MIFX — Machining Intent Format eXchange

Core Specification

MIFX Open Specification

Version 1.0 — February 26, 2026

Contents

1 Introduction

The Machining Intent Format eXchange (MIFX) defines a portable, system-neutral job package for the structured exchange of machining intent between software systems.

MIFX provides a structured representation of:

- machining setups,
- ordered operations,
- tool references and definitions,
- and optional machine-neutral motion data (e.g., APT/CL) for reference.

The format is designed to reduce ambiguity and information loss in machining job exchange by providing a structured, machine-readable representation of machining intent.

MIFX does not prescribe how machining is executed, validated, simulated, or governed. It does not define machining strategies, machine capabilities, or execution logic. It does not replace CAM systems, verification tools, machine controllers, or lifecycle management platforms. It provides a portable envelope for machining intent.

A MIFX package is distributed as a single archive file with the extension `.mifx`. The archive contains:

- a required intent definition file (`job.json`),
- setup definitions (`setup/`),
- operation definitions (`operations/`),
- optional motion data (`cldata/`),
- optional geometry (`geometry/`),
- and optional tool payload definitions (`geometry/tooling/`).

The file `job.json` is the sole required entry point of a valid MIFX package. It references all other components using relative paths within the archive.

All additional components are optional and may be ignored by conforming consumers. Conformance to MIFX is defined solely by adherence to the published JSON specification and structural rules defined in this document.

MIFX is an open specification intended for unrestricted implementation.

2 Philosophy and Scope Statement

MIFX — Machining Intent Format eXchange — is a minimal, open specification for the structured representation and exchange of machining intent in a portable, durable, and implementation-independent manner.

A MIFX package is a single archive file with extension `.mifx` containing a canonical JSON intent definition and optional referenced artifacts.

2.1 Purpose

The purpose of MIFX is to reduce ambiguity and information loss in machining job exchange across organizational and software boundaries.

MIFX provides a stable exchange layer without altering how CAM systems program or how machines execute.

It separates intent from execution while preserving explicit structural relationships.

2.2 Scope

MIFX defines:

- Structured machining intent (Setups, Tools, Operations)
- Explicit coordinate systems and transformations
- Deterministic operation ordering
- Artifact references via relative paths
- Optional association to machine-neutral APT/CL toolpaths
- Optional lightweight visualization geometry

MIFX does not define:

- Machine execution
- G-code dialects
- Simulation models
- Digital twins
- Controller behavior
- Feature recognition schemas
- CAM strategy definitions

Execution is not intent.

2.3 Design Principles

The specification is governed by the following principles:

1. **Minimal Core** — Only what is required for portable machining intent exchange.
2. **Sequential Determinism** — Operation order defines semantic processing order.
3. **Explicitness** — No hidden inference or implicit relationships.
4. **Path-Based Referencing** — All non-core artifacts are referenced by relative paths within the package.
5. **Graceful Degradation** — Optional components SHALL NOT invalidate a conforming job.
6. **Longevity** — Human-readable, text-based, archivally stable representation.
7. **Disk Truth** — The archive contents are authoritative; databases MAY index but SHALL NOT redefine intent.
8. **Open Implementation** — Unrestricted implementation of the core specification.

2.4 Strategic Position

MIFX does not replace CAM systems, post processors, simulation platforms, or digital manufacturing suites. It defines an interoperability layer for structured machining intent exchange.

It may be consumed directly by standalone viewers or indexed by external systems (e.g., SQL databases) without altering the canonical archive structure.

2.5 Long-Term Vision

A MIFX document created today shall remain interpretable decades later, independent of vendor survival or software availability.

If JSON parsing capability and ISO APT/CL documentation remain available, machining intent remains reconstructible.

MIFX is designed to be minimal in scope, stable in structure, and independent of proprietary runtime systems.

3 Design Principles

The MIFX specification is defined by strict technical constraints intended to preserve clarity, portability, determinism, and neutrality.

3.1 Single-Package Portability

A MIFX package SHALL be distributed as a single archive file with extension `.mifx`.

All information required to interpret the core machining intent SHALL reside within that archive.

