PXYZ System Reference: Comprehensive Guide

Executive Summary

PXYZ is a revolutionary **workflow compiler and runtime system** designed to safely grant AI agents access to production systems. Unlike traditional imperative programming, PXYZ treats business logic as **auditable data** rather than executable code.

Core Value Proposition:

- **Provable Safety**: Workflows are finite, deterministic, and bounded
- **Full Auditability**: Every operation is explicit and reviewable
- **Compile-Time Safety**: Dangerous patterns are caught before deployment
- **Human-in-the-Loop**: Irreversible actions require human confirmation
- **Production-Ready**: ~600 lines of verifiable WebAssembly runtime

Part 1: System Architecture

Three-Component Design

1. Rust Compiler (~1,500 LOC)

Purpose: Transform XML workflow definitions into portable binary artifacts

Responsibilities:

- Parse XML into Abstract Syntax Tree (AST)
- Lower AST to Graph Intermediate Representation (IR)
- Compile predicates to Predicate VM bytecode
- Apply three-layer validation (Syntactic, Semantic, Pragmatic)
- Optimize for binary size and execution efficiency
- Emit portable `graph.bin` binary

Output: `graph.bin` - A self-contained, portable workflow binary

2. WASM Runtime (~600 LOC)

Purpose: Execute workflows in a completely sandboxed environment

Architecture:

- Written in WebAssembly Text (.wat) for maximum auditability

- Embeds Predicate VM for safe condition evaluation
- Enforces strict execution limits (1000 visited nodes, 256 predicate steps max)
- Zero direct system access (fully sandboxed)
- All side effects mediated through host imports

Safety Limits:

- `MAX_VISITED_NODES = 1000` (prevent runaway traversal)
- `MAX_PREDICATE_STEPS = 256` (prevent infinite loops)
- `MAX_STACK_DEPTH = 16` (prevent stack overflow)
- `MAX_CALL_DEPTH = 4` (prevent recursion)
- `MAX_PREDICATE_BYTECODE = 256 bytes` (limit complexity)

3. JavaScript Host

Purpose: Provide the bridge between sandboxed runtime and external systems

Responsibilities:

- Implement host imports (`io_call`, `io_resolve_var`, etc.)
- Resolve variables from application context
- Execute external I/O operations
- Log and audit all operations
- Return responses to runtime

Host Imports:

- `io_call(op_code)` - Execute external operation
- `io_resolve_var(path)` - Resolve $variable.paths
- `io_is_human()` - Check if actor is human
- `io_is_confirmed(entity)` - Check confirmation status
- `io_log(message)` - Emit log messages
- `emit_event(event)` - Emit audit events

Part 2: Workflow Definition Language (XML)

Document Structure

```
&lt;omar&gt;
  &lt;schemas&gt;&lt;/schemas&gt;
  &lt;predicates&gt;&lt;/predicates&gt;
  &lt;workflow&gt;&lt;/workflow&gt;
  &lt;templates&gt;&lt;/templates&gt;
&lt;/omar&gt;
```

Core Elements

Nodes

Definition: Units of work or control points in the workflow graph

Kind	Purpose	Characteristics
Transform	Validate/transform data	No side effects, local only
External	Call I/O handler	Side effects, irreversible
Render	Generate output	Template-based, user-facing
Signal	UI framework signal	Client-side changes
Auth	Authorization check	Permission validation
Terminal	Success endpoint	Return 2xx status
Error	Error endpoint	Return error status

Edges

Definition: Directed connections between nodes

Properties:

- `from` (required): Source node ID
- `to` (required): Target node ID
- `weight` (optional): Priority (higher = evaluated first)
- `parallel` (optional): Can be traversed in parallel
- `fallback` (optional): Only taken if other paths fail

Conditional Traversal:

```
&lt;edge from="A" to="B"&gt;
  &lt;when&gt;
    &lt;and&gt;
      &lt;eq left="$token.role" right="admin"/&gt;
      &lt;contains left="$state.users.ids" right="$token.sub"/&gt;
    &lt;/and&gt;
  &lt;/when&gt;
&lt;/edge&gt;
```

