nex/spartan/gemini/titan

Slide 1 — Nex: the Minimal Protocol

- **Purpose:** Experimental ultra-simple text protocol for small or local hypertext systems.
- **Design goal:** simpler than HTTP, no headers, no MIME.
- **Request:** a single line — the full URL followed by CRLF.

  `nex://example.com/page\r\n`
- **Response:** starts with a single line containing a *status code* and optional *meta* (like MIME type), followed by content.

  ```
  20 text/plain\r\n
  Hello world
  ```
- **Status classes:** 1x input, 2x success, 3x redirect, 4x temporary failure, 5x permanent failure.

Slide 2 — Spartan: Structured Simplicity

- **Goal:** similar simplicity to Gemini but includes optional upload capability.
- **Tokens:** `SP` = a single ASCII space (0x20). `CRLF` = ‘

  ‘.
- **Request format:**

  `host SP path-absolute SP content-length`

  `[data-block]`
  - `content-length` is **decimal bytes** of the following data block. Use `0` for no upload.
  - The `path-absolute` **must** start with `/`.
- **Response format:** single status line then optional body.

  `2 SP mimetype`

[body] 3 SP /new/path

4 SP human-readable error

5 SP human-readable error

```
- **Default hypertext:** `text/gemini` (Gemtext). `text/plain` is common too.

### Spartan - End-to-End Examples

**A. Download a page (no upload)**
```

Client → Server

example.com /hello.gmi 0

Server → Client

2 text/gemini

Hello Spartan

- This is Gemtext served over Spartan.

```
**B. Submit small text (upload 18 bytes)**
```

Client → Server

example.org /submit 18

Hello Spartan World!

Server → Client

2 text/plain; charset=utf-8

ok id=123

```
Notes:
- After the `

`, the client sends **exactly 18 bytes** of data. The bytes can be text or binary.
- Servers usually read the request line, then read `content-length` bytes, process, and repl

**C. Upload binary (4 bytes)**
```

Client → Server

files.example /upload/logo.bin 4

PNG (first 4 bytes of a PNG)

Server → Client

2 application/octet-stream

saved as /files/logo.bin

```
**D. Redirect**
```

Client → Server

example.com /old 0

Server → Client

3 /new

```
Client should immediately re-request:
```

example.com /new 0

```
**E. Client error (bad path)**
```

Server → Client

4 File not found

```
**F. Server error**
```

Server → Client

5 Database unavailable

```
**When do you start reading?**
- Server reads the request line **until `CRLF`**, parses it, then reads **exactly** `content
- If fewer bytes arrive, the server waits (or times out). If more arrive, they **belong to a

---

## Slide 3 - Gemini: Secure, Read-Only Simplicity
- **Design:** minimal, privacy-respecting hypertext over TLS.
```

- **Request:** exactly one CRLF-terminated line containing the full URL.

gemini://example.com/page

- **Response:** a status code (two digits) + space + meta line, then CRLF and optional body.

20 text/gemini

Hello Gemini!

- **Status codes:**
- **1× Input:** server requests short (1024 B) line of UTF-8 text.
- **2× Success:** returns content.
- **3× Redirect:** client should follow new URL.
- **4× Temporary failure.**
- **5× Permanent failure.**
- **6× Certificate required or rejected.**

Gemini - End-to-End Examples

A. Normal page fetch

Client → Server

(gemini request line) gemini://gemini.example/hello

Server → Client

20 text/gemini

Hello Gemini

Welcome to the capsule!

B. Input flow (1×)

Client → Server

(gemini request line) gemini://form.example/submit

Server → Client

10 Enter your name:

```
Client then sends **a new request** with the same URL but including the input value as a que
```

Client → Server

gemini://form.example/submit?Alice

Server → Client

20 text/gemini

Thanks, Alice!

```
Notes:
- The server cannot receive arbitrary binary data-only short text in the query part.
- Typical maximum input length  1024 bytes.
- This is Gemini's only built-in form of "upload."

**C. Gemtext structure**
```

Heading level 1

Heading 2

=> gemini://example.org/link Link description

- list item one
- list item two

  preformatted block:

```
Gemtext is plain UTF-8 text: readable as-is, rendered with simple rules-
no HTML, CSS, or JavaScript.

**D. Client certificate flow (optional)**
```

Client connects via TLS and presents certificate

Server checks fingerprint and responds accordingly

60 Certificate required

61 Certificate not authorized

62 Certificate valid — welcome

```
Users generate certificates, e.g.:
```

openssl req -new -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365

```
---

## Slide 4 - Titan: Extending Gemini for Uploads
- **Why Titan?** Gemini is intentionally read-only except for short 1× input (  ~1 KB). Tita
- **Transport:** TLS (same as Gemini). Titan commonly pairs with **client certificates** for
- **Idea:** the client sends a CRLF-terminated URL **including upload metadata** (e.g., size
- **Request skeleton:**
```

titan://host.example/path;size=BYTES;mime=TYPE[;token=XYZ][;filename=name]

“Notes: -size= decimal byte length of the body. -mime= MIME type of the body (e.g.,text/plain,image/png). - Optional params likefilenameortoken‘ are application-defined.

Titan — End-to-End Examples

A. Upload a Gemtext note

```
# Client → Server
(gemini-style single request line)
titan://upl.example/notes/new;size=28;mime=text/gemini


# A short gemtext note

# Server → Client
20 text/gemini


# Upload OK
Saved as /notes/42.gmi
```

B. Upload a PNG (binary)

```
# Client → Server
titan://upl.example/files/add;size=4096;mime=image/png;filename=logo.png


<4096 bytes of PNG data>

# Server → Client
20 text/plain
```

```
ok id=123 filename=logo.png
```

C. Authorization with client cert

```
# Server → Client (if missing/invalid cert)
60 Certificate required


# or
61 Certificate not authorized
```

After presenting an accepted cert, retry the same Titan request.

D. Errors

```
# Server → Client (bad size or unsupported type)
40 Invalid upload size


# or
51 Storage backend error
```

Titan vs. Spartan (at a glance)

- **Where the size lives:**
  - Spartan: size is in the **request line** (`content-length`).
  - Titan: size is a **URL parameter** (`;size=`) and the body follows the CRLF.
- **Security:**
  - Spartan: can run over TCP or TLS; auth is app-specific.
  - Titan: runs over TLS (Gemini model) and often relies on **client certs**.
- **URL shape:**
  - Spartan: `host SP /path SP content-length` then body.
  - Titan: `titan://host/path;size=...;mime=...` then body.

Use cases: microblogging, comments, file lockers, wiki edits, form submissions — anything that needs real uploads beyond Gemini's 1× input.

Slide 5 — Summary

Protocol	Transport	Read/Write	Simplicity	Security	Uploads
Nex	TCP	Read	Extreme	Optional	No
Spartan	TCP or TLS	Read + Write	Very high	Optional	Yes
Gemini	TLS	Read (+ tiny input)	Moderate	Strong	Limited (1KB text)
Titan	TLS	Read + Write	Moderate	Strong	Yes

Takeaway: each protocol builds on simplicity — Nex and Spartan emphasize minimalism; Gemini adds TLS and identity; Titan extends Gemini for interactive, write-capable experiences.