Interpretation of the core intent SHALL NOT require external services, databases, network access, or platform-specific infrastructure.

3.2 Structural Intent Only

MIFX represents machining intent as structured data and defines no execution semantics.

It does not define execution behavior, machine control logic, simulation models, verification procedures, inspection processes, or governance mechanisms.

3.3 Separation of Structure and Motion

Intent structure SHALL be defined in `job.json` and associated structured JSON documents.

Machine-neutral motion data (e.g., APT/CL) SHALL be stored as separate artifacts within the package and referenced explicitly via relative paths.

Motion data SHALL NOT be embedded within structural JSON definitions.

3.4 Vendor and System Neutrality

The specification SHALL NOT assume or depend upon any specific CAM system, controller, simulation platform, enterprise system, or runtime environment.

MIFX defines a data structure, not an implementation model.

3.5 Minimal Surface Area

The core specification SHALL include only information necessary to describe setups, operations, tools, coordinate systems, and artifact references.

Lifecycle management, revision control, inspection modeling, execution tracking, enterprise orchestration, and digital twin representations are outside the scope of MIFX.

3.6 Path-Based Determinism

All non-core artifacts SHALL be referenced using relative paths within the archive.

No artifact SHALL require implicit discovery rules or environment-dependent resolution.

Document structure and explicit ordering SHALL define semantic processing order.

3.7 Archival Stability

Core records SHALL use stable, human-readable, text-based formats (JSON).

The structure SHALL remain interpretable without proprietary software or vendor-specific infrastructure.

A conforming MIFX archive SHALL remain structurally interpretable independent of database indexing or vendor survival.

4 Core Concepts

This section defines the primary entities used within a MIFX package.

4.1 Job

A Job represents a single machining definition contained within one MIFX package.

A MIFX archive SHALL contain exactly one Job.

The Job entry point SHALL be defined in `job.json`. The Job document references Setups, Operations, and optional Artifacts using relative paths within the archive.

4.2 Setup

A Setup defines a machining coordinate context under which Operations are performed.

A Setup may represent a workholding configuration, orientation, or coordinate system.

Each Setup SHALL be defined as a separate structured JSON document within the `setup/` directory.

Setups SHALL be processed in the order referenced by `job.json`.

4.3 Operation

An Operation represents a machining action within a Setup.

Operations SHALL be defined as individual structured JSON documents within the `operations/` directory.

Operations SHALL be processed in the order defined by the Setup referencing them.

An Operation MAY reference:

- a Tool definition,
- a Toolpath Artifact,
- optional in-process or visualization artifacts.

4.4 Tool

A Tool represents a machining instrument referenced by one or more Operations.

Tool structural attributes SHALL be defined in a Tool JSON document located in `geometry/tooling/`.

Operations reference Tools by identifier.

Optional visualization geometry (e.g., STL) MAY be associated with a Tool via explicit artifact reference.

4.5 Toolpath Artifact

A Toolpath Artifact is a machine-neutral motion file (e.g., APT, CL).

Toolpath Artifacts SHALL be stored separately from structural JSON definitions and referenced explicitly via relative paths.

Toolpath Artifacts are informational and SHALL NOT redefine structural intent.

4.6 Geometry Artifact

A Geometry Artifact is an optional file associated with the Job.
Geometry MAY represent:

- setup context,
- part representation,
- tool visualization.

Geometry Artifacts are informational and SHALL NOT alter core semantics.

4.7 Extension

An Extension is an optional data block that augments a MIFX document.
Extensions SHALL be namespaced or structurally isolated from core fields.
Conforming consumers SHALL ignore unknown Extensions.
Extensions SHALL NOT redefine, override, or alter core semantics.
Core validity SHALL NOT depend upon the presence of Extensions.

5 MIFX Core Data Model

This section defines the normative structure of a MIFX package.

MIFX uses a modular JSON document model. Each primary entity (Job, Setup, Operation, Tool) is defined in a separate JSON file. Relationships are expressed through identifiers and relative path references.

Semantic order SHALL be defined explicitly by array order within referencing documents. No execution graph, dependency model, or behavioral semantics are defined.

5.1 Package Entry Point

Every MIFX archive SHALL contain a file named `job.json` at the root of the archive.

