

sgcFSearch

User Guide

Version 1.0.0

2026-11-03

eSeGeCe, https://www.esegece.com

Copyright © 2026 eSeGeCe

Contents

1 Introduction
- Main features
- System requirements
- The parts

2 Installation
- Downloads
- Install with the setup
- Manual installation from the zip
- Command line and start at login
- Uninstall

3 Getting started
- First run
- Index states
- The hotkey
- A first search

4 The search bar
- Layout
- Result rows
- Keyboard and mouse
- Filter chips
- History
- Folder picker

5 Query syntax
- Words and modifiers
- Typo tolerance
- Path tokens
- Filter reference
- Size values
- Date values
- Sort values
- Type categories
- Worked examples

6 Content search
- What is indexed
- File formats
- Limits and exclusions
- Search modes
- Symbol search
- Combining with name filters
- Automatic fallback

7 Ranking
- Match quality
- The score
- Frecency

8 Tray menu and settings
- Tray icon and menu
- Folders dialog
- Start at login

9 Configuration reference
- Data folder and files
- config.json
- Environment variables

10 Command line (sgcfs)
- Commands and options
- Output and exit codes
- Examples
- JSON and stdio mode

11 Pipe API (JSON lines)
- Connection and security
- Requests
- Errors
- Client sample

12 How it works
- Architecture
- The crawl
- The name index
- Live updates
- The content index

13 Performance

14 Troubleshooting and FAQ

15 Version history

1 Introduction

sgcFSearch is a whole-disk file search tool for Windows. It keeps an index of the names of every file and folder on your fixed drives, answers queries in milliseconds while you type, and keeps the index current while files change. A search bar opens with a global hotkey, the `sgcfs` command line client gives the same results in a terminal, and a local named pipe API lets other programs ask questions too.

Besides file names, sgcFSearch can search the text inside files located in your content folders (your user folder by default), including Office documents and, through Windows filters, PDF files.

sgcFSearch is modelled on scry and inspired by fsearch. It is an independent implementation written in Delphi.

Main features

- Name index of every fixed drive, built by a multi-threaded crawler on the NT directory API and saved to `names.idx` for fast restarts.
- Live updates while the app runs, plus a catch-up through the NTFS change journal (USN journal) for changes made while it was closed. No administrator rights are needed.
- Search bar on a global hotkey with ranked results, file icons, highlighted matches, filter chips, search history, open, show in folder and copy.
- Typo tolerant matching and a rich query syntax: name modifiers, path tokens, extension, type, kind, folder scope, path text, size, modified date, regular expression, limit and sort filters.
- Ranking that learns from the files you open or copy (frecency).
- Content index of the files under your content folders, searched with `grep:`, `regex:` and `sym:`.
- `sgcfs` command line client with plain, long and JSON output, a JSON lines bridge, status, stop, show, install, uninstall and bench commands.
- Named pipe API (one JSON object per line) restricted to the current user.
- Tray icon with the index status, a Folders dialog, Start at login and the daemon log.
- Two downloads, both digitally signed: a per user installer (recommended) and a zip for a manual installation. No administrator rights are needed.

System requirements

Item	Requirement
Operating system	Windows 10 version 1809 (build 17763) or later, or Windows 11, 64-bit (x64).
Drives	Every fixed drive is indexed by default. On NTFS volumes the change journal is used to catch up quickly after a restart. Other file systems are indexed by the crawler and are rescanned instead of using the journal.
Rights	A normal user account. sgcFSearch runs, indexes and reads the change journal without administrator rights.
Memory and disk	They grow with the number of files. See Performance for measurements on a PC with 3.4 million files.

The parts

Part	What it is
`sgcFSearch.exe`	The tray application. It owns the index (it is the "daemon"), shows the search bar on the hotkey and serves the pipe API. Only one instance runs per user.
`sgcfs.exe`	The command line client. It talks to the running app through the pipe and starts it in the background when needed.
Pipe API	`\\.\pipe\sgcFSearch-<your user SID>`, JSON lines, for your own scripts and tools.
Data folder	`%LOCALAPPDATA%\sgcFSearch`: settings, index files, history, usage and the log.

2 Installation

Downloads

sgcFSearch has two downloads. Both are per user and need no administrator rights. The installer and the executables inside the zip are digitally signed by eSeGeCe.

