POLISH-JAPANESE ACADEMY OF INFORMATION TECHNOLOGY

Information Technology

Computer Networks

Systems and network programming

Filip Tomaszewski
s15403

Automating Management of SourceMod Packages

BEng thesis
Radosław Nielek PhD

Warsaw, January, 2020

POLISH-JAPANESE ACADEMY OF INFORMATION TECHNOLOGY

Informatyka

Sieci Komputerowe

Programowanie systemowe i sieciowe

Filip Tomaszewski
s15403

Automating Management of SourceMod Packages

Praca dyplomowa inżynierska
Dr Inż. Radosław Nielek

Warszawa, Styczeń, 2020

Abstract

A package-management system consists of a client application working in conjunction with remote software repositories. The community of SourceMod, a plugin loader for Source engine's game servers, lacks both of these imperative facets. Having nearly all community-made plugins released in an arbitrary manner, a system must be put in place to abstract and standardize their distribution. It needs to utilize recurring conventions in publishing to be compatible with a matured community with over a decade long development history. The package manager on the client side can then use this system to provide its users the comfort of automatic installation, removal, dependency resolution and more.

Keywords: SourceMod, Package manager, Server-client model, Automation

Streszczenie

System zarządzania pakietami składa się z aplikacji klienckiej współdziałającej ze zdalnymi repozytoriami pakietów. W społeczności SourceMod, modyfikacji zarządzającej rozszerzeniami do serwerów gier w silniku Source, brakuje obu tych niezbędnych aspektów. Posiadając prawie wszystkie wtyczki wypuszczone w arbitralny sposób, specjalny system musi być zbudowany w celu abstrakcji oraz standaryzacji ich dystrybucji. Powinien on wykorzystywać powtarzające się konwencje w publikacji, aby być kompatybilnym z rozwiniętą społecznością z ponad dziesięcioletnią historią pracy. Aplikacja kliencka jest wtedy w stanie użyć tego systemu, żeby zapewnić użytkownikom komfort automatycznej instalacji, usuwania, pobierania zależności, i nie tylko.

Słowa kluczowe: SourceMod, System zarządzania pakietami, Klient-serwer, Automacja

Contents

Chapter 1

Introduction

Proper command line tools play a crucial role in server management. Their text-only approach through CLI provide operators a straightforward and powerful mean to interact with the machine over a diverse set of media. One nature of such tools is automating software installation to the point of entering a single command.

SourceMod Addon Manager (SMAM) aims to provide this missing automation to the community built around SourceMod, a server modification for video games that run on Valve's Source engine. SourceMod is a platform which provides an abstraction layer for its official and community-developed plugins. A plugin itself is just a binary file that is loaded and run by SourceMod. However, complications may begin when it is distributed with other files such as configuration, localization or more. Most of such files must be placed in their individual directories to properly install a SourceMod plugin. More cases may arise with more advanced plugins, sometimes introducing the need to navigate across the entire server file repository. Moreover, there exists no standardized mean of distribution, making the administrator rely entirely on instructions provided by the author, wherever the plugin is published.

By introducing an intermediary system consisting of a client application and a server hosting a database, these problems are mitigated. The database accessible via a web server is effectively standardizing distribution by providing necessary information needed to install a plugin. The client, SMAM, then uses that information to download and install plugins proper. All this is achievable without explicit developer intervention or changing means of distribution, and supports the most important websites hosting required files.

1.1 Chapter outline

The following chapter presents how SourceMod is installed and utilized, and with that, the introduction to the server file structure and its caveats. Many newbie server administrators and plugin developers face common problems in this matter. A few tools come to help in their endeavors, and while successfully providing it, not every problematic aspect is addressed.

Chapter 3 takes a look at current conventions which could be exploited to implement a reliable solution. From those conventions it delves into how they are utilized in the employed system, as well as the overview of that system. More in-depth look is taken at both the client application itself, and the underlying web server it works in conjunction with.

The research chapter analyses a questionnaire which has been sent out to the community of SourceMod server owners and developers. It outlines how each feature of SMAM stands up against practical requirements and which parts still need improvements.

Conclusions will wrap things up, and give an outlook on how the project could reflect in the community. Most importantly, it will plan ahead which actions must be taken to further support SourceMod developers and server owners alike.

Chapter 2

Problem

2.1 Source engine dedicated servers

Valve's game engine, Source, is the engine behind many popular titles such as Half-Life 2, Counter-Strike: Global Offensive or Team Fortress 2 [Val08]. It allows for running headless instances of any of its games, internally called *mods*, through their Source Dedicated Server tool (srcds) on Linux, Windows and macOS. The term *mods* comes from the fact that originally all games started as extensions of the base game – Half-Life 2.

The directory structure is the most relevant aspect of the server and both convention and the engine enforce placement of files in specific directories. In the root of the server the most relevant ones are:

- `mod` – abbreviated name of the mod, stores game specific assets,
- `hl2` – assets from the base game, Half-Life 2,
- `cfg` – configuration presets for the game and server extensions.

Under `mod`, there exists another structure, mostly to group different kinds of assets, like models or textures. The noteworthy directory in there is `addons`. In it, each folder corresponds to different extensions, C++ libraries that modify the behavior of the server [Val09].

Source engine exposes a set of interfaces they may interact with, for instance, capture an event of a player connecting. Unfortunately, these interfaces are the only API an extension may safely use. Interacting with server classes is possible, but not without risks. Afterwards, the developer needs to build the extension along with the Source SDK. Setting up the environment may be challenging as there are separate releases for each of the platforms.

The caveats of extension development are twofold:

1. The API that the Source engine provides is very bare bones and lacking features often required in servers of any kind, such as external database querying. This leaves the developer to implement their own solutions, which may unnecessarily vary from one extension to another.

2. The fact that extensions are natively built C++ libraries forces the developer to write and provide support for platform-dependent code. Using different SDK's across platforms adds great overhead to both development and deployment.

SourceMod seeks to alleviate these problems by wrapping the underlying C++ API with its own, exposed in a simple scripting language called SourcePawn [All07]. Developing a SourceMod plugin allows programmers to extend the functionality of game servers in OS-agnostic way. On top of that, the new API provides them with more tools beyond than just altering server behavior. Developers are granted features such as a MySQL database driver or a parser for Valve-specific configuration files right from the base API.

2.2 SourceMod as platform for extensibility

Installing SourceMod is as simple as extracting its archive into `mod/addons/`, after which `mod/addons/sourcemod/` will look as follows:

- `bin` – SourceMod specific binaries,
- `cfg` – configuration files for plugins or extensions,
- `plugins` – plugin binaries,
- `extensions` – extension binaries,
- `gamedata` – text files storing game's variable memory offsets and function signatures to be shared across plugins for more complex extensibility capabilities,
- `scripting` – plugin sources and the SourcePawn compiler,

- `translations` – text files storing phrases for localization.

A plugin is a binary compiled by the SourcePawn compiler, it is independent of the operating system and handled exclusively by SourceMod. An extension is required for low-level control which goes beyond the scope of plugins, which in practice is very rare. Both plugins and extensions are loaded by SourceMod, but the latter are C++ libraries and thus system-dependent. To give an example, a plugin is a menu pop-up in the game, while an extension is a database driver.

In relation to the aforementioned directory structure and Source engine conventions, SourceMod expects a sort of order. A plugin loads localization only from `translations`, or function signatures from `gamedata`. Ultimately, it is up to the plugin what to do and where to load its files from. For instance, if it comes bundled with assets, they usually are not placed in the SourceMod directory at all, but rather two directories up, alongside server files.

Considering a server is usually on a remote machine, it becomes cumbersome to download and install each file in its appropriate directory through CLI. Operators understandably resolve to downloading the plugin to their computers, then using an FTP client to send the files over. This process is not ideal and graphical approaches are not very flexible nor extensible. Furthermore, it is problematic to automate, given there exists no standard on how to distribute plugins.

When a novice wishes to install a plugin, they must resolve to looking up instructions provided by the author. These are usually found in a readme file that comes with it, or posted on a website where it was published. The most common of such websites are the official forums – AlliedModders [All]. Developers write a post in an appropriate section with general information and installation instructions, and attach plugin files or a packaged archive. By no means this is enforced. The developer is free to omit any of these steps, even publish their work on other sites, like GitHub, if it suits them better. Undoubtedly, this adds yet additional overhead from the server operator perspective.

2.3 Package managers

Package managers are administrative tools which automate installation of software packages. Their goal is to download a set of files constituting a package, place them in correct directories and keep track of them. Along with that automation come three perks, which are missing from the SourceMod community:

- simplifying the installation process to a single command,
- automatically resolving potential dependencies on other packages,
- standardizing the process of distribution.

