Overview

The **XINA API** (or **XAPI**) provides programmatic access to a XINA server.

> Note that client applications do not connect directly to the server. The XINA Tunnel utility performs the actual server connection, authentication, and security, and provides a local server for local client to connect.

XAPI is built on the **XINA Protocol** (or **XProtocol**), a TCP format used to communicate with the XINA server. It is designed to be simple and easy to implement across many languages and environments, using standard UTF-8 character encoding and JSON data structures.

Tokens

XProtocol is intended to be parsed in-place as a stream is read. This is achieved with variable length **tokens** of the following format:

- prefix length: one byte UTF-8 digit indicating length of the prefix in bytes
- prefix: UTF-8 digit(s) indicating the length of the content in bytes
- content: UTF-8 encoded string or binary data of (prefix count) bytes

For example:

- `14cake` = `"cake"`
- `213big hamburger` = `"big hamburger"`

The maximum allowed length of a single token is 2GB. A token may also be empty, which can be denoted with either the prefix `10` or the shorthand prefix `0`.

> Note that the content length is specified in *bytes*, not *characters*. Because UTF-8 is a variable length encoding format, it is recommend to first convert string data to bytes before creating a token for an accurate count.

Packets

Tokens are combined together into **packets**, which form all communication between the client and server.

Client Packets

Client packets are sent from the client to server. They use the following format:

- packet type :: one byte UTF-8 character
- header token :: JSON object containing header information (used primarily for system purposes, typically empty)
- content token :: UTF-8 encoded JSON object, binary data, or empty depending on token type

Client packets use the following packet types:

Type	Code	Description
`ACTION`	`A`	contains an API action (most common packet type)
`BINARY`	`B`	contains binary data (used for transmitting file data)
`CLOSE`	`X`	closes the connection
`CONTINUE`	`C`	prompts continuing a data stream from the server
`END`	`E`	indicates the end of a series of binary packets
`INIT`	`I`	initializes the connection
`KEEPALIVE`	`K`	ignored by both server and client, keeps connection open
`OBJECT`	`O`	indicates the start of a binary object

Server Packets

Server packets, inversely, are sent from the server to clients. They use the following format:

- packet type :: one byte UTF-8 character
- status code :: three byte UTF-8 numeric status code
- header token :: JSON object containing header information (only used currently for system functions, typically empty)
- status token :: JSON object containing status information
- content token :: UTF-8 encoded JSON object, binary data, or empty depending on token type

Server packets use the following packet types:

Type	Code	Description
`KEEPALIVE`	`K`	ignored by both server and client, keeps connection open
`SERVER`	`S`	primary server packet type, used for all functions

The server packet status token is a JSON object in the following format:

```
{
 "type"   : "OK" or "ER",
 "code"   : <int>,
```

```
  "message" : <string, optional>
}
```

The `"type"` indicates if an action succeeded (`"OK"`) or failed (`"ER"`). The `"code"` is a numeric identifier for the status and will be in the range of 100 to 500.

Code	Description
`1XX`	Success, more data available
`2XX`	Success, data ended
`4XX`	Content error
`5XX`	Server error

The optional `"message"` contains a plain text description of the status or error.

Control Flow

In practice the general design of XProtocol is call and response. Each packet (of most types) sent by a client will receive a single server packet in response. The exception to this rule is the binary object upload procedure, detailed below.

Initialization

When an application opens a connection with the XINA Tunnel, it must first send a single `INIT` packet containing a JSON object:

```
{ "version": "3.0" }
```

Currently the only attribute for this object is the XProtocol version number, which is currently 3.0. More attributes may be added in the future. The XINA Tunnel will then respond with a server packet indicating if the initialization is accepted. If it is not, the connection will then be closed by XINA Tunnel. If it is accepted the application may then begin sending other XAPI packets.

Actions

The bulk of the XAPI communication consists of a collection of discrete **actions**. Actions are fully **transactional**; any changes performed by an action must **all** be successful or **no** changes will be committed.

Each action is encoded as a single JSON object, with the exact format dependent on the action type. There are two categories of actions; data actions, which read or manipulate data, and administrative actions, which alter data structures or perform other administrative tasks.

All actions have a standard server response. If an action returns no data (such as a write action), or if the returned data fits in a single `SERVER` packet, the server will respond with a single `SERVER` packet, indicating success or failure for the action in the status token, and with any returned data in the content token. If a `SERVER` packet contains data, it will always be formatted at a single JSON object.

client		server
`ACTION` token	?	
	?	`SERVER` token (`2XX` status code)

In some cases the server may send the results of a query over multiple packets. This is indicated by a `1XX` status code in a `SERVER` packet. The client may send a `CONTINUE` token to receive the next packet, until a `2XX` code is received, indicating the data has been sent.

client		server
`ACTION` token	?	
	?	`SERVER` token (`1XX` status code)
`CONTINUE` token	?	
	?	`SERVER` token (`1XX` status code)
`CONTINUE` token	?	
	?	`SERVER` token (`2XX` status code)

In this case, the complete data response can be aggregated from the JSON objects according to the following rules:

- properties appearing in a single object are included as-is
- properties appearing in multiple objects are merged based on the data type
  - if the first instance of a property is a JSON array, successive arrays are concatenated and non-arrays are appended
  - otherwise, the values are appended individually into a JSON array

For the purposes of this merge operation, an explicit `null` value should be included in merged results, whereas an explicit or implicit `undefined` should not.

Example

Given the following three server packets:

```
{
 "a": 0,
 "b": 1
}
```

```
{
 "a": undefined,
 "b": [2]
 "c": [4, 5, 6]
}
```

```
{
  "b": null,
  "c": [7, 8, 9]
}
```

The correctly merged result would be:

```
{
  "a": 0,
  "b": [1, [2], null],
  "c": [4, 5, 6, 7, 8, 9]
}
```

Binary Object Upload

The `OBJECT`, `BINARY`, and `END` packet types allow a client to upload binary objects (such as files) to XINA. Binary objects received by the server are assigned a unique ID, which is returned to the client. The client may then use the ID to refer to the cached object in a future action. Cached objects are deleted if not used within 24 hours.

client		server	notes
`OBJECT` token	?		initializes the object
`BINARY` token	?		contains binary data
`...`	?		contains additional binary data as needed
`END` token	?		ends the object
	?	`SERVER` token (`2XX`)	content contains `object_id`

The `SERVER` token content will be a JSON object in the format:

```
{ "object_id": "<id>" }
```

Unlike other packet types, the server will not respond to each client packet, but only once the `END` packet is received. If the client sends any packet other than an `END` or `BINARY` packet after an `OBJECT` or `BINARY` packet any loaded data will be discarded. If an `END` packet is received without any binary data, an `object_id` will not be returned.

Revision #13
Created 9 June 2022 16:13:54 by Nick Dobson
Updated 15 January 2026 20:32:45 by Nick Dobson