior section above.

Recommended workflow. Mirror Sync is destructive. The recommended pre-flight before a production Mirror Sync import is:

1. **Export** the live solution's current state of the target tables to JSON.
2. **Diff** the exported JSON against the JSON about to be imported. Every row that appears in the export but not in the import is a row that Mirror Sync will delete.
3. **Review** the delete set with whoever owns the configuration. Deletions of operator-facing rows (Tags wired into Displays, Alarms wired into reports, Historian assignments) propagate downstream and may break dependent objects.
4. **Import** with Mirror Sync enabled for the reviewed tables.

Mode comparison. The following table summarizes the three modes against the three operations they may perform on each row in the live solution:

Mode	Insert new	Update existing	Delete missing
Merge (default)	Yes	Yes	No
Append	Yes	No	No
Mirror Sync	Yes	Yes	Yes (built-ins protected)

Tag Types Reference

Built-in Types

Type Name	String Value	Type ID	.NET Type	Range/Notes
Digital	`"Digital"`	151	Int32	0 or 1
Integer	`"Integer"`	152	Int32	±2,147,483,647

Double	`"Double"`	153	Double	±1.79×10³??
Text	`"Text"`	154	String	Unicode, 2GB max
DateTime	`"DateTime"`	155	DateTimeOffset	Year 0001-9999
Long	`"Long"`	156	Int64	±9.2×10¹?
Decimal	`"Decimal"`	157	Decimal	28-29 significant digits
TimeSpan	`"TimeSpan"`	158	TimeSpan	Days to milliseconds
Timer	`"Timer"`	162	Int32	Multiple timer modes
Counter	`"Counter"`	163	Int32	Up/down counting
DataTable	`"DataTable"`	164	DataTable	In-memory table
Image	`"Image"`	165	Byte[]	Binary content
Guid	`"Guid"`	166	Guid	128-bit unique ID
Json	`"Json"`	167	String	JSON with parsing
Reference	`"Reference"`	170	Dynamic	Tag reference

⚠ **Numeric precision in JSON export.** Integer, Long, and Digital tag values export at full precision. **Decimal** tag values — and Decimal UserType members — are rendered through a 64-bit floating-point (IEEE-754 double) representation on export, so a Decimal initial value (S`tartValue`) that requires more than ~15–17 significant digits is rounded to the nearest representable double in the exported JSON. The value stored in the solution database (`.dbsln`) and used at runtime is unaffected — only the JSON export rendering is limited. If you need full-precision Decimal values to survive an export/import cycle, treat the solution database as the source of truth rather than the exported JSON.

UserType References

To reference a UserType in JSON, use the UserType name:

json

```
"Type": "LabelingMachine"
```

Enumeration Values Reference

Domain

String Value	Description
`"Server"`	Server-side execution
`"Client"`	Client-side execution

Visibility

String Value	Description
`"Private"`	Not visible externally
`"Protected"`	Visible to solution
`"Public"`	Fully visible

Retentive

String Value	Description
`"None"`	Not retentive

`"ValueOnly"`	Value persisted
`"Properties"`	Value and properties persisted
`"PropertiesOnly"`	Only properties persisted

Script Code Language

String Value	Description
`"VB.NET"`	Visual Basic .NET
`"CSharp"`	C#
`"Python"`	Python

TableType Reference

Valid `TableType` values:

UNS Module

TableType	Object	Description
`UnsTags`	Tag	Process variables
`UnsUserTypes`	UserType	Data structure templates
`UnsEnumerations`	Enumeration	Value-to-text mappings
`UnsTagProviders`	TagProvider	External data connections
`UnsAssetTree`	AssetFolder	Folder organization

Devices Module

TableType	Object	Description
`DevicesProtocols`	Protocol	Built-in drivers (read-only)
`DevicesChannels`	Channel	Protocol instances
`DevicesNodes`	Node	Device addresses
`DevicesPoints`	Point	Tag-to-address mappings
`DevicesAccessTypes`	AccessType	Read/write behavior

Alarms Module

TableType	Object	Description
`AlarmsGroups`	AlarmGroup	Alarm behavior settings
`AlarmsItems`	AlarmItem	Alarm definitions
`AlarmsAreas`	AlarmArea	Hierarchy organization

Historian Module

TableType	Object	Description
`HistorianStorageLocations`	StorageLocation	Database connections
`HistorianTables`	HistorianTable	Storage rules
`HistorianTags`	HistorianTag	Tag assignments