Download	What it is
https://download.esegece.com/sgcFSearch_setup.exe	The installer, recommended. It installs the app and the command line client, adds a Start menu entry and sets up the optional PATH and start at login entries.
https://download.esegece.com/sgcFSearch.zip	A zip for a manual installation. It holds `sgcFSearch.exe`, `sgcfs.exe`, `sgcFSearch-Help.html` and `sgcFSearch-Help.pdf`, all in the root of the zip.

Use one of them, not both. See Uninstall.

Install with the setup

Download and run `sgcFSearch_setup.exe`. The setup is per user: it does not ask for administrator rights and installs only for the current account.

Item	Value
Install folder	`%LOCALAPPDATA%\Programs\sgcFSearch` (for example `C:\Users\you\AppData\Local\Programs\sgcFSearch`)
Files	`sgcFSearch.exe` (the app) and `sgcfs.exe` (the command line client)
Start menu	A shortcut named **sgcFSearch**
Task: Start sgcFSearch at login	On by default. Writes the value `sgcFSearch` = `"<install folder>\sgcFSearch.exe" --background` under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`
Task: Add sgcfs to the user PATH	On by default. Appends the install folder to the user `Path` variable, so `sgcfs` works in any new terminal. When this task is cleared, the setup removes the folder from the user PATH.
Last page	"Start sgcFSearch now" starts the app in the tray (`--background`), without showing the bar.

Both tasks are on the "Additional options" page of the wizard. When you install over an existing version, the setup first asks the running app to stop (`sgcfs stop`), waits until it has saved its index, and then replaces the files, so you can run the setup again to update.

> **Note.** Open a new terminal after the installation. Terminals that were already open do not see the new PATH entry.

Manual installation from the zip

Download sgcFSearch.zip and unzip it to any folder you own, for example `C:\Users\you\Tools\sgcFSearch`. Then run `sgcFSearch.exe`. Nothing is written to the system until you ask for it.

File	What it is
`sgcFSearch.exe`	The app. It runs in the tray and shows the search bar on the hotkey.
`sgcfs.exe`	The command line client.
`sgcFSearch-Help.html`	This user guide, as a web page.
`sgcFSearch-Help.pdf`	This user guide, as a PDF.

Keep `sgcFSearch.exe` and `sgcfs.exe` in the same folder. `sgcfs` starts `sgcFSearch.exe` from its own folder when the app is not running. Settings and index files go to the data folder (`%LOCALAPPDATA%\sgcFSearch` , or the folder in `SGCFSEARCH_HOME` , see Environment variables), not to the program folder.

To update, quit the app from the tray menu, then unzip the new files over the old ones.

Command line and start at login

With the manual installation, two optional commands of `sgcfs` set this up for the current user. The setup does the same through its two tasks.

Command	What it does
`sgcfs install`	Adds the folder of `sgcfs.exe` to the user `Path` variable, so `sgcfs` works in any new terminal. If the folder is already there, it says so and changes nothing.
`sgcfs install --login`	Does the same, and also starts sgcFSearch at login. It writes the value `sgcFSearch` = `"<folder>\sgcFSearch.exe" --background` under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` .

You can also switch Start at login on or off from the tray menu item **Start at login**.

> **Note.** Open a new terminal after `sgcfs install` . Terminals that were already open do not see the new PATH entry. If you move the folder later, run `sgcfs install` again from the new location.

Uninstall

Installed with the setup. Uninstall sgcFSearch from the Windows Settings app, under **Apps**. The uninstaller:

- stops the running app with `sgcfs stop` ,
- removes the program files and the Start menu shortcut,
- removes the install folder from the user PATH,
- removes the `sgcFSearch` login entry from the Run key, also when the tray menu created it.

Installed manually from the zip.

1. Run `sgcfs uninstall` . It removes the login entry and removes the folder of `sgcfs.exe` from the user PATH.
2. Quit sgcFSearch from the tray menu.
3. Delete the folder you unzipped.

In both cases the data folder `%LOCALAPPDATA%\sgcFSearch` (index, settings, history, usage, log) is kept. Optionally delete it by hand to remove every trace.

