# The Syntax of Cognition: A Comprehensive Analysis of the Prompt Orchestration Markup Language (POML) and the Standardization of Artificial Intent

## 1. Introduction: The Crisis of Unstructured Dialogue

The trajectory of Artificial Intelligence in the first half of the 2020s was defined by a singular, paradoxical struggle: the attempt to control stochastic, probabilistic systems using the deterministic, rigid tools of traditional software engineering. As Large Language Models (LLMs) transitioned from research curiosities to the backbone of enterprise infrastructure, the primary interface for human-machine interaction—the "prompt"—remained stubbornly primitive. For years, the state of the art in prompt engineering consisted of "string hacking": the concatenation of raw text, interpolated variables, and few-shot examples into monolithic string objects. This practice, while flexible, was inherently fragile, unscalable, and opaque. It lacked the structural rigor necessary for complex, multi-turn agentic workflows, leading to a phenomenon widely recognized in the industry as "spaghetti prompting," where narrative logic, data context, and system instructions became inextricably amused in a single, unmanageable text block.

Against this backdrop, the release of the Prompt Orchestration Markup Language (POML) in August 2025 marked a pivotal moment in the history of AI engineering.1 Developed by a team at Microsoft Research, POML was not merely a new tool but a proposal for a new ontological framework. It posits that a prompt is not a string of text, but a structured document—a hierarchical arrangement of semantic intentions, data references, and presentational rules.2 By adopting an XML-inspired syntax, POML sought to bring the standardization that HTML brought to the web to the chaotic domain of Generative AI.

This report provides an exhaustive, archival analysis of POML. It traces the language's genesis within the broader context of Human-Computer Interaction (HCI) research, dissects its technical specifications and architectural decisions, and evaluates its reception across three distinct communities: the pragmatic world of AI engineering, the theoretical domain of the Digital Humanities, and the experimental sphere of creative coding. Through a close reading of primary source code, technical white papers, and community discourse, we argue that POML represents a "structural turn" in AI interaction—a movement away from the alchemy of natural language persuasion toward the engineering of semantic documents.

## 2. Historical Genesis and Theoretical Foundations

### 2.1 The Pre-POML Landscape (2022–2024)

To understand the necessity of POML, one must reconstruct the operational environment of the early GenAI era. Between 2022 and 2024, prompt engineering was characterized by a lack of standardization. Developers relied on ad-hoc patterns, often storing prompts as JSON strings or plain text files interspersed with Python f-strings or JavaScript template literals. This approach suffered from critical deficiencies:

* **Format Sensitivity:** LLMs demonstrated extreme volatility regarding whitespace and formatting. A prompt that functioned correctly in a Python script might fail if copied into a web interface due to invisible character differences. This "brittle" nature of prompts made version control and regression testing nearly impossible.2
* **Data Integration Bottlenecks:** Inserting multimodal data—such as a CSV table or a reference to an image—required complex pre-processing. Developers had to manually serialize data into strings, often running into token limits or formatting errors that confused the model.3
* **The "Context" Problem:** As context windows expanded to 128k and 1M tokens, the "flat" structure of a text string became inadequate for organizing massive amounts of information. There was no semantic way to tell the model that *this* block of text was a high-priority instruction while *that* block was low-priority background data.4

By early 2025, the industry had begun to fragment. Frameworks like LangChain and LlamaIndex attempted to abstract this complexity through code (classes and objects), but these solutions were often criticized for being "heavy and unwieldy," introducing layers of abstraction that obscured the actual prompt being sent to the model.5 The community demanded a solution that was "native" to the medium of text yet structured enough for engineering.

### 2.2 The Microsoft Research Initiative (August 2025)

The formal introduction of POML occurred on August 13, 2025, with the publication of the white paper *Prompt Orchestration Markup Language* on arXiv (arXiv:2508.13948) by Yuge Zhang, Nan Chen, Jiahang Xu, and Yuqing Yang.2 The composition of this research team is significant. Nan Chen, for instance, brought a deep background in data visualization and narrative systems, having previously published on "Content-Format Integrated Prompt Optimization".6 This lineage suggests that POML was conceived not just as a backend utility, but as a Human-Computer Interaction (HCI) layer designed to facilitate "human-agent collaboration".8