Predicates

Definition: Reusable boolean conditions

Available Operations:

Category	Operations
Comparison	`eq`, `neq`, `gt`, `gte`, `lt`, `lte`
String	`contains`, `matches`, `startsWith`, `endsWith`
Logic	`and`, `or`, `not`
Type	`fn name="is_defined"`, `fn name="is_null"`

Variable Paths:

- `$token.sub` - User ID
- `$token.perms` - Permissions array
- `$token.tenant` - Tenant ID
- `$entity.owner_id` - Entity owner
- `$entity.status` - Entity status
- `$input.query` - User input
- `$state.node_id.field` - Output from node

Part 3: Compilation Pipeline

Six-Stage Transformation

Stage 1: Parsing

- **Input**: XML text
- **Output**: Abstract Syntax Tree (AST)
- **Function**: XML parsing and basic structure validation

Stage 2: Lowering

- **Input**: AST
- **Output**: Graph Intermediate Representation (IR)
- **Function**: Convert high-level AST to graph-centric IR, resolve symbolic names to numeric IDs

Stage 3: Predicate Compilation

- **Input**: Predicate expressions
- **Output**: Predicate VM bytecode
- **Function**: Compile all predicates to efficient bytecode

Stage 4: Validation

- **Input**: Graph IR
- **Output**: Validation diagnostics
- **Function**: Three-layer constraint checking

Stage 5: Optimization

- **Input**: Validated IR
- **Output**: Optimized IR
- **Function**: Dead code elimination, predicate deduplication

Stage 6: Emission

- **Input**: Optimized IR
- **Output**: `graph.bin` binary
- **Function**: Serialize to binary format with header and data sections

Three-Layer Validation System

Syntactic (SYN) - Structure Validation

Ensures graph structure is well-formed:

- SYN001: Edge targets exist
- SYN002: Entry points reference existing nodes
- SYN003: Predicate references exist
- SYN004: No duplicate node IDs
- SYN005: At least one entry point defined
- SYN006: No duplicate entry points
- SYN007: Edge sources exist

Semantic (SEM) - Logic Validation

Ensures graph logic is coherent:

- SEM001: Auth nodes have predicates
- SEM002: External nodes have op codes
- SEM003: Terminal nodes shouldn't have outgoing edges
- SEM004: No cycles (must be DAG)
- SEM005: All nodes reachable from entry
- SEM006: Error nodes have incoming edges
- SEM007: Render nodes have templates

Pragmatic (PRAG) - Business & Safety Rules

Enforces high-level safety policies:

- **PRAG001**: LLM → Irreversible action paths require validation gate
- **PRAG002**: Write operations should have error-handling branches
- **PRAG003**: Irreversible actions require human-in-the-loop
- **PRAG004**: Irreversible actions require confirmed inputs
- **PRAG005**: Quarantined data cannot escape to I/O operations

Part 4: Binary Format (graph.bin)

File Structure

Header (96 bytes)

```
Offset | Size | Field | Description
-------|------|-------|------------
0x00   | 4    | Magic | 0x504E5958 (ASCII: "PXYZ")
0x04   | 2    | Ver.Major | Binary format version
0x06   | 2    | Ver.Minor |
0x08   | 4    | Node count | Total nodes in graph
0x0C   | 4    | Edge count | Total edges in graph
0x10   | 4    | Pred count | Total predicates
0x14   | 4    | String pool size | String pool bytes
0x18   | 4    | Entry count | Number of entry points
0x20   | 32   | Source hash | SHA-256 of source XML
0x40   | 4    | Nodes offset | Byte offset to nodes section
0x44   | 4    | Edges offset | Byte offset to edges section
0x48   | 4    | Predicates offset | Byte offset to predicates
0x4C   | 4    | Strings offset | Byte offset to string pool
0x50   | 4    | Entries offset | Byte offset to entry points
```

Node Entry (16 bytes)

```
Offset | Size | Field | Description
-------|------|-------|------------
0x00   | 4    | Node ID | Index in node array
0x04   | 1    | Kind | 0-6 (Transform, External, etc.)
0x05   | 1    | Flags | Bitfield of properties
0x06   | 2    | Op code | I/O operation code
0x08   | 4    | Data offset | String pool offset
0x0C   | 2    | Edge start | Index in edges array
0x0E   | 2    | Edge count | Number of outgoing edges
```