Note. Do not use both ways side by side. A copy from the zip and a copy from the setup would share the same data folder and the same login entry, and the PATH could point to either one. Uninstall the one you do not want before you use the other.

3 Getting started

First run

Start sgcFSearch from the Start menu entry created by the setup, or run `sgcFSearch.exe` from the folder where you unzipped it. The sgcFSearch icon appears in the notification area of the taskbar. When you start the app yourself, without `--background`, the search bar also opens right away.

On the first run there is no index yet, so sgcFSearch crawls every fixed drive. While it crawls:

- the tray tooltip reads *sgcFSearch: Indexing N entries*,
- the status line of the search bar reads *Indexing... N files*,
- you can already search. Results come from the part of the disk crawled so far and refresh as the index grows.

When the crawl ends, the index is saved to `names.idx` and the state becomes *Ready*. Then the content indexer starts and reads the files of your content folders in the background, at low priority. While it works, the tray tooltip adds *N files being read*.

On later starts, sgcFSearch loads `names.idx`, catches up with the changes made while it was closed, and is ready in about a second.

Index states

State (pipe, `sgcfs status`)	Tray tooltip	Search bar status	Meaning
`starting`	Loading index	Starting...	The engine is starting.
`loading`	Loading index	Loading index...	`names.idx` is being read.
`crawling`	Indexing N entries	Indexing... N files	First run (or no usable index): the drives are being crawled.
`catchup`	Updating, N files, N folders	Updating index...	Changes made while the app was closed are applied. Drives listed in the settings but missing from the index are crawled.
`ready`	Ready, N files, N folders	(nothing)	The index is complete and watched for changes.
`stopping`, `stopped`	Stopping, Stopped	Index stopped	The app is saving its index and closing.

Searches work in every state.

The hotkey

Press `Alt` + `Shift` + `Space` to show the search bar. Press it again, or `Esc`, to hide it. If another program already owns that hotkey, sgcFSearch tries `Win` + `Shift` + `F` and then `Ctrl` + `Alt` + `F`. The hotkey in use is shown next to **Search** in the tray menu and written to the log. You can choose another hotkey in the Folders dialog.

A left click on the tray icon, the tray menu item **Search** and `sgcfs show` also open the bar.

A first search

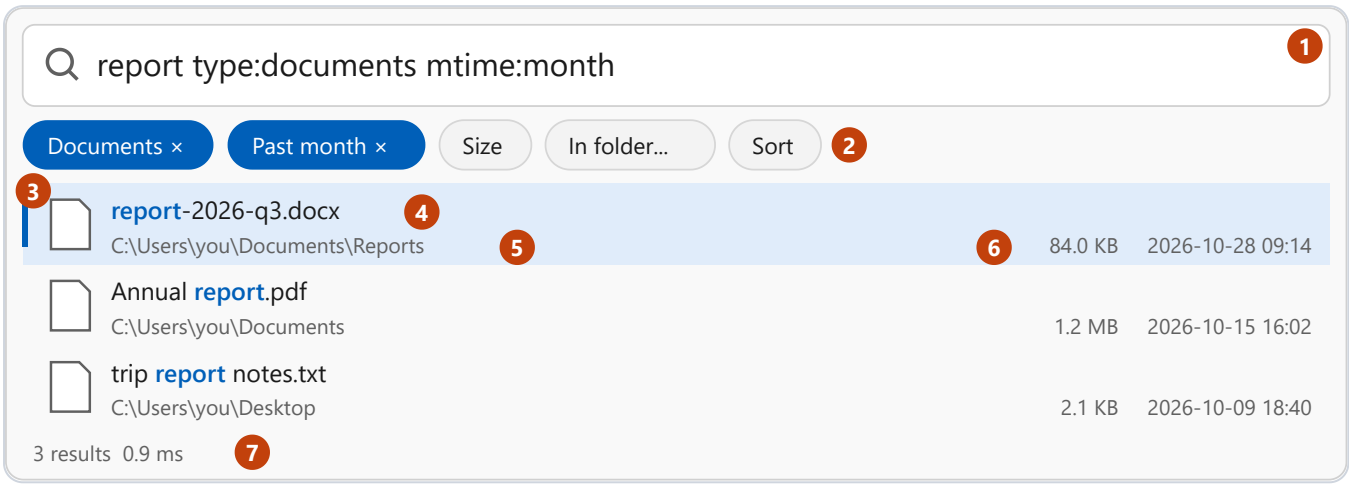
1. Press `Alt` + `Shift` + `Space`.
2. Type part of a file name, for example `invoice`. Results appear as you type, best matches first.
3. Narrow it down: `invoice ext:pdf mtime:month` shows PDF files with "invoice" in the name modified in the last 30 days.
4. Use `Up` and `Down` to select a result and press `Enter` to open it, or `Ctrl` + `Enter` to show it in its folder.