The latter is usually thanks to a single or multiple public databases associated with the package manager. They provide the metadata on how to deal with packages, whether it would be their installation, removal, building from source or resolving dependencies. This makes most package managers rely on external sources of information, either official or third-party. An important thing to note here is that the databases are available right from the inception of the corresponding package manager.

Implementing exactly the same solution for SourceMod plugins would force all developers to update their mean of distribution, should they want to make use of it. Considering the community has been maturing for over ten years now, this severely undermines the usefulness of a dedicated package manager. The original developer must be in charge of file distribution, but not necessarily the metadata containing installation instructions for these files. Because of this, the database has no need to host any files, only store text.

2.4 LinuxGSM

Short for Linux Game Server Managers, LinuxGSM [Lin18] is a CLI tool and a set of Bash scripts, which take the burden of deploying and managing servers of various games. In case of Source engine servers, it is essentially a wrapper around SteamCMD, an official installer and updater of dedicated game servers on Steam. While being versatile, LinuxGSM provides much simpler usage explicitly tailored to its supported servers. This makes deploying one as simple as running a script for the selected game. On top of that, it brings extra quality of life features, like monitoring, back ups, alerts and a debug mode.

Another noteworthy feature is supporting the installation of mods for some game servers. The user passes the `mods-install` parameter to the script followed by the name of the mod, and its latest version is automatically downloaded and set up. This can be used to install SourceMod, which is the first step when deploying a custom server. After that comes the installation of plugins and extensions for SourceMod itself. However, the nature of plugin distribution is very irregular, requiring a tailored installation script for each of them. This makes it very unreasonable to be utilized in such manner, and a more complete system is required, explicitly developed to deal with those inconsistencies.

2.5 Updater plugin

In October 2011 a plugin for SourceMod called Updater [GT11] was released. It has been written to automatically update any plugin which explicitly provides support for it. The idea is as follows:

1. A developer has their plugin register itself to Updater.
2. During registration, a URL is provided to an online text file listing the most up to date version and all files of the plugin.
3. Updater periodically checks for version differences and downloads the files when necessary.

This file, which holds the information required to update a plugin, is referred to as the "update file." It contains the version string of the most recent release, a few optional notes about the changes, and a list of files which constitute this plugin. There may also be a version string of the last but one release coupled with a list of files which have been altered since then, specified manually by the developer. If that version matches the currently installed one, and there is a newer one available, Updater will download only the specified files. Unnecessarily downloading everything each update is wasteful for all kinds of plugins, especially larger ones, and optimizations are always appreciated.

The regular list of files defines them in a format of a path, starting from the root of either SourceMod installation or the mod. The update file is assumed to be found in that root, and every other file in appropriate subdirectories, effectively replicating the local structure. This puts distribution constrains on the developer, as they are forced to use an online file repository to host their project, should they wish to support Updater. While definitely preferred, within the SourceMod community websites like GitHub, Bitbucket or similar are not the primary means of distribution. Developers are used to doing nothing more than creating a thread on the official forums promoting their plugin, and simply attaching its files as attachments.

Updater understandably cannot be used to install plugins and requires explicit developer support to even update one. It is a great solution for what it was designed for, but still requires a lot of intervention from the user and developer side both. While automatic updates is a step in the right direction, support for the versatile distribution means is very much sought after.

Chapter 3

Solution

3.1 Current publishing conventions

Although there is no set standard on how to publish plugins, conventions have emerged over time. Taking advantage of them is the first step towards standardizing distribution.

There are three websites where either the important ones or the majority of plugins are found:

- **AlliedModders forums** – official forums of SourceMod, usually there exists a corresponding thread, regardless where the files are hosted,
- **GitHub** – preferred by some developers, either to upload binaries directly to repositories or have them posted in the *Releases* section,
- **LimeTech** – third-party website hosting high-traffic SourceMod extensions, owned by one of the major contributors.

3.1.1 AlliedModders forums

The official forums are the main source of advertising for developers, thus a sure way to find plugins. A thread dedicated to a release of one, either hosts the files as its attachments or instructs readers where to download them from. All addons posted on the forums must adhere to the SourceMod license, GNU General Public License, version 3. This license applies to derivative works as well [All08], which means that source must be provided along with distribution.

Because of this the forums have a unique feature available to developers when posting their work. If a file is the source of a plugin (*.sp* extension), it will automatically be compiled by the online compiler and displayed next to the source attachment. While convenient, it is quite limited. Should a plugin make use of libraries outside of the standard, the compilation will fail. In such cases, the developer needs to compile the plugin themselves and upload it as a separate attachment. Each are given and identified by a unique ID, which is utilized for downloading by being present in the URL. Unfortunately, there is no API to parse them.

3.1.2 GitHub

GitHub is a preferable alternative to the forums as a file host by many developers. For that, they mostly use the *Releases* feature of the service but may store binary files along with everything else. In case of the latter, there exists a URL which will always point to the latest version of any file in the repository. A similar URL may exist pointing to a file in the *Releases* section, but only for a specific release. Therefor, if anyone wishes to programmatically fetch a file uploaded in the latest release, the GitHub API must be utilized. It is then possible to iterate all the uploaded assets and read their names or URLs, among other properties.

3.1.3 LimeTech

LimeTech [Ash] is not used by the public but rather a single developer, who also is a major contributor to both SourceMod and SourcePawn. The website is his portfolio but additionally hosts files for many popular extensions, making it worthwhile to consider when finding the way to standardize distribution.

Like the forums and GitHub, it lacks a URL to download the latest publication of a project from. And due to a lack of an API, one must resolve to scraping. All the project releases hosted on the website are archives. They are kept on pages of their own, in a three-column table of operating systems to specific versions. A caveat here may be that a cell is empty, either because a certain release for a specific system is not out yet, or is simply not supported. Thus, a traversal of the table is required to find the next, latest one available.

3.1.4 Other websites

Because of the heuristic nature of the solution for standardizing distribution, support for all websites is impossible. The top three mentioned ones are where all plugins are found, save for a few exceptions. And all of them require a dedicated solution in order to fetch the files.

Having said that, the only information that is needed about a file is its name and download URL. If both are provided as a single string of data, it can be utilized to implement a generic solution agnostic of the website. This generic solution must not rely on any sort of API or a scraping method. It may only download a file from a specific URL and save it under appropriate name and directory. Limited of a solution as it is, it opens up possibility to support a whole range of websites.

A notable example here is BitBucket, an alternative to GitHub. BitBucket has a *Downloads* section, where each file is addressed by name. Because of this, it is trivial to provide download URLs for the latest version of published files that need no specific processing. Unlike attachments in case of the official forums, for example, which are required to be scraped. Likewise, any website with a similar simplistic functionality is integrable with.

3.2 AlliedModders' database

When a developer creates a thread on AlliedModders to promote their plugin, they are asked to fill a short form about it. This form contains fields such as the description, category, or applicable games. Upon submittal, the thread is assigned a unique ID under which the plugin's metadata is saved in the forums' database. They are then displayed along with this plugin ID in a header unique to threads in the *Plugins* section of the forums. Underneath, the author attaches applicable files, or points where to download them from.

For the vast majority of plugins, a thread is guaranteed to exist and the database is a promising source of metadata about plugins. Unfortunately, it may not be used for four main reasons:

1. No API to read from the database is available to third-parties. Developing it would require support from the website maintainers, and although sought after, cannot be relied upon in early stages of the project.

2. Post attachments are not tied to the plugin ID. At best, they may be scraped after finding the associated thread. Not to mention the fact, that the file download URLs need not be present in the post at all.

3. There still is the problem of a non standardized way of installing the plugins. By no means can the author's instructions be utilized for automating the process.

4. Plugin ID may not exist for every installable addon for SourceMod. Though ill-advised and rare, the author is not required to make a thread should they want to publish their work on other websites only. Furthermore, SourceMod extensions have their own section, unrelated completely to that of plugins'. There, no plugin ID or database entry is present. And considering extensions have an analogous installation process and may be dependencies to other addons, they are worth supporting. Lastly, in exceptionally rare situations, an extension may be simplified to become a plugin, and moved to the *Plugins* section. No database triggers will run, and it will be left as an atypical plugin lacking its ID. This has affected the No Thriller Taunt plugin [phe11].

To combat the above issues, a custom database is set up.

3.3 SourceMod Addon Manager database

As per every package manager, there exists a corresponding repository of software packages. For a solution to SourceMod community's distribution problem, such a repository may not host files. What it can store is the data about files which constitute a package, and where they can be downloaded from. The crucial part is defining the format of metadata to fit plugins which are distributed across the three aforementioned websites.

Anyone should be able to describe a plugin to the system, regardless whether it is self-developed or not. One of the most important concepts of the project is not requiring any action from other developers. As such, the maintainer is separated from the author. After providing the plugin specification, the rest should be taken care of automatically. For instance, if the author releases a new version, it should not require any update from the side of the maintainer. The plugin data is parsed by the client application, which should be able to fetch the most recent version, adapting the method to the website it detected.