The `job.json` file defines:

- global metadata,
- global unit definition,
- ordered references to Setup documents,
- optional global artifact references.

Exactly one `job.json` SHALL exist per archive.

5.2 Job Structure

The Job document SHALL contain:

- `exportVersion`
- `header`
- `units`
- `setups` (ordered array of setup references)

The `setups` array order SHALL define semantic Setup order.

5.3 Setup Definition

Each Setup SHALL be defined as a separate JSON document within the `setup/` directory.

A Setup definition SHALL contain:

- `id` (unique within the Job)
- optional transformation matrix
- ordered array of Operation references

Operation order SHALL be defined by array position.

If no transformation is defined, identity SHALL be assumed.

5.4 Operation Definition

Each Operation SHALL be defined as a separate JSON document within the `operations/` directory.

An Operation SHALL contain:

- `id` (unique within the Job)
- `setupRef`
- optional `toolRef`
- optional `cuttingConditions`
- optional `artifactRefs`

Operations SHALL NOT imply execution behavior.

5.5 Tool Definition

Tools SHALL be defined as JSON documents within `geometry/tooling/`.

A Tool definition SHALL contain:

- `toolNumber`
- `toolType`
- optional dimensional attributes
- optional extension metadata

Tool identifiers SHALL be referenced by Operations via `toolRef`.

Optional visualization geometry (e.g., STL) MAY be referenced explicitly via artifact references.

5.6 Artifact References

Artifacts SHALL be referenced using explicit relative paths.

An artifact reference SHALL contain:

- `role`
- `kind`
- `path`
- optional `present`

Example:

```
{
  "role": "toolpath",
  "kind": "apt",
  "path": "cldata/op-1.apt",
  "present": true
}
```

Artifact resolution SHALL NOT rely on directory inference rules. The `path` value SHALL be authoritative.

If `present` is `true`, the referenced file SHALL exist within the archive. If `present` is omitted or `false`, the file MAY be absent.

Artifacts SHALL NOT redefine structural semantics.

5.7 Toolpath Artifact

For toolpath artifacts:

- `role` SHALL be `"toolpath"`
- `kind` SHALL be `"apt"` or `"cl"`
- the referenced file SHALL reside within `cldata/`

Toolpath artifacts are informational and non-normative with respect to execution.

5.8 CuttingConditions

An Operation MAY define cutting conditions:

```
{
  "cuttingConditions": {
    "spindleDir": "CW",
    "spindle": { "value": 12000, "unit": "rpm" },
    "feed": { "value": 350, "unit": "mm/min" },
    "coolant": "through_tool"
  }
}
```

These values represent declared intent only and SHALL NOT define controller behavior.

No inheritance model is defined.

5.9 Determinism

A conforming MIFX consumer SHALL be able to:

- parse `job.json`,
- reconstruct Setup order from the `setups` array,
- reconstruct Operation order from Setup references,

- resolve Tool references,
- resolve artifact paths deterministically.

A conforming consumer SHALL produce identical structural interpretation given identical archive contents.

Consumers SHALL NOT infer implicit relationships beyond those explicitly defined.

MIFX defines structure and relationships only. It does not define execution behavior, simulation models, controller syntax, or lifecycle management.

6 MIFX Job Package

A MIFX Job Package is a single archive file with extension:

`.mifx`

The archive SHALL be a ZIP-compatible container.

All directory names defined by this specification SHALL be treated as case-sensitive.

A conforming MIFX archive SHALL contain exactly one Job and SHALL include the following structure:

```
job.json                        (REQUIRED)

setup/                          (REQUIRED)
    setup-1.json
    setup-2.json
    ...

operations/                     (REQUIRED)
    op-1.json
    op-2.json
    ...

geometry/                       (OPTIONAL)
    setup/                      (OPTIONAL)
        setup-1.3mf
        setup-x.3mf
    tooling/                    (OPTIONAL)
        main-6.json
        main-6.stl

cldata/                         (OPTIONAL)
    op-1.apt
    op-2.apt
```

6.1 job.json

The file `job.json` SHALL be located at the root of the archive.