The release strategy was dual-pronged: a theoretical academic paper validating the methodology, and an immediate open-source release of the reference implementation on GitHub (microsoft/poml).9 The project was explicitly positioned as a "standard," drawing direct analogies to the role of HTML in web development. Just as the <div> and <p> tags standardized the visual web, tags like <role> and <task> were intended to standardize the semantic web of agents.4

### 2.3 The "Nightly" Evolution and Community Engagement

The archival record of the POML repository reveals a rapid, iterative development cycle characteristic of modern open-source projects. Snippets from the npm registry show a flurry of "nightly" builds (e.g., 0.0.9-nightly.20251206) following the initial release, indicating active, daily refinement of the core parser.10

Crucially, the development was not hermetically sealed within Microsoft. The GitHub repository shows engagement with the broader "Responsible AI" standard, implying that POML was designed from the ground up to support enterprise governance requirements, such as safety auditing and bias mitigation.9 The inclusion of a Contributor License Agreement (CLA) bot and strict code of conduct points to a desire to foster a regulated, corporate-friendly open-source ecosystem, distinguishing it from more anarchic community projects.9

## 3. Technical Architecture and Specifications

POML is formally defined as an XML-inspired markup language that separates the *logical structure* of a prompt from its *presentation*. Its architecture rests on three pillars: Semantic Componentization, Specialized Data Integration, and Decoupled Styling.

### 3.1 Semantic Componentization: The Grammar of Intent

At the highest level, a POML document is enclosed in a <poml> root tag. Inside, the language enforces a separation of concerns through "Intention Components".11 These components map distinct cognitive functions of the LLM to specific tags, moving beyond the amorphous "system prompt" concept.

#### 3.1.1 The <role> Component

The <role> tag formalizes the definition of the AI's persona. In traditional prompting, this might be a sentence like "You are a helpful assistant" buried in a text block. In POML, it is a structural element:

XML

<role>You are a patient teacher explaining concepts to a 10-year-old.</role>

**Theoretical Implication:** By isolating the persona, the POML engine can treat this information differently during execution. For instance, it can map the content of <role> to the "System Message" field in the OpenAI API, or to a high-priority attention mask in a local model. This ensures that the persona definition is "sticky" and persists throughout the interaction, resisting the "catastrophic forgetting" often seen in long context windows.3

#### 3.1.2 The <task> Component

The <task> tag defines the teleological goal of the prompt—the "what" to the role's "who."

XML

<task>Explain the concept of photosynthesis using the provided image as a reference.</task>

Separating the task from the role allows for modularity. A developer can define a library of Roles (e.g., "Teacher," "Editor," "Coder") and a library of Tasks, mixing and matching them dynamically. This modularity is essential for "Agentic" workflows where a single agent might need to switch tasks (e.g., from "Plan" to "Execute") while maintaining the same persona.3

#### 3.1.3 The <example> Component

Few-shot prompting—the technique of providing examples to guide the model—is codified in the <example> tag.

XML

<example>  
 <input>What is 2+2?</input>  
 <output>4</output>  
</example>

**Design Decision:** This explicit structure addresses a common failure mode in unstructured prompting where the model confuses the few-shot examples with the actual user query. By enclosing examples in semantic tags, the POML parser can format them with clear delimiters (e.g., "### Example 1"), reducing the likelihood of "leaky context".3

### 3.2 Specialized Data Integration: The Multimodal Layer

One of the most significant pain points POML addresses is the integration of external data. The language introduces "Data Components" that abstract the complexity of serialization.11

#### 3.2.1 The <document> Tag

The <document src="file.pdf" /> tag allows developers to reference external files directly. The POML engine handles the file reading, text extraction, and tokenization. This feature, described as "lazy loading" for prompts, keeps the source code clean and readable while enabling the inclusion of massive datasets.3 It eliminates the need for developers to write boilerplate Python code to open files and paste their contents into strings.

#### 3.2.2 The <table> Tag

Tabular data is notoriously difficult for LLMs to parse if formatting is inconsistent. The <table> tag allows data to be embedded in a structured format (CSV or XML-like rows).