To properly define a package, the following metadata must be provided, for specific reasons:

1. **Name** – Unique, case-insensitive name of the plugin; for identification, installation and searching.

2. **Author** – Name the original author goes by, which may differ from the maintainer; for searching.

3. **Description** – Short description of the idea behind the plugin; for searching.

4. **Category** – Category name as it appears on the AlliedModders forums; for searching.

5. **Plugin ID** – Plugin ID number as it appears AlliedModders forums' plugin's thread; for possible future integration.

6. **Base URL** – URL associated with the plugin; for installation.

7. **Files** – Paths and filenames defining a plugin; for installation.

8. **Games** – Multiple choice field defining which games the plugin works for, may be all; for searching.

9. **Dependencies** – Names of other plugins which this one is dependent upon; for installation.

3.3.1 Fields utilized for searching

Optional but equally important aspects of the database are its search capabilities. Akin to AlliedModders' database, there exist fields like description, category or games. The category is a single choice list: *Admin Commands*, *Fun Stuff*, *Gameplay*, *General Purpose*, *Server Management*, *Statistical*, *Technical/Development*. Similarly to author, description, or games, it is used purely for searching purposes. The applicable games could be enforced during the installation, but often so it happens that plugin works for more games than specified by the author.

3.3.2 Plugin ID

Plugin ID is as it appears on the forums. It serves no purpose in the early stages of the project, but might be useful should the official integration come in. If so, there will be no need to store the author, description or category, as all three could be fetched from the forums' database. This way, data would not be needlessly duplicated and be required to be entered twice by the person submitting. This feature can only be supplied for plugins, as extensions or rare cases of plugins have no ID.

3.3.3 Base URL

Base URL is the first of the two most important fields of the project. It is by this field, that the website will be detected by the client-side application, after which the appropriate method of parsing will be used. Each plugin entry must have this field carefully selected based on the website by the submitter.

AlliedModders

In case of AlliedModders, this will be the URL to the post containing the files. From there, the client is able to scrape the attachments. It is also possible to provide a link to the entire thread, instead of only the original post, or any webpage from which post attachments can be accessed.

GitHub

As mentioned, in GitHub it is possible to provide a URL to the latest version of the file in a repository. This is what is used in the generic solution used for unknown websites. However, GitHub API usage is required to parse a specific file from the latest release. Should this functionality be needed, a URL to the repository may be entered. After that, it is trivial for the client to find the *Releases* API page and fetch the files published there.

LimeTech

LimeTech in this sense is very simple. Every project submitted there follows a standard of being an archive in a cell of a table. All that is needed is scraping that table to find the latest version. The project-specific page's URL is all the information the client needs to find the files.

Other websites

Base URL field for websites besides the main three is redundant. This is because the field is used only to detect the parsing method, but none may be supported when the URL is unknown. In such cases, the only thing left to fill is the *Files* field appropriately.

3.3.4 Files

Files is the second of the two most important fields of the project. This is the field the client uses to link attachments parsed from the *base URL* to their actual placement and file names, as they appear on the disk. Every file here is defined manually by the addon's maintainer in the following format: `<path to file>;<file identifier>`. The two fields must be separated by a semicolon.

Path to file

This is the property that relates directly to where the file belongs on the disk, it must be unique per server installation. For convenience purposes, it is defined to be relative to SourceMod installation root. From there, are the shortest paths to commonly used directories, for instance, `mod/addons/sourcemod/plugins` needs only to be `./plugins`.

Relative paths are supported, such as "`..`" to go up a directory. This is required because, as previously mentioned, addon files may not be limited to the SourceMod directories. A possible security flaw appears here, as the maintainer can exploit this freedom to move up as many directories as they please. This must be accounted for and limited to the server installation, such as `mod/`. An enforcement is present on the client side, making it impossible to go up more than two directories. Furthermore, it is illegal to enter any non-directory locations, such as symlinks or mounted drives.

File identifier

This property relates to the attachment names retrieved from the *base URL*. If this identifier is just a name of the file, it is cross-referenced with the attachments found. Should it match one, the URL the attachment points to will be used to download and save the file under `{path/to/file}/{identifier}`.

The file identifier can also be a URL. In this case the retrieved attachments are not used, since the URL is already known. It is downloaded and saved in a similar manner, whatever legal location *path to file* points to. This is the reason the explicit semicolon separation is needed, to clearly distinguish what could be either a URL or a simple file name.

There is a third possibility, in case the name of the file is not static. The actual name itself is not prone to change, however, there may be situations when a developer includes a version number. This happens most often when a whole project is archived but may also come up when the plugin is a single binary file. For this purpose, the support for regex-defined file identifiers is included. This will ensure the version number could be dynamic. In such case, it is possible to match against multiple attachments. Only one may be valid, however, assuming their only difference is the version number, it is trivial to compare them to retrieve the latest one.

3.3.5 Dependencies

Dependencies on the forums are for the end user to note the prerequisites, they are listed as plugin IDs linking to appropriate threads. The dependencies field in this project are not specifically for viewing, but are rather used in the installation. In practice, it is quite rare for a plugin to have dependency on another. But the less often it happens, the more surprising it may be when a plugin refuses to load.

This field is simply a string of addon names separated by comma. Just as the names themselves, the dependencies are agnostic of whether they refer to a plugin or an extension. For implementation-specific reasons, dependency addons must be installed prior.

3.3.6 Deployment

Security

The database is set up on a virtual private server provided by *Digital Ocean* [Dig18]. Interaction is achieved through a web server bound to a custom domain name – `smamdb.net`. Communication with that server is end-to-end encrypted with TLS using a certificate issued by *Let's Encrypt* [Enc18]. This warrants three important aspects of fetching packages [IBM19]:

1. **Authenticity** ensures that the web server the client interacts with is indeed owned by a trusted party.
2. **Integrity** takes care of fetching the data fully and correctly, without falling victim to potential tampering during transfer.
3. Despite being less important in this situation, **confidentiality** keeps the connection between the client and the web server private.

Plugins have access to many sensitive parts of the server including database connections or the file system. Because of this, it is important to ensure authenticity of the metadata retrieved. This, of course, does not warrant full safety as the metadata itself could lead to vulnerable endpoints. One reason could be that the developer who is publishing their work fancies distributing it on their own website, which happens to be not secured properly. Another reason could be that the attacker had gained access to publishing or editing plugins, rendering them compromised.

By using the OAuth2 [Val] protocol, Valve provides third-party authentication mechanism against their Steam services. It is by these mechanisms that publishers are recognized and validated, as each of them is expected to have a Steam account. Not everyone is allowed to publish their work on the website by default. The user must be trusted in order to have access, and that trust is granted by the owner. Furthermore, each submission is tied to the submitter, they may edit or delete their submissions but not others'. To reiterate, somebody may publish the plugin for the developer instead, so they are not required to have access to make use of the service.

By the end of the day, addons and the game server itself are meant to be public entities, visible and open source. This is why confidentiality in this case is fully optional, as SourceMod officially provides means for anyone to view the running plugins. However, a few people might find it useful should they decide to host their own database of plugins on their own machine, as explained in later sections.

Interaction API

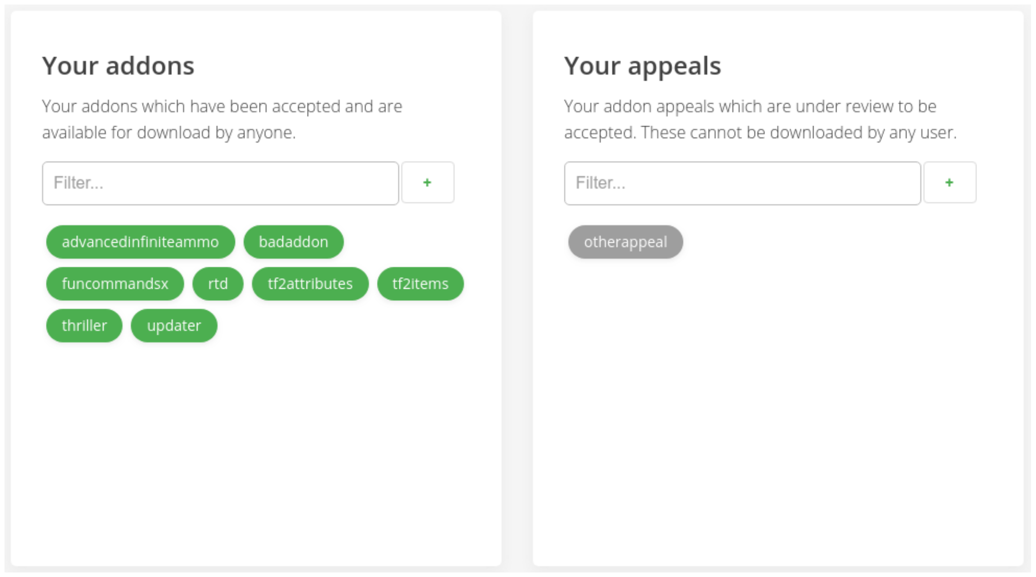
To interact with the database the client sends GET requests to `https://smamdb.net/`. For installation purposes, only the `ids` parameter must be supplied, which is a single name or multiple names of addons separated by a comma. As a response, the server provides a list of JSON objects representing data about each addon requested, with the following fields:

- **id** – name of the addon,
- **author** – to be saved locally for displaying,
- **description** – to be saved locally for displaying,
- **url** – *base URL* which to parse,
- **files** – definition of files to be searched on said URL,
- **deps** – list of dependencies, omitted if there are none.