It defines the normative machining intent entry point and SHALL reference:

- global metadata,
- unit definition,
- ordered Setup documents.

Only one `job.json` file SHALL exist per archive.

6.2 setup/

The directory `setup/` SHALL contain one JSON file per Setup.

Each Setup document SHALL define:

- a unique `id`,
- optional transformation matrix,
- ordered references to Operation documents.

If a Setup is referenced in `job.json`, its JSON file SHALL exist.

6.3 operations/

The directory `operations/` SHALL contain one JSON file per Operation.

Each Operation SHALL define:

- a unique `id`,
- `setupRef`,
- optional `toolRef`,
- optional cutting conditions,
- optional artifact references.

If an Operation is referenced by a Setup, its JSON file SHALL exist.

6.4 geometry/

The directory `geometry/` MAY contain optional geometry artifacts.

Subdirectories MAY include:

- `setup/` for part or setup geometry (e.g., 3MF, STEP)
- `tooling/` for Tool JSON definitions and optional STL visualization

Geometry artifacts are informational and SHALL NOT affect structural semantics.

6.5 cldata/

The directory `cldata/` MAY contain machine-neutral motion files referenced by Operations.

If an Operation declares an artifact reference with:

```
"role": "toolpath"
```

the referenced file SHALL exist at the specified relative path.

Toolpaths SHALL conform to ISO 4343 APT/CL or a documented equivalent neutral motion format.

If no toolpath artifacts are referenced, the `cldata/` directory MAY be absent.

6.6 Path Authority

All artifact references SHALL use explicit relative paths.

Directory inference rules SHALL NOT be used.

The `path` value within each artifact reference SHALL be authoritative.

6.7 Extensions

MIFX Core does not define a dedicated `extensions/` directory.

Implementation-specific files MAY be included within the archive provided that:

- they do not alter core semantics,
- they are not required for structural interpretation,
- core validity does not depend upon them.

Conforming consumers SHALL ignore unknown files without error.

6.8 Manifest

No manifest mechanism is defined by the MIFX Core specification.

The authoritative structure of a MIFX archive is determined solely by:

- the presence of `job.json`,
- valid Setup and Operation references,
- and explicit artifact path resolution.

7 Conformance Levels

MIFX implementations MAY conform at different levels of capability.

Conformance levels describe the degree to which an implementation can interpret and process the MIFX core data model. They do not define execution behavior.

7.1 Level 1 — Structural Conformance

A Level 1 implementation SHALL:

- Load `job.json`
- Resolve referenced Setup documents in `setup/`
- Resolve referenced Operation documents in `operations/`
- Preserve declared order (document order in arrays and explicit references)
- Resolve internal references deterministically (e.g., `setupRef`, `toolRef`)

Toolpath and geometry artifacts are optional at this level.

7.2 Level 2 — Artifact-Aware Conformance

A Level 2 implementation SHALL meet all Level 1 requirements and:

- Resolve artifact references via explicit relative `path`
- Load toolpath artifacts (e.g., APT/CL) when present
- Validate artifact presence when declared `present=true`

No simulation, verification, or execution behavior is required.

7.3 Level 3 — Advanced Interpretation

A Level 3 implementation MAY:

- Interpret Setup transforms for downstream processing
- Interpret declared cutting conditions as machining intent metadata
- Perform structural validation beyond basic parsing (e.g., schema validation, reference integrity)

Advanced interpretation behavior is implementation-defined and outside the normative scope of the MIFX core specification.

MIFX does not define controller syntax, post-processing behavior, machine scheduling, or lifecycle orchestration.

8 Determinism and Long-Term Readability

MIFX SHALL support deterministic interpretation and long-term archival stability.

8.1 Sequential Semantics

The semantic order of machining intent is defined exclusively by explicit array order within JSON documents.

- Setups SHALL be interpreted in the order listed in the `setups` array of `job.json`.
- Operations SHALL be interpreted in the order referenced by each Setup.
- No dependency graph, execution tree, or parallel structure is defined.

Reordering elements defined as sequential SHALL be considered a semantic modification.

8.2 Explicit Referencing

All relationships SHALL be explicit.