Edge Entry (12 bytes)

```
Offset | Size | Field | Description
-------|------|-------|------------
0x00   | 4    | Target node ID | Destination node
0x04   | 2    | Predicate ID | Condition ID (0=unconditional)
0x06   | 2    | Reserved | Future use
0x08   | 2    | Weight | Priority (higher first)
0x0A   | 2    | Flags | Edge properties
```

Entry Point Entry (8 bytes)

```
Offset | Size | Field | Description
-------|------|-------|------------
0x00   | 4    | PX hash | FNV-1a hash of (P, X)
0x04   | 4    | Node ID | Starting node for this entry
```

Data Sections

- **Nodes**: Contiguous array of Node Entries (16 bytes each)
- **Edges**: Sorted by source node ID
- **Predicates**: Variable-length bytecode chunks
- **Strings**: Null-terminated UTF-8 strings (deduplicated pool)
- **Entry Points**: Lookup table for (P, X) → Node ID

Part 5: Predicate VM

Bytecode Instruction Set

The Predicate VM is a stack-based interpreter with 26 opcodes:

Stack Operations

- `0x01 PUSH_INT` - Push 32-bit integer
- `0x02 PUSH_STR` - Push string reference
- `0x03 LOAD_VAR` - Load variable from host
- `0x04 LOAD_FIELD` - Get field from object

Comparisons

- `0x10 EQ` - Equal (pop a, b → push a==b)
- `0x11 NEQ` - Not equal
- `0x12 GT` - Greater than
- `0x13 GTE` - Greater or equal
- `0x14 LT` - Less than
- `0x15 LTE` - Less or equal

Logical Operations

- `0x20 AND` - Logical AND
- `0x21 OR` - Logical OR
- `0x22 NOT` - Logical NOT

String Operations

- `0x30 CONTAINS` - String contains
- `0x31 MATCHES` - Regex match
- `0x32 STARTS_WITH` - String prefix
- `0x33 ENDS_WITH` - String suffix

Type Operations

- `0x40 LEN` - String/array length
- `0x41 GET` - Array element access
- `0x42 IS_NULL` - Check null
- `0x43 IS_DEFINED` - Check defined
- `0x44 IS_CONFIRMED` - Check confirmation status

Advanced

- `0xF0 CALL_PRED` - Call another predicate (by 16-bit ID)
- `0xFF RET` - Return boolean value

Part 6: IO Operation Codes

Operation Code Structure

Operation codes are 16-bit values: `0xXXYY`

- `XX` = Category
- `YY` = Specific operation

Categories

Entity Operations (0x01xx)

- `0x0100` ENTITY_CREATE
- `0x0101` ENTITY_READ
- `0x0102` ENTITY_UPDATE
- `0x0103` ENTITY_DELETE ⚠ IRREVERSIBLE
- `0x0104` ENTITY_LIST
- `0x0105` ENTITY_SEARCH

Google Workspace (0x03xx)

- `0x0300` GOOGLE_CONTACTS_SEARCH
- `0x0301` GOOGLE_CONTACTS_GET
- `0x0302` GOOGLE_CONTACTS_CREATE
- `0x0310` GOOGLE_CALENDAR_LIST
- `0x0320` GOOGLE_DRIVE_SEARCH
- `0x0330` GOOGLE_GMAIL_SEARCH
- `0x0332` GOOGLE_GMAIL_SEND ⚠ IRREVERSIBLE

Communication (0x034x-0x036x)

- `0x0340` EMAIL_SEND ⚠ IRREVERSIBLE
- `0x0350` SMS_SEND ⚠ IRREVERSIBLE
- `0x0360` WEBHOOK_CALL ⚠ IRREVERSIBLE

HTTP (0x04xx)

- `0x0400` HTTP_GET
- `0x0401` HTTP_POST
- `0x0402` HTTP_PUT
- `0x0403` HTTP_DELETE

Vector/RAG (0x07xx)