If there is more than one dependency, the server is forced to recursively retrieve all of them. This makes it possible to return a longer list of addon objects than requested. Furthermore, an action is taken to ensure each addon is provided only once in the list, and not processed for dependencies again.

Management interface

SourceMod Addon Manager Database is a project consisting of more than just a database and the API to call it. This entire endeavour is largely dependent on user support and their accessibility towards it. Under `https://smamdb.net/interface/` a button can be found which directs users to a Steam website in order to validate their authenticity. After logging in through Steam, they will be forwarded to a panel showing their submitted addons, both accepted and pending, shown in Figure 3.1.


Your addons

Your addons which have been accepted and are available for download by anyone.

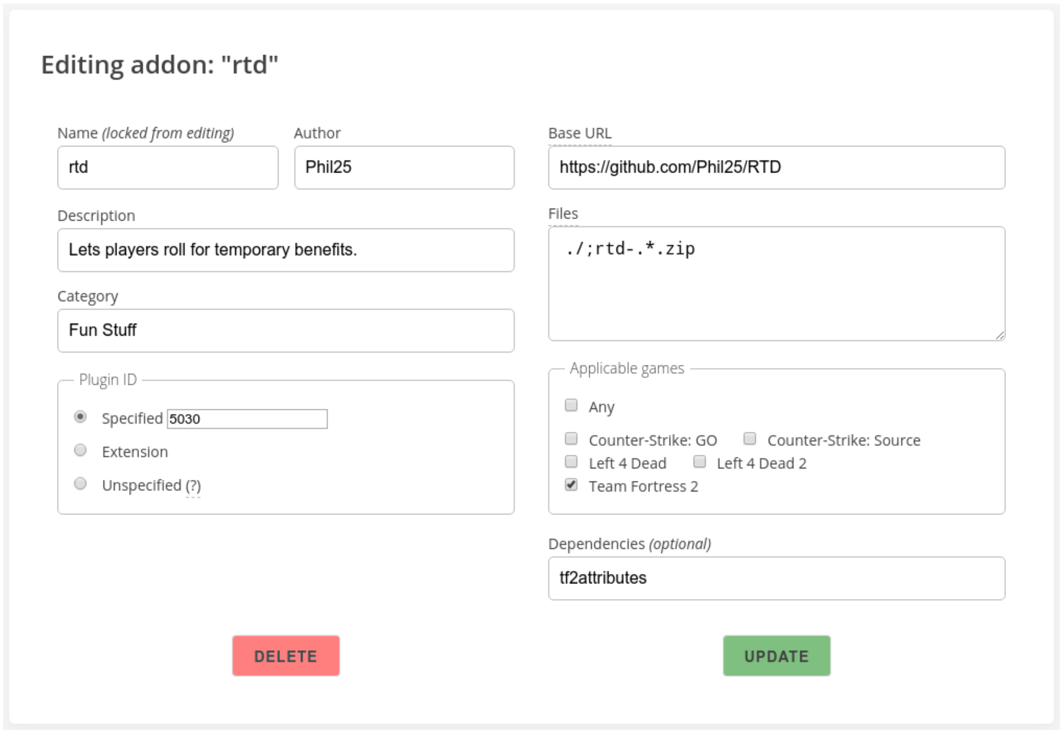
Your appeals

Your addon appeals which are under review to be accepted. These cannot be downloaded by any user.

Figure 3.1: SourceMod Addon Manager Database user interface panel

Pending appeals are ones awaiting a trusted user to confirm their validity. They must be checked for correctness of the properties given but also for the trustworthiness of the submitter. Steam API is a great help for the latter. It can retrieve whether the user's profile is private, how long the account has existed for, or whether it had any incidents in the past, cheating or otherwise. After this step the appeal is moved to the submission section and is available to be retrieved using the Database API. At any moment the submitter is able to edit the metadata of their submissions for correction or potential upkeep requirements.

The *Filter...* field is used to dynamically filter self submitted addons in either section by entering parts of their name. Any of the shown ones may be clicked to open up an editing form, shown in Figure 3.2. The plus button next to it brings up a menu to add a new submission, for untrusted users, this button is available only for the appeals section. Submitting an addon is simply filling a form containing all the fields explained in Section 3.3. This form looks exactly the same as the editing form, save for *submit* button at the bottom, instead of *delete* and *update*.


Editing addon: "rtd"

Name *(locked from editing)*	rtd
Author	Phil25
Base URL	https://github.com/Phil25/RTD
Description	Lets players roll for temporary benefits.
Files	./;rtd-.*.zip
Category	Fun Stuff
Plugin ID	[✓ Specified] 5030 / Extension / Unspecified (?)
Applicable games	Any / Counter-Strike: GO / Counter-Strike: Source / Left 4 Dead / Left 4 Dead 2 / [✓ Team Fortress 2]
Dependencies *(optional)*	tf2attributes

Figure 3.2: SourceMod Addon Manager Database submission editing interface

Self deployment

The database website and the client application are designed to be disjoint entities. The former exposes an API which returns data in JSON, which can be parsed by a multitude of things, due to the popularity of the format. Likewise, calls are simple GET requests, making them trivial to be performed from many bare-bones solutions, like the `curl` program. The burden of implementation is placed on the client application, it is where the complexity of the system is located.

The website is a completely different matter. The custom interface exists for human interaction and is not needed by the client. For this reason, its only required function is the ability to parse a GET parameter and return the requested plugin metadata in a valid format. This makes it possible to write a simple solution for a self hosted web server. As long as it is bound to a hostname, for example `localhost`, the client application can interact with it just like it does with `https://smamdb.net/`.

A self hosted solution may be preferred by communities having their own repository of plugins and wish not to publicize them. Another reason might be a plugin developer using it to quickly deploy their work on a testing server located on a different machine. To achieve this, one may host the web server on their work computer and use SourceMod Addon Manager to retrieve and install the plugin remotely. All that is needed is to point to client application to the right URL. Many things are possible by having this flexibility.

3.4 SourceMod Addon Manager

The bulk of the complexity is contained within the client-side part of the project – SourceMod Addon Manager (*SMAM*). It is a command line tool used to fetch and parse data from database websites, download addon files and keep track of them. SMAM is distributed as a standalone binary natively compiled from modern, idiomatic C++ code, adhering to the latest standard, being C++17 at time of writing. When the time comes, making use of many of the C++20 features is planned.

The target platforms are Linux, Windows and macOS, considering SourceMod itself is available on each. To support this, the application is built with the help of a powerful, cross-platform build system utilizing CMake [Kit] and Conan [JFr] tooling. CMake is a build system generator which creates and configures files for the selected build utility, such as UNIX Makefiles or Visual Studio. Conan, being a praiseworthy supplement for CMake, additionally wraps the whole process in a cross-platform dependency management solution.

3.4.1 Building

The building operations are entirely abstracted by Conan and are split into a two-step process:

1. Firstly, `conan install` command is run, specifying the root directory of the source code. From this directory a file named `conanfile.py` is parsed. In it, the crucial parts of the build are defined, such as its type, the generator, or most notably, the project's dependencies. Conan will ensure that they are available for the same target platform and architecture, as well as compiler and its version, as configured

for the project. The dependencies are downloaded to a remote location and built if the specific pre-compiled binaries are not available, provided they even need to be compiled. The appropriate paths are then given to the selected generator, such as CMake, to aid the compilation process in finding and linking the libraries.

2. The execution of `conan build` creates the generator configuration files in the current working directory. When everything is in place, compilation of the project will be triggered, producing the binary in the designated location. It is also possible to pass the `-c` (configure) flag, to create CMake files only and not trigger the build afterwards.

Additionally, Conan provides cross-building support, i.e., building for a different platform than the one the process happens on. It does that through the help of profiles. A profile is a file containing a pre-configured set of options defining the target and host platforms, or compiler and version, among many others. The name of one can be provided during the setup stage and the rest will be taken care of by Conan.