The same search from a terminal:

```
sgcfs invoice ext:pdf mtime:month
```

4 The search bar

Layout



SCHEMATIC Drawn from the bar's form layout (760 by 46 pixel rows at 100% scaling), not a screenshot. File names and values are examples.

#	Part	Description
1	Search box	Type the query here. The placeholder reads "Search files and folders". Results update as you type.
2	Filter chips	Type, Modified, Size, In folder... and Sort. A chip turns blue and shows its value when its filter is in the query. See Filter chips.
3	Result list	Up to 9 rows are visible, scroll for more. The bar loads up to 300 results per query. The selected row has a tinted background and an accent bar on the left.
4	Name and match	The file icon, then the name. The matched part of the name is drawn in the accent color.
5	Folder	The parent folder of the result, shortened in the middle when it is too long.
6	Size and date	File size and last modified date and time, right aligned.
7	Status line	Result count and search time, warnings, errors and the index state. See below.

The bar opens centered on the monitor that holds the mouse pointer, in the upper part of the screen, and stays on top of other windows. It grows with the number of results and hides itself when it loses the focus (for example when you click another window). It follows the Windows light or dark app mode.

Result rows

The bar shows results as two-line rows, not as columns:

Row type	Line 1	Line 2
File or folder name match	Icon and name, matched part highlighted	Parent folder on the left, size and modified date on the right (no size for folders)
Content match	Icon, file name and its folder	`line N:` followed by the matching line, with the match highlighted. Long lines are cut so the match stays visible.

The status line shows, from left to right:

- *No file name matches, showing matches inside files*, when the content fallback produced the results,
- the number of results and the search time, for example *1,234 results 12.5 ms*. For name searches this is the total number of matches, even when only the first 300 are loaded,
- a warning about the query, for example *invalid size: 10q*, or an error in red, for example for an invalid regular expression,
- the index state while it is not ready, for example *Indexing… 1.2M files*.

Keyboard and mouse

Key	Action
`Alt` + `Shift` + `Space`	Show or hide the bar (global hotkey, configurable, with the fallbacks `Win` + `Shift` + `F` and `Ctrl` + `Alt` + `F`)
`Up`, `Down`	Move the selection. When the search box is empty, or shows an entry recalled from the history, they step through the history instead (`Up` older, `Down` newer).
`PgUp`, `PgDn`	Move the selection one page (9 rows)
`Enter`	Open the selected file or folder with its default program
`Ctrl` + `Enter`	Show in folder: opens Explorer with the item selected
`Ctrl` + `C`	Copy the selected file to the clipboard as a file, ready to paste in Explorer. Only when no text is selected in the search box, otherwise it copies the selected text as usual.
`Ctrl` + `Shift` + `C`	Copy the full path of the selected item as text
`Tab`	When the cursor is on an empty `in:` or `path:` token, open the folder picker
`Esc`	Hide the bar

Mouse	Action
Double-click a row	Open the item
Right-click a row	Menu with **Open** (`Enter`), **Show in folder** (`Ctrl` + `Enter`), **Copy file** (`Ctrl` + `C`) and **Copy path** (`Ctrl` + `Shift` + `C`)
Click a chip	Open the chip menu. On an active chip, a click on the **×** at its right end removes the filter.

Open, Show in folder and Copy file feed the frecency ranking. Copy path does not.

Filter chips

The chips are a mouse friendly way to write filters. Each chip reads and writes one filter in the search box, so a filter you type by hand also lights up its chip. Choosing an option replaces the current value of that filter. **Remove filter**, at the bottom of the menu of an active chip, deletes it.