- `0x0700` QDRANT_SEARCH
- `0x0701` QDRANT_INDEX
- `0x0702` EMBEDDING_GENERATE

AI/LLM (0x08xx)

- `0x0800` LLM_COMPLETE
- `0x0801` LLM_CLASSIFY
- `0x0802` LLM_STRUCTURED
- `0x0810` LOCAL_MODEL_RUN

Storage (0x09xx)

- `0x0900` STORAGE_GET
- `0x0901` STORAGE_SET
- `0x0910` EVENT_LOG_APPEND
- `0x0911` EVENT_LOG_QUERY

Part 7: Command-Line Interface

pxyz compile

Compile XML workflow to binary:

```
pxyz compile --input workflow.xml --output graph.bin --audit --strict
```

Options:

- `--input FILE` - Input workflow.xml
- `--output FILE` - Output graph.bin
- `--audit` - Generate audit.json with metadata
- `--strict` - Treat warnings as errors

pxyz inspect

Inspect compiled binary:

```
pxyz inspect --input graph.bin --format mermaid
```

Formats:

- `text` - Human-readable summary
- `json` - Detailed metadata
- `mermaid` - Visual flowchart

pxyz check

Validate workflow without compiling:

```
pxyz check workflow.xml --strict
```

pxyz init

Create new project:

```
pxyz init --name my-workflow
```

Part 8: Design Philosophy

Why These Constraints?

Business Logic as Data

Benefit: Complete static analysis before execution

- Graph is fully traversable without execution
- All possible paths identifiable at compile time
- Behavior is deterministic and repeatable

Bounded Execution

Benefit: Guaranteed termination

- No unbounded loops or recursion
- Execution time is predictable
- Resource exhaustion is impossible

Explicit I/O

Benefit: Full auditability

- Every external interaction is declared
- Side effects are traceable
- Accidental operations are caught

Compile-Time Safety

Benefit: Problems caught early

- Dangerous patterns prevented at compilation
- No runtime surprises in production
- Constraints enforced by architecture

Human Gates

Benefit: Critical actions are controlled

- Irreversible operations require approval
- AI cannot independently perform high-risk actions
- Humans remain in control

Auditable Runtime

Benefit: Complete transparency

- ~600 lines of WebAssembly can be reviewed in hours
- Formal methods verification possible
- No hidden behaviors

Part 9: When to Use PXYZ

Ideal Use Cases ✅

- AI agents executing workflows
- Multi-step approval processes
- Data validation pipelines
- Third-party integrations
- Systems requiring audit trails
- Workflows with irreversible actions
- Any system where safety is critical

Not Ideal ✖

- Real-time high-frequency operations
- Complex algorithms (use external services)
- Stateful long-running processes
- Systems with extreme performance requirements

Part 10: Getting Started

1. Define Your Workflow

Create `workflow.xml` with nodes and edges

2. Compile

```
pxyz compile --input workflow.xml --output graph.bin --audit
```

3. Inspect

```
pxyz inspect --input graph.bin --format mermaid
```

4. Review

Check `audit.json` for validation results

5. Test

Load graph.bin in runtime with test data

6. Deploy

Embed graph.bin in production with host implementation

Appendix: Quick Reference

Node Kinds (0-6)

0=Transform, 1=External, 2=Render, 3=Signal, 4=Auth, 5=Terminal, 6=Error

Node Flags

- Bit 0: ASYNC
- Bit 1: REQUIRES_AUTH
- Bit 2: HAS_SIDE_EFFECTS
- Bit 3: IRREVERSIBLE
- Bit 4: REQUIRES_HUMAN
- Bit 5: CACHEABLE

Edge Flags

- Bit 0: PARALLEL
- Bit 1: FALLBACK
- Bit 2: ERROR_EDGE

Safety Limits

- MAX_VISITED_NODES: 1000
- MAX_PREDICATE_STEPS: 256
- MAX_STACK_DEPTH: 16
- MAX_CALL_DEPTH: 4
- MAX_PREDICATE_BYTECODE: 256 bytes

Magic Number

`0x504E5958` (ASCII: "PXYZ")

PXYZ: Safely Grant AI Agents Access to Production Systems

⁂