Datasets Module

TableType	Object	Description
`DatasetsDBs`	DatasetDB	Database connections
`DatasetsQueries`	DatasetQuery	SQL statements
`DatasetsTables`	DatasetTable	Direct table mappings
`DatasetsFiles`	DatasetFile	File I/O definitions

Scripts Module

TableType	Object	Description
`ScriptsTasks`	ScriptTask	Triggered code blocks
`ScriptsClasses`	ScriptClass	Reusable libraries
`ScriptsExpressions`	ScriptExpression	One-line calculations
`ScriptsReferences`	ScriptReference	External DLL references

Reports Module

TableType	Object	Description
`ReportsForms`	ReportForm	Document templates
`ReportsWebData`	WebData	JSON/XML data objects

Security Module

TableType	Object	Description
`SecurityUsers`	SecurityUser	User accounts
`SecurityPermissions`	SecurityPermission	Permission flags
`SecurityPolicies`	SecurityPolicy	Login/session rules
`SecuritySecrets`	SecuritySecret	Protected credentials

Displays Module

TableType	Object	Description
`Displays`	Display	UI screens
`Layouts`	Layout	Frame organization

`Images`	Image	Imported graphics
`Symbols`	Symbol	Reusable components

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	`Channel.{name}`	`"Channel.ModbusPLC1"`
Node	`Node.{name}`	`"Node.PLC1"`
AccessType	`AccessType.{name}`	`"AccessType.Read"`
AlarmGroup	`Alarm.Group.{name}`	`"Alarm.Group.Critical"`
AlarmArea	`Alarm.Area.{name}`	`"Alarm.Area.Plant1"`
HistorianTable	`Historian.Table.{name}`	`"Historian.Table.Table1"`
StorageLocation	`Historian.StorageLocation.{name}`	`"Historian.StorageLocation.Default"`
DatasetDB	`DB.{name}`	`"DB.TagHistorian"`

UNS Objects

Tag (UnsTags)

json

```json
{
  "Data": [
    {
      "Name": "Temperature",
      "Type": "Double",
      "Path": "\\Line1\\Tank1",
      "Description": "Tank temperature sensor",
      "Array": 0,
      "StartValue": "25.0",
      "Min": "0",
      "Max": "100",
      "Units": "°C",
      "Format": "N1",
      "Domain": "Server",
      "Visibility": "Public",
      "Retentive": "None",
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Name`	Yes	string	Unique tag name
`Type`	Yes	string	Type name (see Tag Types Reference)
`Path`	No	string	Folder path (backslash-separated)
`Description`	No	string	Documentation (max 1024 chars)
`Array`	No	int	Array size (0=scalar, N=array[0..N])
`StartValue`	No	string	Initial value at startup

`Min`	No	string	Minimum allowed value
`Max`	No	string	Maximum allowed value
`Units`	No	string	Engineering units label
`Format`	No	string	Display format (N0, N2, etc.)
`Domain`	No	string	"Server" or "Client"
`Visibility`	No	string	"Private", "Protected", or "Public"
`Retentive`	No	string	"None", "ValueOnly", "Properties", "PropertiesOnly"
`DateCreated`	No	string	Creation timestamp (ISO 8601)
`DateModified`	No	string	Last modification timestamp (ISO 8601)

AssetFolder (UnsAssetTree)

json

```
{
  "Data": [
    {
      "Name": "Line1",
      "Path": "",
      "LinkedProvider": "",
      "Alias": "",
      "Visibility": "Public",
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    },
    {
      "Name": "Tank1",
      "Path": "\\Line1",
      "LinkedProvider": "",
      "Alias": "Primary Tank",
      "Visibility": "Public",
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Name`	Yes	string	Folder name
`Path`	No	string	Parent path (empty=root)
`LinkedProvider`	No	string	TagProvider link
`Alias`	No	string	Display name override
`Visibility`	No	string	"Private", "Protected", or "Public"

