

Heart of Darkness

A Journey in Vibe Coding an XML-Aware Application

Hermetic Foundry - Task

- Add schema validation for Character and Covenant editors
- Preserve xs:annotation and xs:documentation when editing schemas
- Run: `pnpm check`
- Run: `pnpm test`
- Do not persist changes on Apply; save writes to disk only
- Keep all changes unstaged until review
- Support pictures for Character, Creature, Covenant, Laboratory, and Item
- Verify Quick Item additional profiles persist after save

```
<xs:element name="character" type="CharacterType">
  <xs:annotation>
    <xs:documentation>Hermetic Foundry character schema.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="CharacterType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="covenant" type="CovenantRefType" minOccurs="0"/>
    <xs:element name="virtues" type="VirtuesType" minOccurs="0"/>
    <xs:element name="abilities" type="AbilitiesType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

Ben Wolfe

Know One Systems

```
<xs:complexType name="CovenantType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="house" type="HouseType"/>
    <xs:element name="regio" type="xs:string"
      minOccurs="0"/>
    <xs:element name="resources" type="ResourcesType"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

```
``ts
const result = await validateSchema(xmlDocument);
``ts

<xs:element name="item" type="ItemType">
  <xs:complexType name="ItemType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="description" type="xs:string" minOccurs="0"/>
      <xs:element name="properties" type="PropertiesType" minOccurs="0"/>
      <xs:element name="images" type="ImagesType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

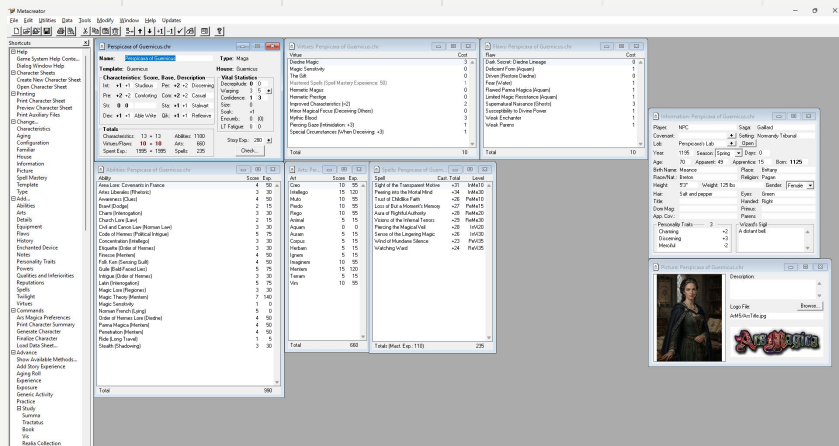
- Validate against Hermetic schemas before commit
 - Ensure all documentation is updated
 - Maintain backward compatibility
- Run: `pnpm lint && pnpm build`

Balisage 2026



How It All Started

- What is “Ars Magica?”
- Metacreator was used by most for it



Press F1 for game-system specific help, or Shift+F1 for general help.

- But Metacreator was discontinued!
- Atlas Games open sourced the entire Ars Magica corpus
- “Why, it’s so simple. I can build a replacement!”
- This looks like a job for...XML!

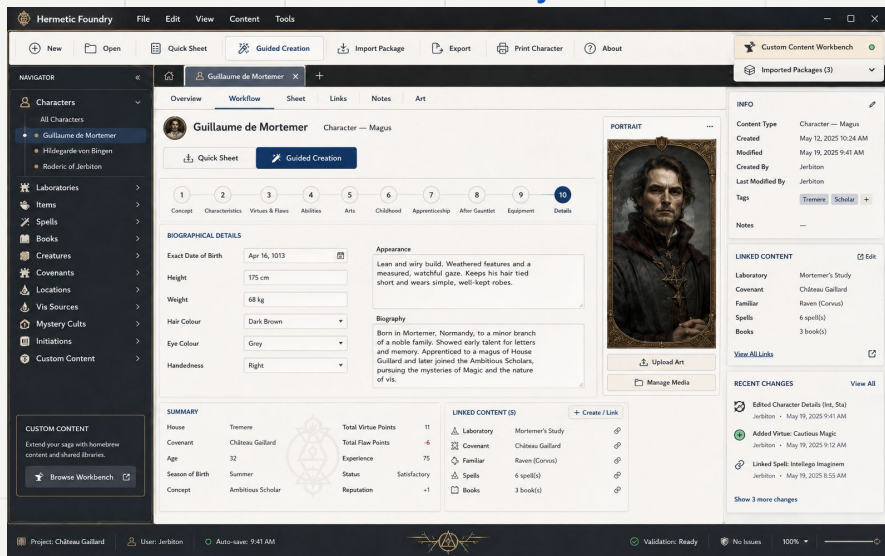
It's a Data Problem

- It's not just a simple character generator.