3.4.2 Tracking installed addons

There can be many game servers set up on a single machine, each having their own SourceMod directory, which in turn have their own set of installed addons. Files of such addons can be split up across different directories over the server repository, which makes them non-trivial to be detected. Because of this, SMAM keeps track of addons it has installed itself, as only then it may know all the relevant files. It does that by storing their metadata in a local file named `.smamdata.json`, in the aforementioned SourceMod's root directory. Locally stored data is not a perfect one-to-one representation of the one stored remotely in the public database (as described in Section 3.3). It is, however, very similar. This allows the local and remote storage to be parsed using the same mechanism with minor adjustments, saving a lot of logic complexity. Namely, the shared fields are *id*, *author*, *description*, *files* and *deps*.

However, one more field is present only locally – *explicit*. It is a boolean property which specifies whether the user requested a particular addon themselves or it was installed as a dependency outside user's instructions. This ensures that during the removal process, all implicitly installed addons which are no longer dependent on are cleaned accordingly.

Storing a file on the disk comes with a few caveats. It must be read and updated every time an operation is executed, which makes it vulnerable to random IO errors. It may also fall victim to user tampering, which could easily invalidate the format. These issues are addressed in the following ways:

1. Placing a period at the beginning of the file in UNIX-based file systems effectively hides it. Making `.smamdata.json` visible only during explicit searches.

2. The storage file is saved in conjunction with `.smamdata.json.bak`, which is its twin copy.

3. In the root of `.smamdata.json` two fields are present – *data* and *hash*. While *data* is the main field from which metadata for installed addons is retrieved, the latter is its hash value. After every saving and loading operation this value is verified. In case a mismatch is found, a recovery using `.smamdata.json.bak` is attempted.

3.4.3 Networking implementation

Any network interaction is achieved with the help of cURL, a versatile, cross-platform C library supporting a wide range of protocols. Each network request is accompanied by a specific user agent, generated each build using the following formula:

```
SourceMod Addon Manager/{major}.{minor}.{patch}-{revision}
```

Where `{major}`, `{minor}` and `{patch}` are consecutive fields adhering to the semantic versioning formatting standard. Additionally, `{revision}` is provided to indicate the commit number. For convenience purposes, this further aids the receiver in being able to pin point down the exact calling code.

Currently, the only networking functionality is downloading JSON data from a database, HTML code from websites needing scraping and files from various sources. For these purposes, the only protocols used are HTTP or HTTPS, as only interaction from publicly available sources is needed. The cURL library opens up possibilities for future implementations of interacting with more private sources, over protocols such as FTP or SSH.

3.4.4 Fetching addon files

Due to vast differences in websites needing to be searched for files, the fetching method must be adjusted accordingly. Additionally, it must conform to a standardized output which the rest of the program can interpret in a similar manner. Its definition is simple:

- **input** – the *base URL* and file identifiers (explained in Section 3.3.4),
- **output** – a map of file names to their respective downloadable URIs.

File identifiers are used for the scraping mechanism exclusively and do not necessarily represent the final names of actual files. And because they need to be known, the file discovery process is composed of two main steps:

1. retrieve every relevant file from the website, then
2. evaluate the identifiers to actual file names.

In the first step, the chosen scraper finds files on the website, these can be attachments to a forum post or a list of assets from the latest GitHub release. It retrieves both the actual file names and where to download them from. However, a plugin is defined by the file identifiers as specified by users in the database, not what can be found on a website by a scraper. As such, step two is cross-referencing those identifiers with the retrieved files, effectively filtering what is unneeded. The identifiers are evaluated to actual file names and the corresponding download URI is assigned to them. This process follows the given formula:

1. Using the exact file identifier, find the file name. If a match of the same name is found, retrieve the URI pointed to by it and finish.
2. Check whether the file identifier is a valid URL by itself. If so, disregard the data scraped from the *base URL* and extract the file name from the back of the identifier then finish.
3. Treat the file identifier as a regex pattern and use it against all files retrieved from the *base URL*. If matches are found, assume they contain a version number as part of their name, then get the most up-to-date one along with its URI and finish. Otherwise, if the regex search returned zero hits, continue to the last step.

4. At this point around 95% of cases have been covered. A few more might be still accounted for after introducing a simple solution to complete the process. Simply treat the file identifier as a valid file name and append it to the *base URL* to find the download link.

To reference the overview of publishing conventions from Section 3.1, three of the major websites are accounted for individually. GitHub is taken care of because of its versatile JSON-based API, which is trivial to parse. As for AlliedModders or LimeTech, scraping of individual pages is needed. Luckily, HTML is syntactically very close to the XML format, making it easily traversable with XPath. Much like regex is used for finding substrings in a short and convenient syntax, XPath is a parsing solution used for finding data in XML documents. For instance, retrieving a list of values from the second column in a table (used for LimeTech) is simplified to a single query: `//td[position()=2]`. AlliedModders is much more tricky in this regard, and many problems must be taken under consideration, outlined under Section 3.1.1.

It might happen that the website is not recognized and no scraper can be utilized. In such instance nothing more can be done than jumping straight to step four of the evaluation process. That is appending the file identifier to *base URL* and hoping for the best. This is not an ideal solution yet adds support for plethora of other websites, most notably BitBucket. The worst that could happen is an error during installation, designed to be trivially reversible as outlined in the upcoming Section 3.4.6.

3.4.5 Usage

Currently, there are three modes of operations (commands) available:

1. `install` – install specified addons, i.e., fetch their metadata and download files into appropriate directories, as well as dependencies,

2. `remove` – remove downloaded files and clean up implicitly installed dependencies,

3. `info` – show information about currently installed addons,

Choosing a mode is simply typing its name after the command, followed by a potential list of addons passed as space-separated arguments: `smam install rtd`. Only one mode is performed at a time.

3.4.6 Install command

Typing `smam install` will trigger the functionality responsible for collecting addon metadata and downloading files. After the `install` keyword addon IDs should be provided as space-separated arguments. This is understandably the most complex of all the commands as it is the crux functionality of the project.

Given that administrators are free to set up as many game servers on a machine as they please, SMAM must know the location of targeted one's SourceMod directory. Luckily, the game server has a specified, easily detectable structure, which allows for simple but efficient heuristics to be implemented. SourceMod will be found provided the application is executed anywhere within the *mod* directory, the root of server files. Users are also able to specify the path themselves using the `-d [--destination]` flag, with which the same method will be used to find the correct folder. All the operations henceforth will be performed relative to SourceMod's root directory.

The installation process consists of five steps:

1. A GET request is performed to the website pointed to by the `--db-url` flag, defaulting to `https://smamdb.net/` if left unspecified. This request is supplied with a variable named `ids`, consisting of comma separated list of addon IDs provided by the user. In accordance with Section 3.3, requested addon metadata is returned in the JSON format. The data is stored by the application to be used later during the installation process.

2. The `.smamdata.json` file is read, providing information on currently installed addons. With it, SMAM is able to recognize whether a requested addon or addons need to undergo the installation process. If they are found already installed, each is marked as *explicit* (explained in Section 3.4.2) and are no longer acted upon.

3. If an addon can be installed, a directory is created in the temporary location defined by the operating system. This step acts as initialization of a transaction, as the process is quite error-prone and meant to be easily revertible should anything go wrong.

4. Using the parsed addon's metadata, dependencies are iterated through, downloaded and installed; then the requested addon follows suit. Files are fetched with the method explained in Section 3.4.4, then placed in

the temporary location. An appropriate directory structure is created along with each file, constructing a replica of the actual server structure. Should installation of any dependency or the addon itself encounter an error, the process ends.

5. Transaction is committed by overlaying the temporary folder with the actual one, overwriting any files. This step's logic is largely provided by the operating system, through the standard library. The installation process moves on to the next addon ID specified by the user, ends if there is none.

3.4.7 Remove command

Correspondingly to installation, `smam remove` cleans the files from the disk and the local cache. Additionally, it takes care of removing every dependency which has been installed alongside the specified plugin. To remove a dependency plugin, the following three conditions must be satisfied:

1. The plugin must have been installed automatically without user's explicit command. Supposing `A` is dependent on `B`, executing

   ```
   smam install A && smam remove A
   ```

   will install then remove `B` as well. However, if `B` has been specified during installation alongside `A` or installed separately, it will not be removed.

2. The plugin must not be required by any other one, i.e., the number of plugins in need of that dependency, excluding itself, must equal one. Appropriate checks are in place to take care of various edge cases. For instance, a situation in which plugin `A` depends on `B` and `B` depends on `A`, removing `A` is not posing a difficulty.

3. The user has not passed the `--no-deps` flag, which forces the application not to delete any dependencies.

3.4.8 Info command

When the user types `smam info`, the application shows a list of installed plugin names in a compact manner. In case the command is followed by a list of plugin names, more detailed information is displayed about each. This command queries only the local database and is used to retrieve all kinds of information about installed plugins, including:

- the name of the plugin,
- its author,
- its description,
- whether it has been installed explicitly or not,
- the list of files it manages, and
- the list of dependencies it requires.