XML

<table>  
 <header>Name, Age, Role</header>  
 <row>Alice, 30, Engineer</row>  
</table>

**Mechanism:** At runtime, the POML engine renders this tag into the format most optimal for the specific model being called (e.g., Markdown table for GPT-4, JSON for Claude). This abstraction layer insulates the developer from the quirks of individual model tokenizers.3

#### 3.2.3 The <img> Tag and Vision Support

Reflecting the multimodal nature of modern models (GPT-4o, Claude 3.5 Sonnet), POML includes first-class support for images via the <img> tag. This allows visual data to be treated as a standard context input, co-located with text instructions.

XML

<img src="photosynthesis\_diagram.png" alt="Diagram of photosynthesis" />

This design decision normalizes "vision" as just another form of context, demystifying multimodal prompting.9

### 3.3 The Styling Layer: Decoupling Content from Presentation

Perhaps the most radical innovation in POML is the introduction of <stylesheet>, a concept borrowed directly from CSS. The authors argue that prompt engineering involves two distinct activities: designing the *logic* (content) and optimizing the *format* (presentation).13

#### 3.3.1 The <stylesheet> Mechanism

A stylesheet allows a developer to define formatting rules separately from the content:

XML

<stylesheet>  
 role { captionStyle: "bold"; }  
 task { verbosity: "concise"; }  
</stylesheet>

**Implication:** This decoupling addresses the "Format Sensitivity" problem. If a specific model (e.g., Llama 3) responds better to bold headers, the developer can apply a specific stylesheet. If they switch to GPT-4, which prefers natural language delimiters, they can switch the stylesheet without rewriting the core prompt logic. This feature enables "Cross-Model Portability," a critical requirement for enterprises seeking to avoid vendor lock-in.9

### 3.4 Templating and Logic Control

POML is not a static markup language; it is dynamic. It incorporates a templating engine (similar to Jinja2) that supports:

* **Variable Injection:** {{user\_name}}
* **Control Flow:** <if condition="is\_vip">
* **Loops:** <for item in history>
* **Variable Definition:** <let name="context" value="..." />

This hybrid approach—combining declarative markup with imperative logic—positions POML as a "Markup Programming Language," blurring the line between data and code.14

### Table 1: Core POML Component Taxonomy

| **Component Category** | **Tag** | **Purpose** | **Semantic Function** |
| --- | --- | --- | --- |
| **Intention** | <role> | Persona Definition | Establishes the epistemic authority and voice. |
| **Intention** | <task> | Objective Definition | Defines the teleological goal of the generation. |
| **Intention** | <example> | Few-Shot Demonstration | Provides pattern-matching templates for the model. |
| **Data** | <document> | External Reference | Ingests and serializes unstructured text files. |
| **Data** | <table> | Structured Data | Serializes tabular data into model-optimal formats. |
| **Data** | <img> | Visual Context | Embeds multimodal inputs (Vision). |
| **Presentation** | <stylesheet> | Format Control | Decouples prompt logic from token representation. |
| **Logic** | <let>/<if> | Control Flow | Enables dynamic, conditional prompt generation. |

## 4. Narrative Structures and Digital Humanities Applications

While POML was engineered for efficiency, its structured nature has found profound resonance in the Digital Humanities (DH) and the field of interactive storytelling. The language's architecture aligns with the "Narrative Structuralism" often employed in literary analysis, treating a text as a system of functional components.

### 4.1 The "Soft-Token" Theory and Narrative Control

Research in "ICL Markup" (In-Context Learning Markup) suggests that using "soft-token tags" (like those in POML) helps to compose prompt templates that reduce arbitrary decisions.15 For DH scholars, this means POML can be used to construct "Generalist-Specialist" frameworks. A scholar might define a "Generalist" historical context in a root POML file, which then imports "Specialist" tasks (e.g., "Analyze this text from a Marxist perspective," "Analyze from a Feminist perspective"). This modularity allows for the rigorous, reproducible analysis of literary texts, moving DH away from subjective "chatting" with bots toward systematic inquiry.15

### 4.2 Interactive Storytelling: The Photosynthesis Archetype

The canonical example provided in the POML documentation—a "patient teacher" explaining photosynthesis—serves as a narrative archetype.9

XML

<poml>  
 <role>You are a patient teacher explaining concepts to a 10-year-old.</role>  
 <task>Explain the concept of photosynthesis...</task>  
 <output-format>Start with "Hey there, future scientist!".</output-format>  
</poml>

