

Tronsoft Interactive Fiction Engine

Lean Developer Specification - Version 3.0.1

Status: Baseline implementation specification

Audience: engine developers, game authors, tool developers, portal integrators, and testers

This edition intentionally replaces the proposed multi-thousand-page documentation set. It defines only the contracts needed to build, test, package, install, and publish TIFE games.

1. Purpose and Scope

TIFE is a data-driven interactive narrative engine built around actions, world state, rules, events, services, and configurable user-interface panels. The specification defines stable package formats and runtime behavior while allowing games, plugins, and themes to extend the platform without modifying the core engine.

1.1 Design goals

- Human-readable project files
- A stable engine identifier: tife
- Separate engine and game packages
- Keyboard-first and accessibility-first operation
- Classic parser commands plus intuitive command buttons
- Portable browser runtime
- Deterministic validation and packaging
- Portal-ready metadata and update support

1.2 Non-goals

- A general-purpose 3D engine
- Mandatory online services
- A proprietary binary authoring format
- Requiring authors to write JavaScript for ordinary game content
- Embedding portal-specific business logic inside the runtime

1.3 Normative language

MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY are used as requirement terms. A conforming implementation satisfies every MUST requirement.

2. System Architecture

The runtime is organized as cooperating services connected through an event bus. Games submit actions. Rules examine the action and current state. The world service applies valid changes. Presentation services update text, panels, audio, narration, and accessibility announcements.

Player Input -> Action Service -> Rule Service -> World Service -> Event Bus -> UI / Audio / Save / Director

2.1 Required core services

Service	Responsibility
ActionService	Creates and processes normalized action objects.
ParserService	Converts free-form commands into candidate actions.
RuleService	Validates preconditions and applies rule outcomes.
WorldService	Owns rooms, objects, exits, NPCs, and world variables.
StateService	Owns persistent game state and state transitions.
EventBus	Publishes typed events to decoupled subscribers.
SaveService	Creates, restores, lists, and deletes saves.
UIService	Renders transcript, commands, panels, dialogs, and notifications.

AccessibilityService	Manages narration, captions, focus, scaling, and profiles.
PackageService	Loads and validates installed game packages.

2.2 Optional services

- AudioService
- AchievementService
- QuestService
- HintService
- GameDirector
- AnalyticsService
- CloudSaveAdapter
- LocalizationService

3. Runtime Lifecycle

1. Load engine configuration.
2. Register core services.
3. Load installed theme and plugins.
4. Validate the selected game manifest.
5. Load game indexes and initial world data.
6. Restore a selected save or create initial state.
7. Render the cover screen.
8. Obtain user activation for audio and narration.
9. Enter the action-processing loop.

3.1 Failure behavior

A package validation failure **MUST** stop game startup and present a readable error report. The runtime **MUST NOT** silently ignore missing required files. Noncritical optional assets **MAY** produce warnings and fall back to text-only behavior.

4. Action Model

Every player or system operation is represented as an action object. Parser commands and clickable buttons produce the same action structure.

```
{
  "type": "take",
  "actor": "player",
  "target": "brass_lantern",
  "indirect": null,
  "source": "command_button",
  "raw": "take lantern",
  "timestamp": 0
}
```

4.1 Required fields

Field	Type	Requirement
-------	------	-------------

type	string	Required normalized verb.
actor	string	Required actor ID.
target	string null	Primary object or entity.
indirect	string null	Secondary object or entity.
source	string	parser, command_button, system, dialogue, or test.
raw	string	Original user input when available.

4.2 Action processing result

```
{
  "accepted": true,
  "events": ["inventory.item_added"],
  "messages": ["Taken."],
  "state_changed": true,
  "turn_consumed": true
}
```

5. World Model

5.1 Room

```
{
  "id": "forest_clearing",
  "title": "Forest Clearing",
  "description": "A narrow path disappears into the trees.",
  "exits": {"north": "cave_entrance"},
  "objects": ["brass_lantern"],
  "npcs": [],
  "tags": ["outdoors"],
  "variants": []
}
```

Room IDs MUST be unique within the game. Exits MUST reference valid room IDs unless marked as dynamic. The runtime SHOULD announce available exits according to the active accessibility profile.

5.2 Object

```
{
  "id": "brass_lantern",
  "name": "brass lantern",
  "description": "An old battery-powered lantern.",
  "portable": true,
  "visible": true,
  "states": {"lit": false},
  "actions": ["take", "drop", "examine", "switch_on", "switch_off"]
}
```

5.3 NPC

```
{
  "id": "caretaker",
  "name": "the caretaker",
  "home_room": "visitor_center",
}
```

```

    "dialogue": "caretaker_intro",
    "schedule": [],
    "inventory": [],
    "factions": ["park_staff"]
  }

```

5.4 Variables and flags

Variables MAY be boolean, integer, number, string, or string arrays. Game data SHOULD declare variable types and defaults in story/variables.json. Rules MUST NOT create undeclared persistent variables in release builds.