3.4.9 Options

Among commands the user may pass options:

- `-h [--help]` – prints help outlying all the commands and options;
- `-v [--version]` – prints version;
- `-q [--quiet]` – suppresses output;
- `-f [--force]` – forces command execution, different behaviour dependent on the command;
- `--no-deps` – disables automatic dependency installation/removal;
- `--no-prefix` – disables output prefixes indicating severity, such as information, warning or error;
- `--no-color` – disables coloring in output;
- `--allow-running-as-root` – enables running the command with root privileges;
- `-d [--destination]` – specify where to look for the server directory;
- `-db-url` – URL of the database to fetch metadata from, defaults to `https://smamdb.net/`.

3.5 Unsupported features

Package managers are understandably very rich in features. Modern solutions are equipped with version management, PGP package signing and trust system, installation from source or snapshots with rollbacks, among many more. Those features help tremendously in server maintenance and security, however, many of them cannot be utilized in the current state of affairs around SourceMod. The reasons are that those features are either impossible to implement, or the benefits are simply not worth the trade-off for complexity and development time.

3.5.1 Installing specific version

With plethora of software out there, it often may happen that parts of the system are incompatible. When using a modern package manager, those incompatibilities are immediately detected and the maintainer is promptly informed. Version management gives the flexibility of installing a specific version of any package, often being the go-to solution. Despite being used in an edge case situation, it is a very effective feature to have should such problems occur.

This requires the database to store a version controlled repository of package files. Unfortunately, the SourceMod community does not have this convenience for majority of distributed plugins. Each time a developer publishes a new version, the old one is usually lost from the public's perspective.

3.5.2 Update command

Updating is a core aspect of package managers. However, similarly to previously mentioned lack of version control around plugin distribution, detecting new versions is non-trivial. This leaves re-installation of addons to be the best option should anyone want to update.

Luckily, an alternative solution for this already exists in a form of a plugin called *Updater*. Developers are required to provide support for it themselves by providing a text file accessible online which it then parses. The plugin compares version strings specified in it and the one saved locally, and downloads the files marked as needing updating. Additionally, it provides an option to schedule this process. Despite explicit developer intervention not being the idea of SMAM, *Updater* is fully compatible with it, provided the files do not change across versions.

Chapter 4

Research

4.1 Overview

To better investigate the impact on the community, a questionnaire has been sent and filled by various active members of the SourceMod community. It was distributed on the official AlliedModders forums, promoted on the SourceMod Discord server, and sent to relevant people willing to provide their answers. Those answers were collected from 44 of them, where 7 were server owners, 7 were developers and the remaining 30 were both an owner and a developer. Subjects varied in their involvement and time in the community, some being members less than a year, while others where for over 9 years.

A special thanks goes to two moderators of the Discord server and heads of the community: **asherkin**, also the owner of LimeTech, and **headline**. They helped in getting this questionnaire to more people than a single person could.

4.2 Questions

1. **Are you a developer and/or a server owner?**

   SourceMod Addon Manager is a tool aimed at helping those two kinds of users. The end users – players – are not affected by it. It is worth distinguishing which of them is answering the questionnaire.

2. **How long have you been a part of the SourceMod community?**

The community has its years and it is important to roughly establish when the subject had joined. Many of the tools listed in the introduction were not available back in the day, and different users are used to different solutions.

3. **Are you familiar with package managers and ever used one?**

   An understandably popular platform for servers is Linux, where package managers are widespread. Due to this, most developers and server owners are accustomed with the concept of automatically installing software packages. Yet, a small amount of community might be unaware of this, as game server rentals abstract the platform and file management entirely.

4. **Do you own a SourceMod server? (ignore if not)**

   This multi-choice question overviews different means of owning a server. The subject may answer that they either own a dedicated machine, rent a game server only, or self host it. The latter is popular among developers, while the former is more popular among server owners instead. Lastly, renting a game server is cheaper, but much less scalable in the long run. It's this group that SourceMod Addon Manager cannot help yet.

5. **Do you feel the SourceMod community needs a package manager?**

   The community is used to manually install addons by their own means. Answers to this question range from simply "no," to "yes, very much."

6. **Would you use a package manager?**

   To complement the previous question, this one relates to the user themselves. Likewise, answers range from "no," to "yes, very often."

7. **What do you find most problematic when you want to install a plugin?**

   Subjects were able to provide their own answer to this question or skip it altogether.

8. **What is your most important feature in a fully functioning package manager for SourceMod, besides automated installation and removal?**

Subjects were selecting from a predetermined list of answers or were able to provide their own. It is plausible that the SourceMod community is looking for something more specific than a standard package manager.

4.3 Results

Are you a developer and/or a server owner?

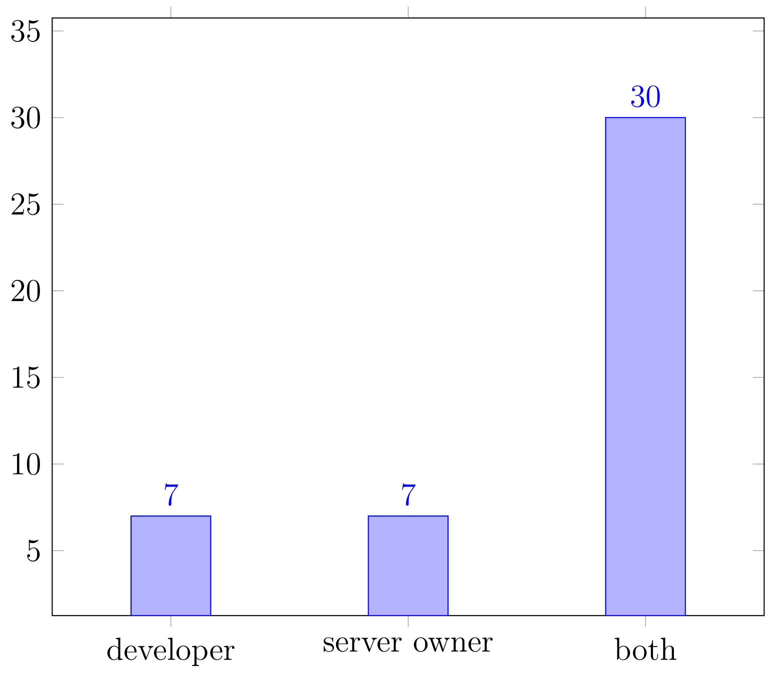


Figure 4.1: Number of developers and server owners

The vast majority of people – over 68% – claim they take on the responsibility for both plugin development and server management. Developers are required to know the basics of hosting and managing a server for the sake of testing their products. In rare cases it may also happen that someone who is not a programmer may need to edit the source code of a plugin, and recompile it themselves. This is mostly due to the fact that plugin are sometimes left abandoned.

How long have you been a part of the SourceMod community?

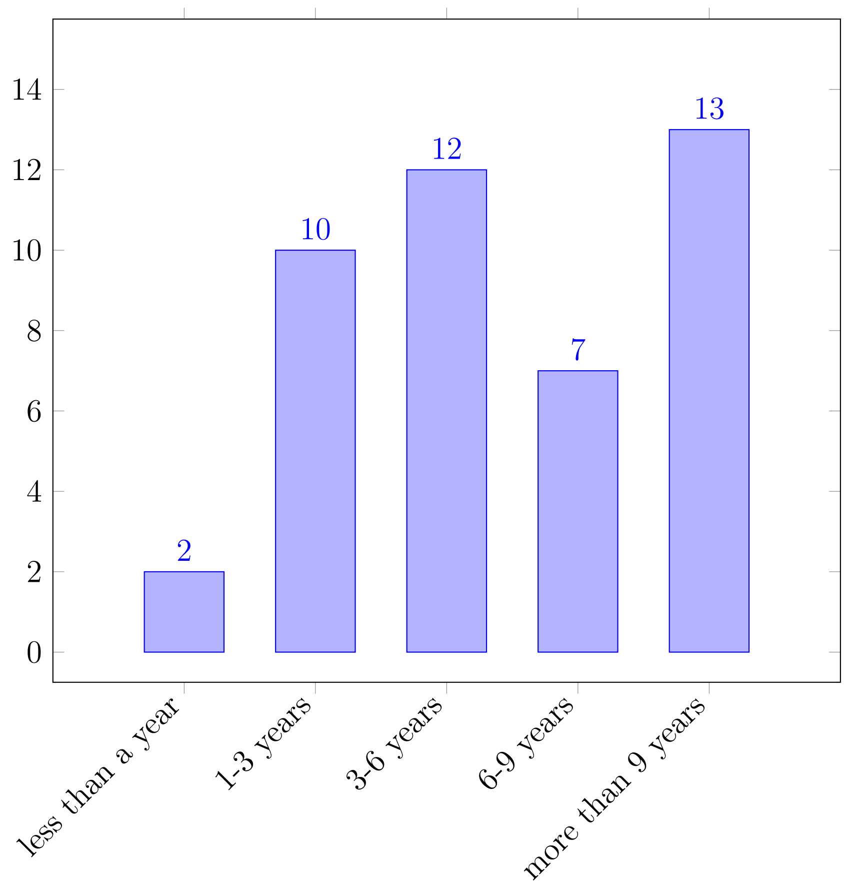


Figure 4.2: Years of community engagement

Almost 30% have been a part of the community for a very long time now. SourceMod itself is the counterpart of AMX Mod X, a similar server mod for GoldSource, the previous iteration of the Source engine, from 1996 [Val05]. People involved in it moved onward with the update to the newer engine and games alike.