- Characters in Ars Magica are connected to:
 - Decades of in-game advancement
 - Characteristics, Abilities, Virtues, Flaws, Spells, etc.
 - Interconnected objects like Laboratories, Familiars, Allies, etc.
 - Loosely connected objects like locations, equipment, books, etc.
 - Packages of rules content
 - Core rules content
 - Supplemental rules content
 - Third party rules content
 - Homebrew rules content, too?!
- Replacing Metacreator requires something that can handle a network of long-lived documents.
- *XML to the rescue!*

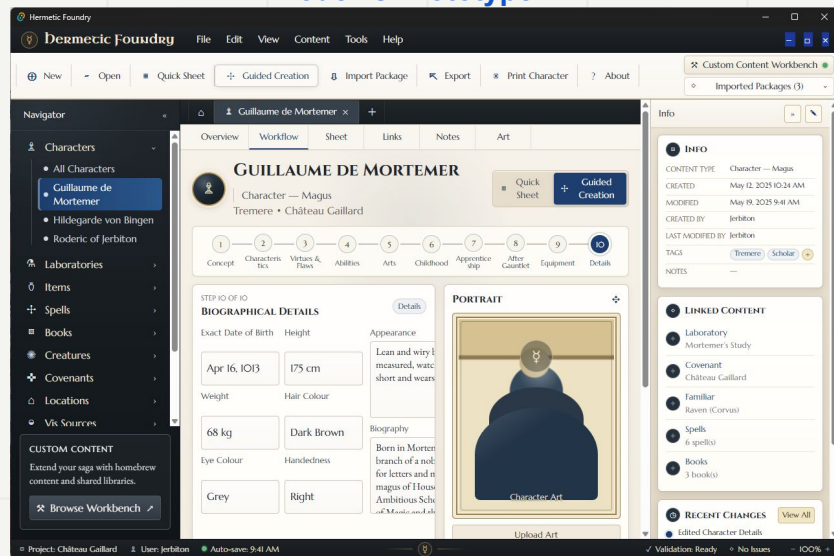
The Skill Prosthetic

- I have extensive backend, data processing, and XML experience.
- I also have the design skills of a rock.
- “What if I asked ChatGPT to mock something up?”
- ChatGPT told me Codex could make it into a working prototype.

ChatGPT's First Try



Codex's Prototype

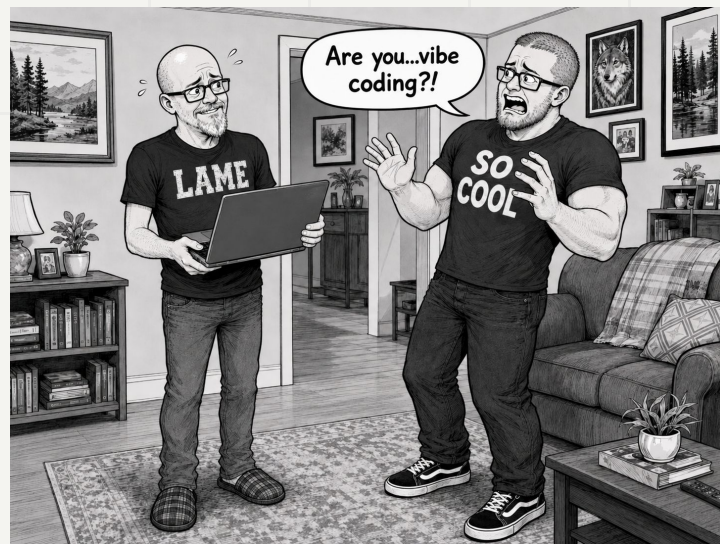


"Are You...*Vibe Coding*?!"