Devices Objects

Channel (DevicesChannels)

json

```
{
  "Data": [
    {
      "Name": "ModbusPLC1",
      "Protocol": "Modbus",
      "PrimaryStation": "192.168.1.10:502",
      "BackupStation": "",
      "Timeout": 3000,
      "Retries": 3,
      "Enabled": true,
      "Description": "Main Modbus PLC",
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Name`	Yes	string	Channel name
`Protocol`	Yes	string	Protocol name (Modbus, OPCUA, etc.)
`PrimaryStation`	Yes	string	Connection string
`BackupStation`	No	string	Redundant connection
`Timeout`	No	int	Timeout in milliseconds
`Retries`	No	int	Retry count
`Enabled`	No	bool	Active state
`Description`	No	string	Documentation

PrimaryStation Formats by Protocol:

Protocol	Format	Example
Modbus TCP	`IP:Port`	`192.168.1.10:502`
Modbus RTU	`COM:Baud:Parity:Data:Stop`	`COM1:9600:N:8:1`
OPC UA	`opc.tcp://host:port`	`opc.tcp://server:4840`
MQTT	`mqtt://host:port`	`mqtt://broker:1883`
Siemens S7	`IP:Rack:Slot`	`192.168.1.20:0:1`
AB ControlLogix	`IP:Slot`	`192.168.1.30:0`

Node (DevicesNodes)

json

```json
{
  "Data": [
    {
      "Name": "PLC1",
      "Channel": "Channel.ModbusPLC1",
      "PrimaryStation": "1",
      "BackupStation": "",
      "Enabled": true,
      "Description": "Main PLC unit 1",
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Name`	Yes	string	Node name
`Channel`	Yes	string	Parent channel reference
`PrimaryStation`	Yes	string	Device address
`BackupStation`	No	string	Redundant address
`Enabled`	No	bool	Active state
`Description`	No	string	Documentation

Point (DevicesPoints)

json

```json
{
  "Data": [
    {
      "Node": "Node.PLC1",
      "Tag": "Tag.Line1.Tank1.Temperature",
      "Address": "HR100",
      "AccessType": "AccessType.Read",
      "Enabled": true,
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Node`	Yes	string	Parent node reference
`Tag`	Yes	string	Tag reference
`Address`	Yes	string	Device address
`AccessType`	No	string	Default: Read
`Enabled`	No	bool	Active state

Address Formats by Protocol:

Protocol	Format	Examples
Modbus	`Type+Offset`	`HR100`, `IR50`, `C10`, `DI200`
OPC UA	`NodeId`	`ns=2;s=Tag1`, `ns=2;i=1001`
Siemens S7	`DB.Offset.Type`	`DB1.DBD0`, `DB1.DBX0.0`
AB ControlLogix	`TagName`	`Program:MainProgram.Tag1`

Alarms Objects

AlarmItem (AlarmsItems)

json

```json
{
  "Data": [
    {
      "Tag": "Tag.Line1.Tank1.Temperature",
      "Group": "Alarm.Group.Critical",
      "Area": "Alarm.Area.Plant1",
      "Condition": "GreaterThan",
      "Setpoint": "90",
      "Message": "Tank temperature high: {0}°C",
      "Priority": 800,
      "Enabled": true,
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Tag`	Yes	string	Monitored tag reference
`Group`	Yes	string	Alarm group reference
`Condition`	Yes	string	Trigger condition
`Setpoint`	Yes	string	Threshold value

`Message`	Yes	string	Alarm message ({0}=tag value)
`Area`	No	string	Alarm area reference
`Priority`	No	int	Priority override
`Enabled`	No	bool	Active state

Condition Values:

String Value	Description
`"GreaterThan"`	Value > Setpoint
`"LessThan"`	Value < Setpoint
`"GreaterOrEqual"`	Value >= Setpoint
`"LessOrEqual"`	Value <= Setpoint
`"Equal"`	Value == Setpoint
`"NotEqual"`	Value != Setpoint
`"HiHi"`	High-high limit
`"Hi"`	High limit
`"Lo"`	Low limit
`"LoLo"`	Low-low limit

Historian Objects

HistorianTag (HistorianTags)

json

```
{
  "Data": [
    {
      "Tag": "Tag.Line1.Tank1.Temperature",
      "HistorianTable": "Historian.Table.Table1",
      "Deadband": "0.5",
      "Enabled": true,
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Tag`	Yes	string	Tag to historize
`HistorianTable`	Yes	string	Storage table reference
`Deadband`	No	string	Logging deadband
`Enabled`	No	bool	Active state

Scripts Objects

ScriptTask (ScriptsTasks)

json