Are you familiar with package managers and ever used one?

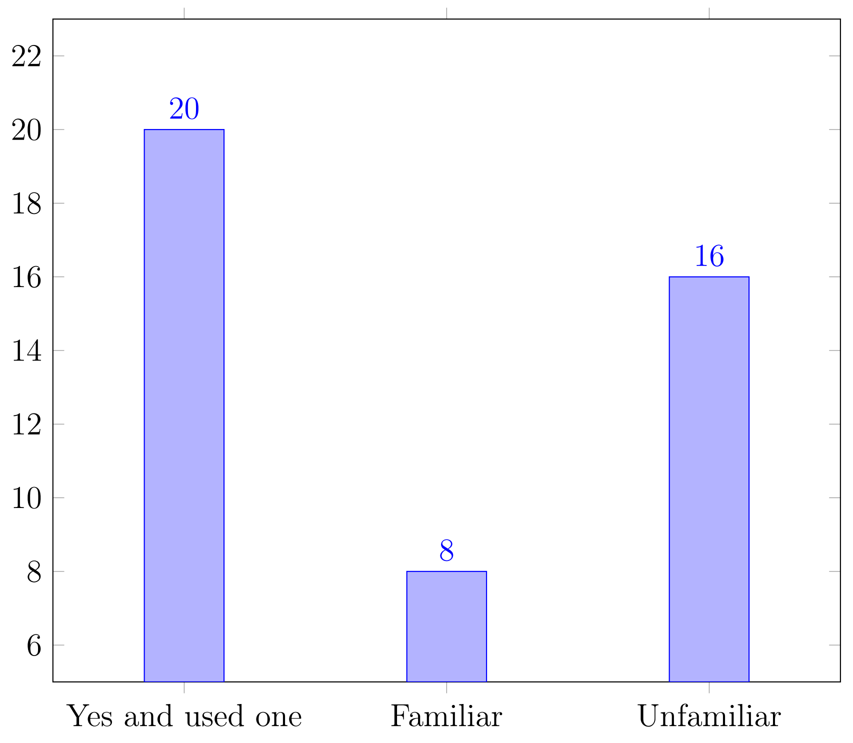


Figure 4.3: Familiarity with package managing solutions

SourceMod supports all major operating systems – Linux, Windows and macOS. Of those systems, macOS is quite rare and Windows is not very popular with package managers. Despite Chocolatey [CS17] being a popular solution for those who wish to use one on that system, it is unnecessary to manage a server on Windows.

Do you own a SourceMod server?

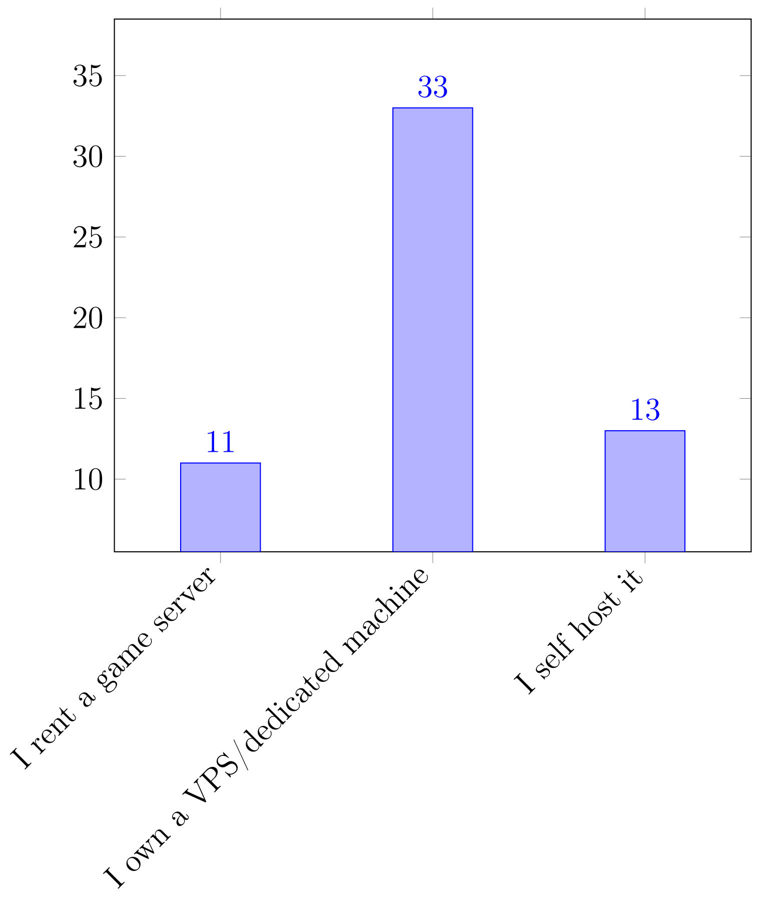


Figure 4.4: Form of the owned SourceMod server

Above are listed the three popular means of owning a SourceMod server. This is a multi-choice question and respondents were able to select any or none. Two out of 44 of them claimed they did not own one.

Eleven people are renting a game server, which is the cheapest solution out of all in case you wish to run a single server and have the system set up for you. Provides of this service sell it by server slots, which define how many concurrent players can be connected. Because access to the underlying system is denied by the provider, SMAM cannot be utilized.

The majority of people – 33 – own a dedicated machine or a virtual private server. The latter being a virtual emulation of the former [Lut10]. This is the most scalable way of managing a game server as the configuration, including available slots, is entirely in the server owners' hands. Self hosting is very similar to this, only with a direct access to the physical machine, which 11 people claimed they have.

Do you feel the SourceMod community needs a package manager?

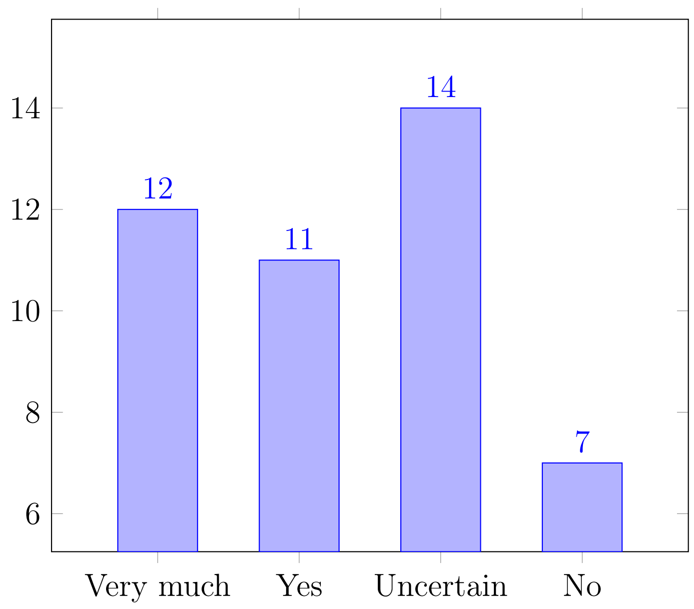


Figure 4.5: Necessity of a package manager for the community

The SourceMod community has matured well without the need of a package manager. Almost 43% of people were unsure if such a solution will be utilized. However, just over a half – 23 persons – responded in a positive way, most of which were very sure of the necessity. Those who answered negatively are in a minority of below 16%.

Would you use a package manager?