- Operations reference Setups via `setupRef`.
- Operations reference Tools via `toolRef`.
- External artifacts are referenced via explicit relative `path` fields.

Consumers SHALL NOT infer implicit relationships beyond those explicitly defined.

8.3 Stable Mathematical Representation

Coordinate transformations SHALL be represented explicitly as 4x4 matrices.

- Matrix storage order SHALL be declared.
- Units SHALL be declared.
- Coordinate convention SHALL be declared.

Transformation matrices SHALL be interpreted exactly as declared without inferred coordinate system assumptions.

No hidden orientation logic is permitted.

8.4 Unit Declaration

Linear units SHALL be declared in `job.json`.

All numeric values SHALL respect declared units.

Numeric values SHALL NOT be interpreted under implicit default unit systems.

8.5 Graceful Degradation

If optional components are missing:

- Missing geometry SHALL NOT invalidate the Job.
- Missing tool visualization geometry SHALL result in default primitive rendering.
- Missing toolpath files SHALL NOT invalidate structural parsing unless explicitly declared with `"present": true`.

MIFX readers SHALL degrade gracefully.

8.6 Human Readability

Core MIFX JSON documents SHALL remain:

- Plain UTF-8 encoded text.
- Fully viewable and inspectable in a standard text editor.
- Free from binary encoding.

Core documents SHALL remain readable without proprietary tooling.

8.7 Archival Longevity

MIFX is designed to remain interpretable independent of specific vendor software implementations.

As long as JSON parsing capability and ISO APT/CL specifications remain documented, machining intent SHALL remain reconstructible.

8.8 Forward Compatibility

Extensions SHALL be represented as optional JSON fields that do not conflict with core-defined properties.

Unknown fields SHALL be ignored without failure.

Unknown artifact roles SHALL be ignored unless explicitly required by the core specification.

Core validity SHALL NOT depend upon extension-defined fields.

Future core revisions SHALL preserve backward compatibility within the same major version.

9 Scope and Explicit Exclusions

This section defines the functional boundaries of the MIFX specification. The boundaries defined in this section are normative and binding.

9.1 In Scope

MIFX defines a portable, structured representation of machining intent, including:

- Job identity and metadata,
- ordered Setup definitions,
- ordered Operation definitions,
- Tool identity, structural attributes, and references,
- explicit coordinate transformations,
- explicit relative-path references to machine-neutral motion data (e.g., APT/CL),
- optional informational geometry associated with Setups or Tools,
- optional extension fields that do not alter core semantics.

The specification defines structure, ordering, and explicit relationships only. It does not define execution behavior.

9.2 Explicit Exclusions

The following are explicitly outside the scope of MIFX:

- lifecycle management systems,
- revision control mechanisms,
- approval workflows,
- execution tracking or shop-floor feedback,
- inspection feature modeling,
- tolerance definition or GD&T semantics,
- model-based definition (MBD) surface or face tagging,
- simulation or material removal modeling,
- collision detection,
- axis configuration or machine kinematic models,
- controller-specific NC dialect definition,
- post-processing logic,
- manufacturing execution systems (MES) scheduling,
- enterprise resource planning (ERP) integration,
- digital twin representations,
- database indexing models or hash storage strategies.

MIFX does not enforce manufacturing processes, validate machine execution, or replace CAM systems, controllers, simulation platforms, or enterprise orchestration systems.

MIFX defines a portable, implementation-neutral data contract for structured machining intent exchange.

10 Governance Model

MIFX is an open specification intended for unrestricted implementation and exchange.

10.1 Open Specification

The MIFX core specification SHALL be:

- Publicly available
- Freely implementable
- Royalty-free
- Not controlled by a commercial vendor

The specification SHALL be publicly versioned and archived.
No license SHALL restrict reading, writing, distributing, or implementing the format.

10.2 Core Stability Principle

The MIFX Core Data Model SHALL remain stable within a published major version.
Breaking changes SHALL NOT occur within the same major version.
New capabilities SHALL be introduced only through:

- Backward-compatible additions
- Optional fields
- Extension mechanisms

Core fields SHALL NOT be removed or redefined within a major version.