Chip	Menu option	Writes into the query
Type	Documents, Images, Videos, Audio, Code, Archives, Apps, Fonts	`type:documents`, `type:images`, `type:videos`, `type:audio`, `type:code`, `type:archives`, `type:apps`, `type:fonts`
	Folders	`type:folders`
Modified	Today, Yesterday, Past week, Past month, Past year	`mtime:today`, `mtime:yesterday`, `mtime:week`, `mtime:month`, `mtime:year`
Size	Empty, Under 1 MB, 1-100 MB, Over 100 MB, Over 1 GB	`size:empty`, `size:<1mb`, `size:1mb..100mb`, `size:>100mb`, `size:>1gb`
In folder...	Choose folder...	`in:<folder>`, in double quotes when the path has spaces
Sort	Relevance, Name, Size, Date	Relevance removes `sort:`. The others write `sort:name`, `sort:size`, `sort:date`

An active chip shows its value: the Type chip lists the categories (for example *Images, Code*), the In folder chip shows *In* and the folder name, and the Sort chip shows *Sort: Name*.

History

The bar remembers your last 50 searches in `history.json` in the data folder. A search is added when the bar closes with text in the box, and when you copy a file or a path. Repeating a search moves it to the top instead of adding a duplicate.

With an empty search box, press `Up` to recall the most recent search, and keep pressing `Up` for older ones. `Down` goes back toward the newest one and finally to an empty box. As soon as you edit the recalled text, the arrow keys move the selection again.

Folder picker

Two ways lead to the standard Windows folder dialog:

- the **In folder...** chip, menu item **Choose folder...**, which sets the `in:` filter,
- typing `in:` or `path:` and pressing `Tab` while the cursor is on that token. The chosen folder is inserted after the colon, in double quotes when it contains spaces.

5 Query syntax

A query is a list of words and filters separated by spaces. Every word must match the name of the file or folder (not its path), case-insensitive. Filters have the form `key:value`. Only the keys listed in the filter reference are filters: any other `text:text` token is searched as a plain word. The order of words and filters does not matter.

Words and modifiers

Syntax	Meaning	Example
`word`	The name contains the word, case-insensitive. Tolerates one typo for words of 5 letters or more.	`budget`
`w1 w2`	All words must match (AND).	`budget 2026`
`'exact`	The whole name equals the word.	`'readme.md`
`^prefix`	The name starts with the word.	`^setup`
`suffix$`	The name ends with the word.	`.bak$`
`^both$`	Same as `'exact`.	`^desktop.ini$`
`!word`	Exclude names that match. Combines with the other modifiers.	`!^tmp`, `!.log$`
`"two words"`	Double quotes keep spaces inside one word. Modifier characters inside quotes are taken literally.	`"annual report"`

Matching is on the name only. To match folder names along the path, use path tokens, `in:` or `path:`.

Typo tolerance

A plain word of 5 or more letters also matches a name that contains it with one error: one letter missing, one letter extra, one letter different, or two neighbour letters swapped. `confgi` finds `config.json`, `raport` finds `report.pdf`. Typo matches get the lowest match tier, so they normally rank below exact matches. Words shorter than 5 letters, and words with `'`, `^` or `$`, must match exactly.

Path tokens

A word that contains `\` or `/` is a path token. Its last part matches the name, and the earlier parts must match the names of ancestor folders, in that order, as substrings.

Token	Finds
`src\main`	Names containing `main` below a folder whose name contains `src`
`photos\2026\beach`	Names containing `beach` below a `2026` folder below a `photos` folder
`node_modules\`	Anything below a folder whose name contains `node_modules`
`docs/^intro`	Forward slashes work too, and modifiers apply to the last part

Filter reference

Filter keys are case-insensitive. Put the value in double quotes when it contains spaces, for example `in:"C:\My Projects"`.