6. Rules and Events

Rules provide data-driven conditions and effects. Rules are evaluated in priority order. A rule MAY reject an action, modify it, add messages, emit events, or change state.

```

{
  "id": "lantern_required_for_depths",
  "when": {"action.type": "go", "action.target": "down"},
  "if": [{"not": {"inventory.contains": "brass_lantern"}}],
  "then": [{"reject": "It is too dark to descend safely."}],
  "priority": 100
}

```

6.1 Standard event namespaces

- engine.*
- game.*
- action.*
- world.*
- room.*
- inventory.*
- npc.*
- dialogue.*
- quest.*
- achievement.*
- audio.*
- save.*
- ui.*
- accessibility.*

6.2 Event requirements

Events MUST contain a type and payload. Persistent state changes MUST be complete before the corresponding event is published. Event handlers SHOULD be idempotent when practical.

7. Parser and Command Buttons

The parser is an input adapter, not the center of the engine. A game MAY run entirely through buttons, entirely through text, or through both.

7.1 Parser behavior

- Normalize case and punctuation.
- Resolve common verb synonyms.
- Resolve visible objects before global objects.
- Ask a clarification question when multiple targets are equally valid.
- Never expose internal IDs to players.
- Return suggestions when a command is not understood.

7.2 Contextual command buttons

The UI **MUST** be able to request a finite set of valid or likely actions for the current context. Buttons **SHOULD** use plain-language labels such as Examine Lantern, Take Lantern, Go North, Talk to Caretaker, and Open Journal.

```
{
  "label": "Take Lantern",
  "action": {"type": "take", "target": "brass_lantern"},
  "group": "Objects",
  "priority": 80,
  "enabled": true
}
```

8. User Interface and Panels

The default presentation uses a terminal-inspired transcript plus optional panels. Themes control appearance; layouts control placement; panels provide tools without owning game state.

8.1 Required views

- Cover screen
- Transcript
- Command entry
- Contextual command buttons
- Modal message/dialogue area
- Save/load interface
- Settings and accessibility

8.2 Standard optional panels

- Journal
- Quest Log
- Achievements
- Hint System
- Character Stats
- Conversation History
- Transcript Viewer
- Save Manager
- Inventory
- Map

8.3 Responsive behavior

Desktop layouts MAY display several panels simultaneously. Tablet and mobile layouts SHOULD collapse panels into tabs or drawers. The transcript and command controls MUST remain reachable without horizontal scrolling.

9. Accessibility

Accessibility support is a core requirement, not an optional plugin.

9.1 Profiles

Profile	Defaults
Standard	Normal layout and motion.
Low Vision	Large text, strong contrast, larger controls.
Blind / Screen Reader	Linearized interface, focus announcements, narration.
Hard of Hearing	Captions for important audio and visible alerts.
Motor Assistance	Large targets, reduced timing pressure, keyboard-first.
Cognitive Support	Simplified layout, explicit objectives, reduced clutter.

9.2 Mandatory behavior

- All actions operable by keyboard.
- Visible focus indicator.
- Meaningful labels for controls.
- Captions for gameplay-critical sounds.
- No puzzle may require color alone.
- Text scaling must not hide controls.
- Reduced-motion setting must suppress decorative animation.
- Narration settings persist between sessions.

10. Audio and Narration

Audio is optional; equivalent information must remain available through text or captions when audio conveys gameplay-relevant information.

10.1 Categories

- music
- ambient
- effects
- interface
- narration
- voice

10.2 Browser activation

The cover screen SHOULD provide explicit Enter, Enable Audio, and Enable Narration controls. Audio playback MUST begin only after a valid user activation when required by the host browser.

11. Game Director

The optional Game Director coordinates story milestones, dynamic descriptions, hint escalation, NPC schedules, pacing, autosaves, music transitions, and replay decisions. It **MUST** use public runtime services and events rather than bypassing the rule or state systems.