- I showed the prototype to my partner, because she's artistically-inclined
- A teenager discovers his father's...*secret shame!*
- Accusations were made.
- Accusations were denied.
- ...wait a minute.

What's "Vibe Coding?"

- No one seems to agree!
- Definitions range from:
 - Careless prompting with no oversight or inspection, to
 - Developers acting as "managers" for an LLM.
- I seemed to be doing the latter, albeit unintentionally.



An Experimental Notion

“How far could ChatGPT and Codex take a substantial, XML-aware application before the process crashed and burned?”

Conditions –

- Let the LLMs implement as much as they could.
- Touch the code myself as little as possible.
- Intervene directly only when delegation failed so much that it became foolish.
- Judge by total human effort, not generation speed.
- Bad XML would not be accepted, even if the product functioned.

Who Did What?

BEN

Definer

Product vision
Domain model
XML architecture
Acceptance criteria

CHATGPT

Translator

Research
Prose into instructions
Report summaries
Intent into bounded tasks

CODEX

Implementer

Application changes
Automated testing
Repository management
Completion reports

TOOLS

Verifiers

Compilation/building
Automated testing
XML validation
Git diffs/version history

The Development Loop



Requirement



ChatGPT
Task Drafting



Human
Review



Codex
Reconnaissance



Human
Review



Codex
Implementation



Tests and
Validation

Success Parameters

The Five Questions That Defined Success

1. Was I saving time?
2. Was the code maintainable?
3. Was the XML good? Did it follow best practices?
4. Were defects manageable?
5. Was the result at least as good as conventional development?

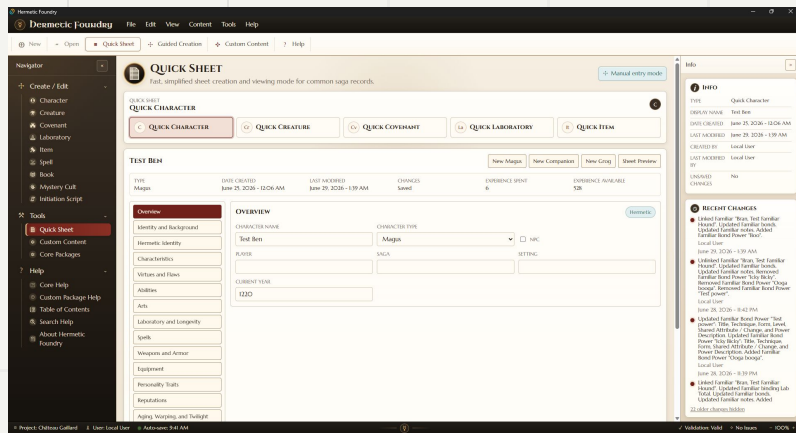
These questions informed every part of the process. If, at any point, the answer was “no” to any of them, the experiment would be abandoned as a failure.

What I wasn't expecting was every answer to be a resounding “yes.”

It Stopped Being a Prototype

The project's scale at the time I wrote the paper for Balisage:

- 262 tracked text files.
- 77,759 nonblank tracked lines.
- 57,135 lines of Svelte, TypeScript, Rust and CSS.
- XML Schemas, packages, tests, fixtures and scripts.
- Approximately 25% complete at that point.



<?xml>



Why XML?

Why is XML the Best Choice?

Hermetic Foundry's objects are recognizable documents with:

- Nested structure.
- Ordering.
- Repetition.