Filter	Value	Meaning	Examples
`ext:`	Comma separated extensions, the dot is optional	Files with one of these extensions. Files only.	`ext:pas,dfm` , `ext:.pdf`
`type:`	Comma separated categories	Files of one of the categories (OR), or folders with `folders` .	`type:images` , `type:code,documents`
`kind:`	`file` , `files` , `dir` , `dirs` , `folder` , `folders`	Only files or only folders.	`kind:dir`
`in:`	A folder path, `~` for your user folder	Items inside that folder, at any depth. The folder itself is not a result. Repeat the filter to search several folders (OR).	`in:C:\Projects` , `in:~\Downloads`
`path:`	Text	The full path contains the text, case-insensitive (`/` counts as `\`). Repeat it to require several texts (AND).	`path:backup` , `path:\2026\`
`size:`	See size values	File size. Files only, except `size:empty` , which also matches empty folders.	`size:>10mb` , `size:1mb..5mb`
`mtime:`	See date values	Last modified time, in local time.	`mtime:today` , `mtime:<7d`
`re:`	A regular expression	The name matches the expression, case-insensitive.	`re:^img_\d{4}\.jpg$`
`limit:`	A positive number	Maximum number of results. In the search bar, at most 300 are loaded.	`limit:20`
`sort:`	See sort values	Result order. The default is by rank.	`sort:date` , `sort:size-asc`
`grep:`	Text	Content search: lines that contain the text.	`grep:TODO` , `grep:"connection string"`
`regex:`	A regular expression	Content search: lines that match the expression.	`regex:"TODO|FIXME"`
`sym:`	An identifier	Content search: definitions of a symbol in source files.	`sym:TsgcFSearchEngine`

When a filter is given twice, `ext:` and `type:` add to the list, `in:` widens the scope, `path:` , `size:` and `mtime:` must all hold, and for `kind:` , `re:` , `limit:` , `sort:` , `grep:` , `regex:` and `sym:` the last one wins. If a query has more than one content filter, `grep:` is used first, then `regex:` , then `sym:` .

An invalid value never stops the search. It is ignored and reported in the status line, for example *invalid size: 10q*, *unknown type: pics*, *invalid mtime: lastweek* or *empty value: ext:*.

The home folder `~`

In `in:` values, and in the content folders and skip list of the settings, a leading `~` stands for your user folder (`%USERPROFILE%` , for example `C:\Users\you`). Use `~` alone or followed by a backslash: `in:~` , `in:~\Documents` .

Size values

Units are `b` (default), `k` , `kb` , `kib` , `m` , `mb` , `mib` , `g` , `gb` , `gib` , `t` , `tb` , `tib` , all 1024 based. Decimals are allowed: `size:>1.5g` .

Value	Meaning
`size:>10mb`	Larger than 10 MB
`size:>=2g`	2 GB or larger
`size:<1k`	Smaller than 1 KB
`size:<=4kb`	4 KB or smaller
`size:10k` , `size:=10k`	Exactly 10 KB (10,240 bytes)
`size:1mb..5mb`	Between 1 MB and 5 MB, both included. One side may be left out: `size:100mb..`
`size:empty`	Empty files (0 bytes) and empty folders

Date values

Dates use your local time zone. Dates are written `YYYY-MM-DD` . Relative ages use the units `h` (hours), `d` (days, the default), `w` (weeks) and `y` (365 days).

Value	Meaning
`mtime:today`	Modified today, since midnight
`mtime:yesterday`	Modified yesterday
`mtime:week`	The last 7 days, today included
`mtime:month`	The last 30 days, today included
`mtime:year`	The last 365 days, today included
`mtime:<7d`	Newer than 7 days (a bare `mtime:7d` means the same)
`mtime:>30d`	Older than 30 days
`mtime:<12h` , `mtime:>2w` , `mtime:>1y`	Newer than 12 hours, older than 2 weeks, older than a year
`mtime:>2026-01-01`	After that day (from 2026-01-02)
`mtime:>=2026-01-01`	On or after that day
`mtime:<2026-01-01`	Before that day
`mtime:<=2026-01-01`	On or before that day
`mtime:2026-10-03` , `mtime:=2026-10-03`	On that day
`mtime:2026-10-01..2026-10-31`	From the first day to the last day, both included

Sort values

Value	Order
`sort:rank` (also `sort:score`)	Best match first. This is the default. The direction suffixes do not apply.
`sort:name`	Name, A to Z
`sort:size`	Largest first. Folders are sorted by the total size of their contents.
`sort:date` (also `sort:mtime`)	Newest first
`sort:size-asc` , `sort:name-desc`	The suffix `-asc` or `-desc` forces the direction
`sort:-date` , `sort:-name`	A leading `-` reverses the default: oldest first, Z to A

Items with the same name, size or date keep their rank order.