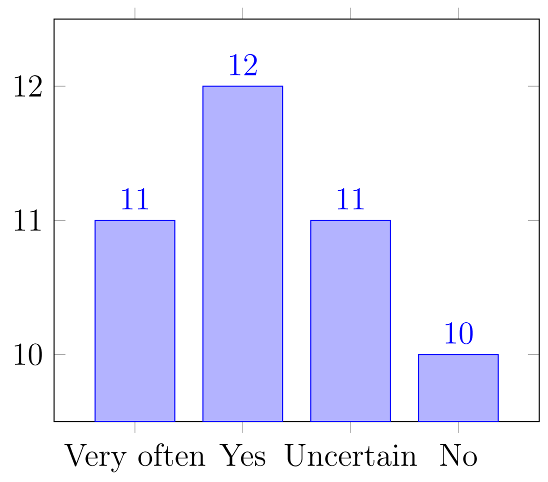


Figure 4.6: Necessity of a package manager for the individual

This time addressing the respondents themselves, answers are comparative. Namely, a bit over half claimed they would use one should it be available. However, as uncertainty shrunk, so did negativity grew, matching at 25% and 23% of answers respectively.

What do you find most problematic when you want to install a plugin?

This is an open question and people could input any answers they wished. The two most recurring topics here were dependency resolution and following installation steps. A slightly lower but still notably ranked were topics like transferring files, manual clean-up on removal and lack of building from source for security reasons. Few claim to have implemented their own solutions to the listed problems.

The standard SourcePawn library allows plugins to do great many things without being dependent on third-party ones. Package dependencies, in general, are not so prominent within SourceMod as much as, say, within a Unix system. Yet people still find this an issue worth taking into consideration. Dependency resolution and automatic installation are the two primary issues that SourceMod Addon Manager solves.

What is your most important feature in a fully functioning package manager for SourceMod, besides automated installation and removal?

Available for selection was a predetermined set of answers as well as possibility to input one's own. Likewise, dependency resolution scored the highest with over 36% of answers, graphical interface followed right after with over 13%. The reminder were split almost evenly, with 3 people wishing for each of the following:

- installation on a remote machine,
- built-in search,
- neatly showing current plugins,
- cross platform (other than Linux),
- server snapshots & migration.

Besides the above mentioned, six respondents input their own answers. The prominent topics in this case are mainly related to support, such as bug fixes and performance improvements.

Chapter 5

Conclusion

Having over a decade old community, SourceMod and SourcePawn both have matured greatly. Due to the open source nature of the entire endeavor, third-party developers have also had their share of contributions. With that, came many improvements to the projects, as well as new plugins and external tools, further widening SourceMod's market share.

SourceMod Addon Manager is an attempt to build upon those community contributions, wrapping them all together in a feature-rich, robust installer. It combines the usefulness of standard package managers with the non-standard ways of distribution, adapting to different cases so the user would not have to. Furthermore, it has been designed to function without requiring intervention from other developers, effectively minimizing its reliance on community support. In compliance with the open source tradition, it is released under the same license as SourceMod itself, GNU General Public License 3.0. Its source code is available on GitHub, under `https://github.com/Phil25/SMAM/` and already is it gaining interest according to traffic statistics.

The official web server – SourceMod Addon Manager Database – follows suit. Available under `https://github.com/Phil25/SMAMDB/`, it shares the license with the client. The web server source code is made to be adaptable. Anyone who wishes to host their private database is free to do so, with any support provided.

5.1 Further extensibility

Current state of the program leaves a fair amount of room for potential extensibility. There are features not yet present which could be very valuable to some users making them worth consideration. A few of them stem from the specific nature of running a SourceMod server and help in generic set-ups found in practice.

5.1.1 Search command

A proper built-in search feature is an important quality of life improvement for a package manager. It allows users to be able to query the remote database for packages straight from the place they use to install one. Executing `smam search` will do just that. From all four commands this is the only one where specifying the server directory could be optional, as no real file interaction is performed. However, specifying the directory has the benefit of the application being able to tell which plugins already are installed.

The list of results must be displayed in a neat and descriptive manner, and must be optimized for consecutive searches. A popular solution for this is caching [Sai13]. The list of packages from available remote repositories is saved locally, then queried for installation or search requests. Furthermore, the remote must be configured to respond in a specific manner to search queries. Namely, it must send the name, description, category and applicable games. In case the search must be more descriptive, it can additionally send the ID of the plugin, and a list of files and dependencies.

5.1.2 Remote file transfer

An important factor but in some cases also one of the biggest cons of SMAM is that it is a terminal application. Managing SourceMod plugins is done per game server basis, not the host machine. This means that in the end a game server is only an application with a file repository around it, which is what is provided by various services out there. Those services charge users per game server, not the machine, and provide access to the files via popular protocols like FTP. End users do not have access to installing software, rendering SMAM ultimately irrelevant.

However, a straightforward solution for this issue may be implementable. When specifying the destination of a command, i.e., the server file repository, a network protocol along with any needed credentials can be provided. This will notify SMAM that the destination is accessed remotely and the specified protocol must be utilized. This feature will enable users to install SMAM on their local machine and issue commands that interact with a remote service. An example of installing a plugin called `rtd` on a server accessible via FTP under "example.com" with the username "user" and password "pass":

```
smam install rtd -d ftp://user:pass@example.com:/home/server/
```

Implementation-wise, directory traversal will remain the same because paths are already known, but every IO operation will be performed over the network. Additional options may be considered, such as utilizing the system's RSA key for SSH transfer, or saving connection credentials in a local configuration file.

Extensions make up a small number of addons, and are specific to the operating system. This could pose difficulties in case the remote one differs from the user's. But because extension files are usually bundled together for each system available, this solution is partially addressed. However, this is not the case on LimeTech. Special precautions need to be taken when installing extensions from there. Either SMAM must somehow detect the remote's operating system or the user must provide a flag explicitly to fetch the appropriate build.

5.1.3 Recreating server state

Snapshots may not be a popular feature of package managers, but the local database of installed software could be a great help for capturing a server state. Moving the local `.smamdata.json` file may become the only thing needed when migrating a game server. Various settings for the installed addons, pose a separate problem, however, those can be set up by nothing more than a text file.

Migrating the entire servers could also be provided by SMAM, when coupled with the remote installation feature. Such example of a command to move a server in the current directory to a remote one could be completely viable:

```
smam migrate --from . --to ftp://example.com:/home/server/
```

5.1.4 Detection of installed addons

Fresh installation of SourceMod comes with a few officially provided plugins and extensions. These addons are not too complex in terms of files and provide the core features only. Suffice to say, they will not be loaded in the local config and thus will not be managed by SMAM. But their simple nature can be a great help for the automatic detection system. Other addons are unfortunately too varied to implement anything reliable enough.

Bibliography

[All] AlliedModders. Sourcemod – AlliedModders forums. `https://forums.alliedmods.net/forumdisplay.php?f=52`.

[All07] AlliedModders. Introduction to sourcepawn. `https://wiki.alliedmods.net/Introduction_to_SourcePawn`, Nov 2007.

[All08] AlliedModders. Sourcemod license. `https://svn.alliedmods.net/viewvc.cgi/trunk/public/licenses/LICENSE.txt?revision=2255&root=sourcemod`, Jun 2008.

[Ash] Asherkin. Limetech builds. `https://builds.limetech.io/`.

[CS17] Inc. Chocolatey Software. Chocolatey documentation. `https://chocolatey.org/docs`, Jan 2017.

[Dig18] Digitalocean. Digital ocean. `https://www.digitalocean.com/`, 2018.

[Enc18] Let's Encrypt. How it works – let's encrypt. `https://letsencrypt.org/how-it-works/`, 2018.

[GT11] GoD-Tony. Updater plugin. `https://forums.alliedmods.net/showthread.php?p=1570806`, Oct 2011.

[IBM19] IBM. How tls provides identification, authentication, confidentiality, and integrity. `https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.sec.doc/q009940_.htm`, Dec 2019.

[JFr] JFrog. Introduction – conan documentation. `https://docs.conan.io/en/latest/introduction.html`.

[Kit] Kitware. Cmake homepage. `https://cmake.org/`.

[Lin18] LinuxGSM. Linux game server managers. `https://linuxgsm.com/`, Jul 2018.

[Lut10] Melissa J Luther. What is vps? virtual private servers explained. `https://hostway.com/blog/virtual-private-servers-explained/`, Mar 2010.

[phe11] pheadxdll. No thriller taunt plugin. `https://forums.alliedmods.net/showthread.php?t=171343`, Nov 2011.

[Sai13] Ravi Saive. 25 useful basic commands of apt-get and apt-cache for package management. `https://www.tecmint.com/useful-basic-commands-of-apt-get-and-apt-cache-for-package-management/`, Mar 2013.

[Val] Valve. Oauth2.0 – steamworks documentation. `https://partner.steamgames.com/doc/webapi_overview/oauth`.

[Val05] Valve. Goldsource – Valve Developer Community. `https://developer.valvesoftware.com/wiki/Goldsource`, Aug 2005.

[Val08] Valve. Source – Valve Developer Community. `https://developer.valvesoftware.com/wiki/Source`, Jan 2008.

[Val09] Valve. Server plugins – Valve Developer Community. `https://developer.valvesoftware.com/wiki/Server_plugins`, Feb 2009.

List of Figures