11.1 Milestone example

```
{
  "id": "act_2_discovery",
  "condition": {"quest.completed": "find_entrance"},
  "effects": [
    {"set": {"story.act": 2}},
    {"unlock_room": "crystal_gallery"},
    {"emit": "story.milestone_reached"}
  ]
}
```

12. Saves and State

A save contains game identity, game version, engine compatibility, timestamp, player state, world state, quest state, and plugin state. Saves **MUST NOT** contain executable code.

```
{
  "format": "tife-save",
  "format_version": "1.0",
  "engine_id": "tife",
  "game_id": "com.tronsoft.colossal-cave",
  "game_version": "3.0.0",
  "created_at": "2026-07-26T12:00:00Z",
  "state": {}
}
```

12.1 Migration

Games **SHOULD** provide migrations when a release changes persistent state. Migration functions **MUST** be deterministic and **MUST** preserve the previous save until the migrated save is written successfully.

13. Project Structure

```
MyAdventure/
  project.json
  manifest.json
  story/
    rooms/
    objects/
    npcs/
    dialogue/
    quests/
    rules/
    variables.json
  assets/
    images/
    audio/
```


localization/
themes/
plugins/
tests/
docs/
build/

Source projects are editable folders. Release packages are immutable ZIP-compatible archives with a .tife-game extension.

14. Package Formats

14.1 Package types

Extension	Purpose	Install location
.tife-engine	Runtime engine	engines/tife/<version>
.tife-game	Playable game	games/<game_id>/<version>
.tife-plugin	Runtime or Studio extension	plugins/<plugin_id>/<version>
.tife-theme	Theme and layout assets	themes/<theme_id>/<version>

14.2 Stable engine identity

Every TIFE engine package MUST declare "engine_id": "tife". Games MUST require engine_id tife. Engine versions change; engine_id does not.

14.3 Game manifest

```
{
  "manifest_version": "1.0",
  "package_type": "tife-game",
  "game_id": "com.tronsoft.sample",
  "title": "Sample Adventure",
  "version": "1.0.0",
  "requires": {"engine_id": "tife", "engine_version": ">=3.0.0"},
  "entry": "story/index.json",
  "language": "en-US",
  "authors": ["Tronsoft Games"]
}
```

14.4 Required package files

- manifest.json
- story/index.json
- README.md
- LICENSE.txt or an explicit license field
- cover artwork or a declared text-only presentation
- checksums.json for portal releases

15. Validation

Validation produces errors and warnings. Errors block release builds. Warnings are reported but MAY be permitted by policy.

15.1 Required checks

- Valid JSON
- Manifest conforms to schema
- Semantic version is valid
- engine_id equals tife
- Entry file exists
- All referenced IDs resolve
- No duplicate IDs
- No path traversal
- No executable files in game packages unless explicitly allowed by policy
- Required metadata exists
- Gameplay-critical audio has captions
- Cover and icon meet portal requirements
- Checksums match

15.2 World analysis

- Unreachable rooms
- Unintended one-way exits
- Objects located in multiple exclusive containers
- Dead dialogue branches
- Quest dependency cycles
- Rules that can never fire
- Unused assets
- Missing localization keys

16. Asset and Content Pipeline

The content pipeline imports, indexes, validates, optimizes, and packages assets. Source files remain authoritative; generated indexes and caches belong in build/ and MUST NOT be required for source control.

16.1 Supported baseline formats

Category	Formats
Images	PNG, JPEG, SVG, WebP
Audio	OGG, MP3, WAV
Fonts	WOFF2; TTF/OTF for authoring only
Documents	Markdown, HTML, PDF
Data	UTF-8 JSON

16.2 Build stages

10. Scan source tree.
11. Validate package and content schemas.
12. Build dependency graph.
13. Copy or convert assets.
14. Generate runtime indexes.
15. Remove excluded development files.
16. Generate checksums.

17. Sign package when configured.
18. Write build report.

17. Localization

Display text **SHOULD** use localization keys rather than being duplicated in logic. The base language is required. Missing translations **MUST** fall back to the base language and generate a build warning.

```
{  
  "room.forest.title": "Forest Clearing",  
  "room.forest.description": "A narrow path disappears into the trees."  
}
```

18. Plugins and Themes

18.1 Plugin rules

- Plugins declare an ID, version, API range, permissions, and entry point.
- Plugins **MUST** use public APIs.
- Game packages **MUST NOT** silently install plugins.
- Portal installations **SHOULD** require user or administrator approval for new plugin permissions.
- Plugin failures **SHOULD** be isolated when possible.

18.2 Theme rules

- Themes control visual tokens, typography, panel layout, and optional sound cues.
- Themes **MUST** preserve keyboard navigation and readable focus.
- Themes **MUST** declare light/dark/high-contrast support.
- A game **MUST** remain usable with the engine fallback theme.

19. Studio Minimum Viable Product

The first TIFE Studio release is complete when an author can create a project, draw a room graph, edit rooms and exits, add objects and NPCs, define dialogue and quests, playtest, validate, and export a .tife-game package.