Type categories

`type:` accepts the category names below, and the alternative spellings in brackets. Several categories are combined with commas.

Category	Extensions
`documents` (document, docs)	txt md pdf doc docx odt rtf xls xlsx ods csv ppt pptx odp epub tex pages numbers key xps
`images` (image, pictures)	jpg jpeg png gif bmp tif tiff webp heic heif svg ico psd raw cr2 nef arw dng avif
`videos` (video)	mp4 mkv avi mov wmv flv webm m4v mpg mpeg 3gp mts m2ts vob
`audio` (music)	mp3 wav flac aac ogg m4a wma opus aiff aif mid midi ape
`code` (source)	pas dpr dpk dfm fmx inc c h cpp hpp cc cxx cs java js mjs ts tsx jsx py rb go rs php swift kt kts scala sh bat cmd ps1 psm1 lua pl sql html htm css scss less json xml yaml yml toml ini vb m mm dart r
`archives` (archive)	zip rar 7z tar gz tgz bz2 xz zst cab iso lz lzma jar
`apps` (app, executables)	exe msi msix appx appxbundle com scr lnk
`fonts` (font)	ttf otf woff woff2 fon fnt eot
`folders` (folder, dirs, dir)	Folders, not files

Worked examples

Query	Finds
`report ext:pdf mtime:month`	PDF files with "report" in the name, modified in the last 30 days
`^setup ext:exe in:~\Downloads`	Executables in your Downloads folder whose name starts with "setup"
`'package.json !node_modules\`	Files named exactly package.json that are not inside a node_modules folder
`type:videos size:>1gb sort:size`	Videos over 1 GB, largest first
`kind:dir ^backup`	Folders whose name starts with "backup"
`type:folders size:empty in:C:\Projects`	Empty folders below C:\Projects
`invoice "march 2026"`	Names containing "invoice" and "march 2026"
`type:images mtime:2026-08-01..2026-08-31 path:holiday`	Pictures modified in August 2026 with "holiday" anywhere in the path
`re:^IMG_\d+\.jpe?g$ sort:-date`	Camera pictures, oldest first
`.log$ size:>100mb mtime:>30d`	Large log files not touched for a month
`src\main ext:cpp,h`	C++ files with "main" in the name below a "src" folder
`ext:dproj in:C:\Projects in:D:\Work`	Delphi projects in either folder
`config limit:10 sort:name`	The best 10 matches for "config", shown in name order
`grep:TODO ext:pas in:~\Projects`	Lines with "TODO" in Pascal files of your Projects folder
`sym:ParseQuery`	Where `ParseQuery` is defined in your source files

6 Content search

What is indexed

Content search looks inside files that are located below the **content folders**. The default content folder is `~`, your user folder. Change the list in the Folders dialog or in the `roots` key of config.json.

When the name index is ready, a background thread at low priority reads the files below the content folders and builds a trigram index of their text in `content.idx`. After a restart, only files whose size or modified time changed are read again. While the app runs, changed files are queued and read again automatically.

File formats

Format	Extensions	How the text is read
Plain text and source code	A built-in list of text and source code extensions, for example txt md log csv ini json xml html css js ts pas dfm c cpp h cs java py go rs sql ps1 bat yaml toml svg sln dproj	Read directly. Byte order marks (UTF-8, UTF-16 LE and BE) are honoured, valid UTF-8 without a mark is detected, anything else is read as Windows-1252.
Other extensions	Any extension not in the text or binary lists	The first 4 KB are checked. The file is treated as text when they hold no NUL byte (or start with a UTF-16 mark).
Word, Excel, PowerPoint (OOXML)	docx docm dotx dotm, xlsx xlsm xltx xltm, pptx pptm potx ppsx	Built-in reader. Office does not need to be installed.
OpenDocument	odt ods odp ott ots otp	Built-in reader.
PDF, legacy Office, RTF	pdf doc xls ppt rtf	Through the Windows IFilter registered for the extension, with a budget of 10 seconds per file. Without a registered IFilter the file is not content indexed.
Binary files	exe dll obj zip 7z jpg png mp3 mp4 db iso and other known binary types	Never read.