- Provenance requirements.
- Relationships between objects.
- Strict validation requirements.
- Descriptive, prosaic content.
- Rules-based content.
- Extensibility for optional material.

A character isn't a row with an alarming number of columns.

This looks like a job for *XML*!

Good XML was a Necessity

Hero Designer Export Template

```
<!--TEMPLATE_NAME-->HTML - Default
Character
Sheet<!--/TEMPLATE_NAME--><!--TEMPLATE_DESC
RIPTION-->
This is the default HTML format for Hero
Designer characters.
<p>
If printing is desired, this format will
only break pages properly if you print
it using Internet Explorer (version 6 or
better) under Windows. Even with IE,
however,
it is entirely possible that pages will not
be broken properly for characters with a
large number of Skills, Perks, Talents,
Powers, or Martial Arts.
<!--/TEMPLATE_DESCRIPTION--><!--FILE_EXTENS
ION-->html<!--/FILE_EXTENSION--><!--FILE_EX
TENSION-->htm<!--/FILE_EXTENSION--><HTML>
<HEAD>
<TITLE><!--CHARACTER_NAME--></TITLE>
```

Aurora Building Character Sheet

```
<?xml version="1.0" encoding="utf-8"?>
<character version="1.0.3" preview="false">
  <!-- Aurora -
https://www.aurorabuilder.com -->
  <!-- information -->
  <information>
    <group>Characters</group>
    <generationOption>1</generationOpti
on>
  </information>
  <!-- display data -->
  <display-properties favorite="false">
    <name>Alanias Windspun</name>
    <race>High Elf</race>
    <class>Wizard</class>
    <background>Noble</background>
    <level>18</level>
    <portrait>
      <local>elf-male-wizard.png</lo
cal>
      <base64><![CDATA[iVBORw0KGgoAA
AANSUhEUgAAASIAAAEGCAIAAABO4Q/BAAAAAXNSR0IArs4
c6QAAAARnQU1BAACxj
```

Defining “Good XML”

My loose requirements for “good XML” were –

- Human-readable.
- Independently validates.
- Portable between versions of the software.
- Gracefully transformable.
- Suitable for version control.
- Broad enough to cover the entire Ars Magica corpus.
- Tightly defined enough to feed reliably into Hermetic Foundry.
- Capable of outliving Hermetic Foundry.
- Capable of being used in other software by programmers lacking XML expertise.

Why XML Schema?

XSDs fit because they have –

- Reusable complex types.
- Strong datatyping.
- Namespaces.
- Facets, including enumerations.
- Identity constraints.
- Explicit schema versioning.
- Broad, open source tool support.
- And especially self-documenting structures like annotations!
 - Not for humans – I never intended for humans to even see the XSDs.
 - I realized early on that the annotations could provide context and instructions for the LLMs!

XML in Hermetic Foundry

Ars Magica corpus



Content extraction



Versioned rules packages → XML Schema validation



Canonical XML documents ← Semantic/application checks



Hermetic Foundry domain model and UI



Saved XML → validation → reload / interchange / transformation

XML Schema structural supports



Future work will add a new dynamic after the last validation – PACKAGING!

Getting into the Schema Vibe

I stumbled into best practices for coding with LLMs.

Being a “data first” programmer, I always define the inputs and outputs of a program before the user interface.

With trepidation, I gave ChatGPT the instructions:

- Use XML industry practices.
- Analyze the Ars Magica corpus.
- Be sure to liberally annotate.
- Reuse repeating structures.
- Organize a list of first-class entities from Ars Magica.



...Am I Obsolete?

The results were shocking.

The first version of the XSDs were nearly flawless. They had:

- Meaningful domain understanding.
- Reusable shared structures.
- Package definitions clearly delineated from character-owned data.
- Extensive, meaningful annotation.
- Very, very few misunderstandings of context.
- No hallucinations! There were no fabricated domain concepts or nonexistent XSD constructs.
- And here's where it outdid me entirely – reference identity and extensive provenance.