19.1 Editors

- Project Wizard
- World Builder
- Room Editor
- Object Editor
- NPC Editor
- Dialogue Graph
- Quest Editor
- Rule Editor
- Asset Browser
- Validation Report
- Integrated Playtest

19.2 Developer playtest controls

- Teleport
- Grant/remove inventory
- Set variables
- Trigger events
- Start/complete quests
- Inspect rule evaluation
- View event timeline
- Create test save

20. Runtime API Summary

Interface	Key operations
TifeRuntime	start, stop, loadGame, dispatchAction, getState
EventBus	on, once, off, emit
WorldService	getRoom, moveActor, getObject, updateObjectState
RuleService	evaluate, registerRule, explain
SaveService	create, load, list, delete, migrate
UIService	write, notify, showPanel, setCommandOptions
AccessibilityService	setProfile, announce, caption, focus
PackageService	validate, install, uninstall, resolveDependencies

Detailed method signatures are provided in the api/ Markdown files included in this package.

21. Testing Requirements

21.1 Automated tests

- Manifest schema tests
- World reference tests
- Action/rule unit tests
- Save/load round-trip tests
- Migration tests
- Keyboard navigation tests
- Accessibility profile smoke tests
- Package checksum tests
- Known walkthrough integration tests

21.2 Release gates

19. No validation errors.
20. All required automated tests pass.
21. A clean install launches successfully.
22. A new game can be started and saved.
23. The save can be restored after restart.
24. Keyboard-only operation is verified.
25. Critical audio information is captioned.
26. Package version is newer than the installed version unless force-repair is explicitly selected.

22. Security

- Reject absolute paths and ../ traversal in archives.
- Limit extracted file count and uncompressed size.
- Verify checksums before activation.
- Treat game data as untrusted input.
- Disallow inline script execution in ordinary game packages.
- Escape text before inserting it into HTML.
- Require explicit permission declarations for plugins.
- Keep staging and rollback copies during updates.

23. Portal Integration

The Tronsoft Games Portal stores TIFE games using `engine_id = tife`. The library query for TIFE titles filters on that value. Installation registers package identity, version, publish status, artwork paths, and compatibility data.

23.1 Publication metadata

- Title
- Short and long description
- Version
- Authors and publisher
- Cover, icon, banner, and screenshots
- License
- Changelog
- Patch notes
- Accessibility summary
- Supported languages
- Engine requirement
- Release date
- News article metadata

23.2 Update behavior

An update **MUST** have a newer semantic version unless an administrator deliberately chooses a repair or force-install operation. Updates **SHOULD** stage files, validate the staged result, atomically activate it, and retain a rollback point.

24. Coding and Data Conventions

- IDs: lowercase ASCII snake_case or reverse-domain identifiers for global package IDs.
- Files: lowercase kebab-case except conventional names such as README.md.
- JSON: UTF-8, two-space indentation, no comments.
- Versions: semantic versioning major.minor.patch.
- Times: ISO 8601 UTC for stored timestamps.
- Paths: forward slashes and package-relative references.
- User-facing text: no internal IDs or stack traces.

25. Conformance Checklist

- ☐ engine_id is exactly tife
- ☐ Game manifest validates
- ☐ Required files exist
- ☐ All references resolve
- ☐ No duplicate IDs
- ☐ Runtime supports actions from parser and buttons
- ☐ Keyboard operation works
- ☐ Accessibility settings persist
- ☐ Save/load round trip works
- ☐ Package installation is staged and reversible
- ☐ Portal metadata is complete
- ☐ Release report contains zero errors

Appendix A. Minimal Game

```
manifest.json
story/index.json
story/rooms/forest-clearing.json
story/objects/brass-lantern.json
README.md
LICENSE.txt
```

A working example matching this structure is included under `examples/minimal-adventure/`.

Appendix B. Standard Commands

Category	Examples
Movement	go north, north, enter cave, climb ladder
Observation	look, examine lantern, inventory, exits
Objects	take, drop, open, close, switch on, use
Conversation	talk to, ask about, tell about, choose response
Meta	save, load, restart, transcript, settings, help, hint

Appendix C. Versioning Policy

Major versions may change public contracts. Minor versions add backward-compatible capabilities. Patch versions fix defects without changing public contracts. Package schemas declare their own versions independently of engine versions.

Appendix D. Build Report Example

Build successful
Game: Minimal Adventure 1.0.0
Engine: tife >=3.0.0
Rooms: 2
Objects: 1
NPCs: 0
Quests: 1
Errors: 0
Warnings: 0
Package checksum: <sha256>