10.3 Minimal Core Philosophy

The MIFX core SHALL remain intentionally minimal.
The following SHALL NOT become part of the core:

- Machine kinematic definitions
- Controller dialect specifications
- Simulation engines
- Feature recognition schemas
- Digital twin models
- Lifecycle or enterprise orchestration models

Additions to the core SHALL require demonstrable improvement to interoperability and portability.
If an addition increases complexity without increasing portability, it SHALL be rejected.

10.4 Extension Mechanism

Vendors and researchers MAY extend the format using additional JSON fields that do not conflict with core-defined properties.

Extensions:

- SHALL NOT modify or override core semantics.
- SHALL NOT redefine required core fields.
- SHALL NOT be required for structural validity.

Consumers that do not recognize an extension field SHALL ignore it without failure.
Core validity SHALL NOT depend upon extension-defined properties.

10.5 Reference Implementations

The specification MAY be accompanied by:

- A reference Viewer
- A reference Exporter

Such implementations are illustrative only and SHALL NOT define or alter the standard.

10.6 No Central Authority Requirement

MIFX does not require:

- A certification body
- Vendor approval
- Formal consortium membership

Adoption is voluntary and implementation-driven.

10.7 Versioning

The version of a MIFX document SHALL be declared in `job.json` using the field:

```
"exportVersion": "x.y.z"
```

Versioning SHALL follow the model:

- Major version (`x`) — May introduce structural changes. Backward readability of prior major versions is not guaranteed.
- Minor version (`y`) — SHALL introduce only backward-compatible additions.
- Patch version (`z`) — SHALL introduce clarifications or corrections without structural change.

Documents conforming to a given major version SHALL remain readable by implementations supporting that major version.

11 Extension Policy

MIFX is designed with a minimal normative core and an unrestricted extension mechanism.

Extensions enable customization without altering, redefining, or fragmenting the core data model.

11.1 Principle

The MIFX core specification is normative and stable.

Extensions are:

- OPTIONAL,
- non-normative,
- implementation-defined.

A conforming MIFX document SHALL remain valid and interpretable even if all extensions are ignored.

Core validity SHALL NOT depend on the presence, interpretation, or processing of extensions.

11.2 Goals of the Extension Mechanism

The extension system exists to:

- allow vendor-specific metadata,
- support controller-specific requirements,
- enable enterprise customization,
- accommodate future domain expansion,
- prevent core specification inflation.

Extensions SHALL NOT modify, override, reinterpret, or supersede the normative semantics of the MIFX core model.

11.3 Extension Mechanism

Extensions MAY be introduced as additional JSON fields within:

- `job.json`,
- Setup documents,
- Operation documents,
- Tool definitions.

Extension fields SHALL NOT conflict with core-defined property names.

Extension properties SHOULD use vendor-qualified naming conventions (e.g., `"vendorName.property"` or nested vendor objects) to reduce collision risk.

Consumers that do not recognize an extension field SHALL ignore it without error.

Unknown fields SHALL NOT invalidate structural parsing.

11.4 Inline Extensions

Inline extensions MAY appear as additional JSON properties alongside core-defined properties.

Example:

```
{
  "id": "op-1",
  "setupRef": "set-1",
  "vendorX.customParameter": "value"
}
```

Inline extensions are informational and non-normative.

They SHALL NOT override, redefine, contradict, or reinterpret core-defined properties.

11.5 External Extension Resources

A MIFX package MAY include additional files that are not referenced by the core model.

Such files:

- are OPTIONAL,
- MAY follow vendor-defined schemas,
- are outside the scope of this specification.

The presence or absence of such files SHALL NOT affect the validity of the MIFX core structure.

11.6 Non-Interference Clause

Extensions SHALL NOT:

- alter declared Setup or Operation ordering,
- redefine core property meaning,
- introduce hidden structural dependencies,
- require mandatory external services,
- introduce required execution semantics.

Ignoring extensions SHALL NOT cause structural ambiguity or invalidate core interpretation.

11.7 Stability Guarantee

The existence of extensions SHALL NOT force revision of the MIFX core specification.

Core evolution, if any, SHALL remain independent of vendor-specific or domain-specific extension mechanisms.