Let's take a look at the schemas!

Digging into the Schemas



```
<xml carousel />
```

The Unexpected Results

What was good –

- Every schema validated with Xerces.
- The LLMs didn't need to be taught XSD.
- Revisions preserved the overall design.
- Annotations were clear and useful.

The big surprises –

- ChatGPT asked about contradictions in the corpus.
- Regression testing was assumed, not taught.
- Provenance was way beyond what I would have designed, including:
 - Canonical and stable IDs
 - Checksums
 - Machine and human-readable versioning

Extracting Content

I had schemas. Now what?

A “data first” approach meant the next step was extracting the corpus.

The process –

- Started with extracting from PDFs.
 - That went surprisingly well, including ChatGPT once again finding errors and contradictions in the source.
 - But it often failed due to confusing layout issues; however, it asked about failures instead of assuming it was correct.
- “A Markdown corpus...?!”
 - This simplified things dramatically.
 - But there were still issues with examples and sidebars.
 - Markdown sucks; XML rules.

Codex's Time Arrives

ChatGPT had already suggested the technology stack for Codex during the sad, bygone days when I thought I was just rapid prototyping: SvelteKit, Typescript, Tauri, and Rust.

I stuck with those, because I understood their fundamentals, but they were also new to me. It was a learning opportunity.

But I didn't know how to use Codex, so I asked ChatGPT for lessons.

I quickly realized that ChatGPT could simply take my prosaic requests and translate them into detailed requirements for Codex.

I had no idea I was recreating a traditional development hierarchy – product manager (me), project manager (ChatGPT), and developer (Codex).

Toolchain and Process

I laid down parameters for development, with reports to me at each stage.

1. REQUIREMENT & SPECIFICATION



- ChatGPT creates a detailed task specification
- Human review and refinement
- Acceptance criteria are explicit

2. IMPLEMENTATION (CODEX)



- Codex inspects the repository
- Implements code, schemas, tests, and assets
- Commits to the task branch

3. BUILD & COMPILE



- Full application build
- Type checking
- Linting
- Bundle/package creation

4. AUTOMATED TESTING



- Unit tests
- Integration tests
- End-to-end tests
- Fixtures
- All tests must pass

5. XML & XSD VALIDATION



- Well-formedness (XML parser)
- XSD validation (Xerces)
- Additional schema checks
- No errors allowed

6. MANUAL VERIFICATION



- Manual save/reload verification
- Data inspection in XML Carousel
- Business-rule spot checks

7. REVIEW & MERGE



- Human code & schema review
- Git diff review
- Only then merge to main
- Traceable history

Where Things Went Wrong

The user interface design started to drift.

I was several screens in before I realized that each screen was developing quirks that were soon becoming issues.

What happened?!

What happened was this –

- I suck at user interface design.
- I never specified a style guide.
- Without specific instructions, Codex had minimal context.
- So it just made stuff up.

I had to make a style guide and then go a few steps back and refactor.

Where Things Went Weird

I began to notice that processes with Codex were taking longer and longer.

It went on for some time before I began to question it.

I had gotten lazy. 🙄 Things were going so well that I stopped reading every instruction in-depth.

ChatGPT had begun to acquire *context bloat* –

- Old branch instructions persisted.
- Superseded requirements remained visible.
- Codex was told to inspect unrelated systems.
- Obsolete content preserved obsolete workarounds.

Curating the instructions dropped completion times by about 30%.

Where Tools Caused Problems

The biggest problems were, surprisingly, not with LLM reasoning.

They were with tools!

- Codex had frequent sandbox issues.
- Intermittent build failures occurred because Codex was working faster than the system could manage.
- Constant authorization pauses and timeouts were a problem.
- OpenAI dropping Codex and ChatGPT releases way too fast and without any useful documentation caused new bugs in the tools and rewired their user interfaces in confusing ways.