```
{
  "Data": [
    {
      "Name": "ServerStartup",
      "Code": "CSharp",
      "Domain": "Server",
      "Trigger": "",
      "InitialState": 0,
      "Description": "",
      "NamespaceDeclarations": "",
      "Contents": "// C# startup code\n@Info.Trace(\"Server started\");",
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Name`	Yes	string	Task name
`Code`	Yes	string	Language: "VB.NET", "CSharp", "Python"
`Domain`	No	string	"Server" or "Client"
`Trigger`	No	string	Trigger expression
`InitialState`	No	int	Initial state
`Description`	No	string	Documentation
`NamespaceDeclarations`	No	string	Additional namespaces
`Contents`	Yes	string	Script code

ScriptExpression (ScriptsExpressions)

json

```
{
  "Data": [
    {
      "Name": "TotalFlow",
      "Expression": "Tag.Flow1 + Tag.Flow2 + Tag.Flow3",
      "TargetTag": "Tag.TotalFlow",
      "TriggerTag": "Tag.Flow1",
      "Enabled": true,
      "DateCreated": "2026-01-16T10:30:00Z",
      "DateModified": "2026-01-16T10:30:00Z"
    }
  ]
}
```

Field	Required	Type	Description
`Name`	Yes	string	Expression name
`Expression`	Yes	string	Calculation formula
`TargetTag`	Yes	string	Result destination tag
`TriggerTag`	No	string	Tag that triggers recalculation
`Enabled`	No	bool	Active state

Common Patterns

Creating a Complete Device Communication Setup

Import order matters - create objects in dependency order:

Step 1: Create Tags

json

```
{ "Data": [{ "Name": "TankLevel", "Type": "Double", "Path": "\\Process" }] }
```

Step 2: Create Channel

json

```
{ "Data": [{ "Name": "ModbusCh1", "Protocol": "Modbus", "PrimaryStation": "192.168.1.10:502" }] }
```

Step 3: Create Node

json

```
{ "Data": [{ "Name": "PLC1", "Channel": "Channel.ModbusCh1", "PrimaryStation": "1" }] }
```

Step 4: Create Point

json

```
{ "Data": [{ "Name": "Point1", "Node": "Node.PLC1", "Tag": "Tag.Process.TankLevel", "Address": "HR100" }] }
```

Bulk Tag Creation

json

```
{
  "Data": [
    { "Name": "Motor1_Speed", "Type": "Double", "Path": "\\Motors", "Units": "RPM" },
    { "Name": "Motor1_Current", "Type": "Double", "Path": "\\Motors", "Units": "A" },
    { "Name": "Motor1_Running", "Type": "Digital", "Path": "\\Motors" },
    { "Name": "Motor2_Speed", "Type": "Double", "Path": "\\Motors", "Units": "RPM" },
    { "Name": "Motor2_Current", "Type": "Double", "Path": "\\Motors", "Units": "A" },
    { "Name": "Motor2_Running", "Type": "Digital", "Path": "\\Motors" }
  ]
}
```

Validation Rules

Name Validation

- Must be unique within object type
- Alphanumeric plus underscore
- Cannot contain dashes (reserved for expressions)
- Cannot start with number
- Max length: 256 characters
- Case-sensitive

Required Fields by Object

Object	Required Fields
Tag	Name, Type
Channel	Name, Protocol, PrimaryStation
Node	Name, Channel, PrimaryStation
Point	Node, Tag, Address
AlarmItem	Tag, Group, Condition, Setpoint, Message
HistorianTag	Tag, HistorianTable
DatasetQuery	Name, DatasetDB, Contents
ScriptTask	Name, Code, Contents

Summary of v1.1 Format Changes

Aspect	v1.0	v1.1

Sections	4 (ObjectIdentification, VersionInfo, ExportMetadata, Data)	3 (ObjectIdentification, ExportMetadata, Data)
ObjectIdentification	Module, TableType, TableGroup	TableType, SolutionName
VersionInfo	TableVersionID, SolutionVersionID, SolutionName	**Removed**
Folder structure	Hierarchical by module	Flat, single folder
File naming	Path-based	`{TableType}.json` or `{TableType}-{Name}.json`
Enum fields	Numeric IDs	String values
Data format	Array or single object	Always array
Type field	All objects	Only UnsTags and UserType tables
Filtered fields	None	ID, XRefInfo, LockState, ContentsRun, BuildMessages, etc.
DateCreated /DateModified	In Data objects	Kept in Data objects