**Analysis:** This structure enforces a specific *narrative voice* and *audience alignment*. The <output-format> tag acts as a "Directorial Note," constraining the improvisation of the AI actor. For interactive fiction writers, this provides a mechanism to ensure character consistency. The <conversation> tag further extends this by allowing the pre-seeding of a dialogue history, effectively setting the "scene" before the user enters the stage.16

### 4.3 Digital Humanities Infrastructures

In archival contexts, POML is being explored as a method for "active metadata." Traditional metadata (like Dublin Core) describes a document. POML allows archivists to wrap a document in a *performative* layer, defining how an AI should interpret and present that document to a user. For example, a POML wrapper around a digitized 19th-century letter could include a <role> tag defining the author's biography and a <task> tag instructing the AI to answer questions *as* the author, effectively resurrecting the archive as an interactive interface.4

## 5. Ecosystem Adoption and Technological Integration

The success of a markup language depends heavily on its tooling ecosystem. POML's rapid adoption in late 2025 was driven by a concerted effort to build a "Rich Development Toolkit".9

### 5.1 The VS Code Extension and "IntelliSense"

The release of a dedicated Visual Studio Code extension was a critical adoption driver. By providing **IntelliSense** (auto-completion), syntax highlighting, and inline error diagnostics, the extension transformed prompt engineering from a text-editing task into a *development* task.9

* **Live Preview:** The ability to render the POML file and see the exact text that would be sent to the API gave developers confidence in the abstraction.
* **Static Analysis:** The extension could flag errors like "File not found" in a <document> tag *before* the prompt was sent, saving money on wasted API calls.

### 5.2 The Model Context Protocol (MCP) Integration

A significant evolution in the POML ecosystem is its integration with the **Model Context Protocol (MCP)** via the poml-mcp server.17 MCP is a standard for connecting AI models to external data and tools.

* **Mechanism:** The poml-mcp server allows an AI agent to "discover" POML templates as tools. An agent can query the server, find a "Summarize" POML template, and execute it.
* **Impact:** This turns POML files into "executable skills" that agents can dynamically load and use. It moves POML from being a tool for humans to write prompts, to a tool for *agents* to orchestrate their own sub-tasks. The poml-mcp project, hosted on GitHub (iberi22/POML-MCP), enables the automatic conversion of free-text briefs into structured POML, creating a loop where agents can refine their own instructions.17

### 5.3 Cross-Language Implementations

While the official SDKs targeted TypeScript and Python, the community quickly ported POML to other languages, demonstrating the demand for the format.

* **Ruby:** The poml-ruby gem (by GhennadiiMir) brought POML to the Rails ecosystem, supporting image processing via libvips.18
* **Rust:** The mini-poml-rs project provided a high-performance renderer for systems where Python/JS overhead was unacceptable.13
* **Julia:** A PomlSDK.jl was developed for the scientific computing community.13

This diverse ecosystem confirms that POML addressed a universal pain point across different programming cultures.

## 6. Critical Reception and Controversies

The reception of POML was not universally positive. It ignited a fierce debate within the developer community about the nature of abstraction in AI.

### 6.1 The "Reinvention of XML" Critique

The most vocal criticism came from developers who saw POML as a regression to the verbose, complex standards of the "XML Era" (late 90s/early 2000s).

* **"XSLT Redux":** Commenters on Reddit compared POML's templating and stylesheets to XSLT (Extensible Stylesheet Language Transformations), a technology infamous for its complexity. "It's a template system that got out of hand and reinvented XSLT again," noted one user.19
* **"JSX but Worse":** The mix of markup and imperative logic (<if>, <for>) drew comparisons to JSX (React's syntax), but with criticism that it was "shoehorning" logic into strings.20

### 6.2 The "Markdown vs. Markup" Debate

A fundamental ideological split emerged between "Minimalists" and "Structuralists."

* **Minimalists:** Argued that since LLMs are trained on the internet (which is largely HTML and Markdown), they naturally understand Markdown. Therefore, a new tag-based language was unnecessary overhead. "Why not just use Markdown?" was a common refrain.19
* **Structuralists:** Argued that Markdown is too ambiguous for *engineering*. Markdown describes formatting (bold, header), not semantics (Role, Task). POML proponents argued that for enterprise reliability, the verbosity of XML tags was a necessary cost for semantic precision.14

### 6.3 Security Challenges: The "Shai-Hulud" Vulnerability

The ecosystem faced its first major security crisis with the "Shai-Hulud" vulnerability in the pomljs library (version 0.0.9-nightly). Security analysis by Socket.dev revealed that the package contained obfuscated code and used eval(), creating a risk of dynamic code execution.10

* **Significance:** This incident highlighted the risks of the rapid "nightly" release cycle and the danger of treating prompts (which are effectively code) as safe, static assets. It forced a conversation about "Prompt Injection" moving from a theoretical risk to a supply-chain vector.