- Hosted service outages were a rare, but significant issue.
- Guardrail oddities sometimes caused stuttering in the process.

Depending on third party infrastructure you can't control is annoying!



The Experiment is a Success

Success Parameters Met

1. Was I Saving Time?

At cutoff time for the paper, I had spent a month on this and produced a pre-alpha program. My rough estimate for how much time this would've taken me is about six months, and it would not have been as polished.

My single controlled experiment at comparing my speed to the LLMs ended up like this: I refactored one part of one screen in a work day. Codex refactored the entire thing in the same time, producing 8 times as much work.

Success Parameters Met

2. Was the Code Maintainable?

Yes.

The code produced was readable, sensibly organized, and well-documented.

I never had a problem finding what I needed in order to check and/or fix things.

Success Parameters Met

3. Was the XML Good?

This is where things were the most questionable.

Yes, the XML was good. Very good. Not as readable as I might have liked, but very acceptable.

The data mining, on the other hand, aptly demonstrated that LLMs will often fail at understanding subtle context. In this case, “What’s a rule and what’s an example?”

Success Parameters Met

4. Were Defects Manageable?

Eminently so.

The most frustrating defect, the user interface drift, cost me a day of time to fix, and produced the most manual effort of the project, as I had to write a style guide, which involved research, asking more skilled friends to go over it, and then ensuring ChatGPT and Codex understood it.

Success Parameters Met

5. Was the Result At Least as Good as Conventional Development?

Yes. In fact, by the success criteria I established, it's better.

Hermetic Foundry, at a pre-alpha stage, is already a smoothly-operating program. ChatGPT's insistence on regression testing at every phase dramatically improved the quality of the application.

One of the biggest issues I've had is having to refer back to my development plan because it *feels* feature-complete, even though it's not.

Limitations of the Experiment

- One developer, one domain, and one substantial application.
- Exact development period and model/tool versions.
- Extensive human review and domain expertise remained essential.
- Not a controlled comparison with a conventionally developed application.
- The “8 times faster” comparison came from one bounded refactoring task. Depending on the task, speed differences ranged from 4 times faster to 50+ times faster, but those were informal checks.
- Schema validity was independently checked by the tools. I assessed maintainability.
- Results should not automatically be generalized to every developer or domain, as persuasive as they’ve been for me.

Ongoing Challenges

- Semantic validation beyond XSD – context, context, context!
- Clearing out instructional cruft.
- Managing my own expertise limitations and the trickle-down effect of those weaknesses.
- Combating laziness – big errors crop up when things are going so well that the human gets careless.
- Third party reliability.
- Ethical questions about LLM training and environmental impact.
 - What to use that doesn't feel stolen?
 - Are the corporations governing these tools ethical enough?
 - How much environmental damage are they doing?

The big takeaway: Humans are more problematic than LLMs.

What Would I Have Changed?

Pre-Built Structure

- I would have documented programming principles, style guides, etc. before ever engaging the LLMs.
- I would have done it all in well-annotated, domain-specific XML, rather than Markdown (I've already started experimenting with this).

Stick it to the Man

- I would abandon the corporate LLMs (doing this now, too).
- I would disengage from other third party dependencies as much as possible.

Whither XML?

LLMs can participate reliably in sustained XML work.

In my work, there were no hallucinations, just misclassifications. This demonstrates a significant evolution in LLMs and XML from previous work in the field. And they understood XSD design!

Markdown sucks; XML rules...and *XML creates context!*

LLMs fail reliably in comprehending context. When presented with Markdown, they made serious errors.

When presented with XML, they were provided with context.

We can use XML to *provide context that LLMs cannot ever discern!*

Resources, Questions, and Gratitude

Hermetic Foundry and XML Carousel are on my GitHub:

<https://github.com/benjaminisawolfe>

This slideshow:

My company: <https://knowone.ca>

Thanks to the Balisage team for having me and for running this excellent conference for all these years!

Questions? Comments? Tomatoes?