### Table 2: Comparative Analysis of Prompting Paradigms

| **Feature** | **POML (Microsoft)** | **BAML (BoundaryML)** | **LangChain** | **DSPy** |
| --- | --- | --- | --- | --- |
| **Core Philosophy** | **Document-Centric:** The prompt is a structured text file. | **Function-Centric:** The prompt is a typed function. | **Code-Centric:** The prompt is an object in a pipeline. | **Optimization-Centric:** The prompt is a parameter to be tuned. |
| **Primary Syntax** | XML-like Markup (<tag>) | Custom DSL (Jinja-like + Types) | Python/JavaScript Classes | Python Declarative Modules |
| **Output Handling** | Text-based instructions (<output-format>) | **Type-Safe:** Returns Pydantic objects. | Output Parsers (Code) | Signatures (input -> output) |
| **Data Integration** | Native Tags (<document>, <img>) | Schema Definitions | Data Loaders | Datasets |
| **Target User** | Enterprise Engineer, DH Scholar, Product Manager | Backend Engineer, Systems Architect | ML Engineer, Data Scientist | AI Researcher, Optimizer |
| **Best For...** | Readability, Collaboration, Archival | Reliability, API integration, Structured Data | Complex Chains, Memory Management | Auto-optimizing prompt performance |

## 7. Broader Discourse: Declarative vs. Imperative Prompting

POML sits at the center of a theoretical shift from **Imperative Prompting** to **Declarative Prompting**.

* **Imperative:** Telling the model *how* to do something step-by-step (e.g., "First read the text, then extract entities, then format as JSON").
* **Declarative:** Telling the model *what* is required (e.g., <task>Extract entities in JSON format</task>).

Research indicates that declarative prompting often yields higher robustness because it allows the model to determine the optimal execution path.21 However, POML is unique in that it is a *hybrid*. It uses declarative tags for structure but allows imperative control flow (<if>, <for>) for logic. This "Markup Programming" paradigm acknowledges that while the *goal* is declarative, the *process* of constructing the context often requires imperative logic.14

## 8. Conclusion: The Standardization of Synthetic Thought

The emergence of the Prompt Orchestration Markup Language represents a maturation point for Generative AI. It signals the end of the "wild west" era of string manipulation and the beginning of a disciplined, engineering-focused approach to AI interaction.

While critics may argue that its XML-like syntax is retrogressive, this verbosity serves a vital purpose: it creates a "boundary object" that is readable by humans, parseable by machines, and semantically rich enough to capture the nuance of human intent. For the AI engineer, POML provides the reliability and modularity needed to build scalable systems. For the Digital Humanist, it offers a rigorous framework for narrative construction and archival interaction.

As AI systems evolve into "Agentic" networks that communicate with one another, the need for a standardized interchange format—a lingua franca of intent—will only grow. Whether POML becomes the HTML of the AI age or remains a specialized tool for enterprise orchestration, its core contribution is undeniable: it has proven that the prompt is not merely a string of words, but an architecture of thought.

### Key Takeaways

1. **Structure as Semantics:** POML redefines the prompt as a hierarchical document, solving the "context soup" problem of early LLM development.
2. **Decoupling:** The separation of content (Intent) from presentation (Style) allows for cross-model portability, a crucial feature for enterprise independence.
3. **Multimodal Nativism:** POML treats images and documents as first-class citizens, simplifying the complex engineering of multimodal pipelines.
4. **Ecosystem Maturity:** The rapid emergence of tools like poml-mcp and cross-language SDKs demonstrates a strong market demand for structured prompting.
5. **The New Literacy:** POML suggests that the "coding" of the future will look less like writing algorithms and more like writing structured specifications for AI reasoning.

#### Works cited

1. What is POML (Prompt Orchestration Markup Language)? - igmGuru, accessed December 10, 2025, <https://www.igmguru.com/blog/prompt-orchestration-markup-language-poml>
2. Prompt Orchestration Markup Language - arXiv, accessed December 10, 2025, <https://arxiv.org/abs/2508.13948>
3. Prompt Orchestration Markup Language (POML) - Emergent Mind, accessed December 10, 2025, <https://www.emergentmind.com/topics/prompt-orchestration-markup-language-poml>
4. What-is-POML-A-Revolution-in-AI-Prompting (1) | PDF - Scribd, accessed December 10, 2025, <https://www.scribd.com/document/930564629/What-is-POML-A-Revolution-in-AI-Prompting-1>
5. Daily Papers - Hugging Face, accessed December 10, 2025, [https://huggingface.co/papers?q=modular%20toolkits](https://huggingface.co/papers?q=modular+toolkits)
6. Nan Chen's homepage, accessed December 10, 2025, <https://cxxxxxn.github.io/>
7. Beyond Prompt Content: Enhancing LLM Performance via Content-Format Integrated Prompt Optimization - ResearchGate, accessed December 10, 2025, <https://www.researchgate.net/publication/388792075_Beyond_Prompt_Content_Enhancing_LLM_Performance_via_Content-Format_Integrated_Prompt_Optimization>
8. VIDEE: Visual and Interactive Decomposition, Execution, and Evaluation of Text Analytics with Intelligent Agents - ResearchGate, accessed December 10, 2025, <https://www.researchgate.net/publication/393148934_VIDEE_Visual_and_Interactive_Decomposition_Execution_and_Evaluation_of_Text_Analytics_with_Intelligent_Agents>
9. microsoft/poml: Prompt Orchestration Markup Language - GitHub, accessed December 10, 2025, <https://github.com/microsoft/poml>
10. pomljs - npm Package Security Analysis - Socket.dev, accessed December 10, 2025, <https://socket.dev/npm/package/pomljs>
11. Prompt Orchestration Markup Language - arXiv, accessed December 10, 2025, <https://arxiv.org/html/2508.13948v1>
12. Spotlight on POML : r/PromptEngineering - Reddit, accessed December 10, 2025, <https://www.reddit.com/r/PromptEngineering/comments/1mmrrcm/spotlight_on_poml/>
13. POML Documentation - Microsoft Open Source, accessed December 10, 2025, <https://microsoft.github.io/poml/latest/>
14. Microsoft released POML : Markup Programing Language for Prompt Engineering - Reddit, accessed December 10, 2025, <https://www.reddit.com/r/LocalLLaMA/comments/1mquliu/microsoft_released_poml_markup_programing/>
15. Daily Papers - Hugging Face, accessed December 10, 2025, [https://huggingface.co/papers?q=specialized%20tags](https://huggingface.co/papers?q=specialized+tags)
16. Components - POML Documentation - Microsoft Open Source, accessed December 10, 2025, <https://microsoft.github.io/poml/latest/language/components/>
17. MCP server that enhances and structures prompts using POML best practices for more reliable agent workflows. - GitHub, accessed December 10, 2025, <https://github.com/iberi22/POML-MCP>
18. GhennadiiMir/poml: Ruby implementation of Prompt Orchestration Markup Language - GitHub, accessed December 10, 2025, <https://github.com/GhennadiiMir/poml>
19. Microsoft releases Prompt Orchestration Markup Language : r/LocalLLaMA - Reddit, accessed December 10, 2025, <https://www.reddit.com/r/LocalLLaMA/comments/1mo9vkh/microsoft_releases_prompt_orchestration_markup/>
20. POML: Prompt Orchestration Markup Language | Hacker News, accessed December 10, 2025, <https://news.ycombinator.com/item?id=44853184>
21. Declarative and Imperative Prompt Engineering for Generative AI | Towards Data Science, accessed December 10, 2025, <https://towardsdatascience.com/declarative-and-imperative-prompt-engineering-for-generative-ai/>
22. arXiv:2305.09656v2 [cs.CL] 17 May 2023 - OpenReview, accessed December 10, 2025, <https://openreview.net/pdf?id=jE_jdMekThY>
23. Three Ways to program with LLMs - Sean Wu - Medium, accessed December 10, 2025, <https://opencui.medium.com/three-ways-to-program-with-llms-b8d3027fbd63>