Limits and exclusions

- **Size cap.** Files larger than `content_max_size_mb` (default 16 MB) are not content indexed.
- **Cloud files.** Online-only files from OneDrive and other cloud providers (offline or recall-on-access placeholders) are never opened, because opening them would download them.
- **Skip list.** Nothing below a folder in the skip list is indexed, neither names nor content.
- **Links and denied folders.** Junctions, symbolic links and mount points are not followed. Folders that cannot be opened (access denied) are left out.
- **Excluded folder names.** Files inside folders with these names are never read for content (their names are still searchable):

```
node_modules  .git  .svn  .hg  .bzr  obj  .vs  .idea  __pycache__  .gradle
bower_components  $Recycle.Bin  AppData  .nuget  .npm  .yarn  .cache  .tox
.venv  venv  .mypy_cache  .pytest_cache  __history  __recovery  .terraform
System Volume Information  .next  .nuxt  .angular  .parcel-cache  .dart_tool
.pub-cache  .m2  .cargo  .rustup
```

Search modes

Mode	Query filter	Pipe `mode`	A line matches when
Literal	`grep:<text>`	`literal`	it contains the text
Regular expression	`regex:<pattern>`	`regex`	it matches the pattern (one line at a time)
Symbol	`sym:<name>`	`sym`	it looks like a definition of the name, see Symbol search
Words	(used by the fallback)	`words`	the file holds every word, and the line holds at least one of them

Smart-case. Content search is case-insensitive, unless the pattern contains an uppercase letter: then it is case-sensitive. `grep:todo` finds "TODO" and "todo", `grep:TODO` finds only "TODO". In `regex:` and `sym:`, escaped sequences such as `\W` do not count as uppercase.

Results. Each hit line is one result, with the file, the line number and the line text. Up to 5 lines are reported per file, and the limit counts lines, not files. Every candidate file is read again at search time to confirm the match, so results never come from stale index data. In the bar, content rows show `line N:` and the text.

> **Tip.** Quote patterns with spaces: `grep:"connection string"`, `regex:"class \w+Helper"`.

Symbol search

`sym:Name` finds lines that define `Name`. It recognises these patterns, which cover Pascal and Delphi, C, C++, C#, Java, JavaScript, TypeScript, Python, Go, Rust, Swift, Kotlin and similar languages:

Pattern	Examples
A definition keyword, then the name, optionally qualified: `function`, `procedure`, `constructor`, `destructor`, `def`, `class`, `fn`, `func`, `interface`, `struct`, `enum`, `trait`, `impl`, `type`, `record`, `module`, `namespace`, `const`, `let`, `var`	`procedure TMyForm.Name(`, `def Name(`, `class Name:`, `fn Name(`, `const Name =`
A Pascal type declaration	`Name = class(TObject)`, `Name = packed record`, `Name = interface`
A Go method	`func (s *Server) Name(`
A C style declaration at the start of a line: a return type, then the name and `(`	`static int Name(`, `public void Name(`, `std::string Name(`

Calls such as `return Name(` or `if Name(` are not reported as definitions.

Combining with name filters

Name filters also apply to content results: `grep:TODO ext:pas in:~\Projects` searches only Pascal files below that folder. Plain words in a query with a content filter must also appear in the file name: `grep:TODO engine` searches for "TODO" only in files whose name contains "engine".

Automatic fallback

When a query has plain words and no content filter, and no file or folder name matches, sgcFSearch searches the content instead: it looks for files below the content folders that contain every word, and reports the lines that contain any of them. Name filters still apply. The bar then shows *No file name matches, showing matches inside*

files, and the pipe response has `"content":true`.

7 Ranking

Match quality

Each word gets a quality tier for each name. With several words, the weakest word decides the tier of the result.

Tier	Quality	Example for the word `report`
5	Exact: the whole name	`report`
4	Prefix: the name starts with the word	`report-2026.pdf`
3	Word start: the word starts after a separator such as `.` `_` `-` or a space, at a letter and digit change, or at a camelCase step	`annual_report.pdf`, `AnnualReport.docx`
2	Substring anywhere in the name	`myreports.txt`
1	Typo match (one error)	`raport.pdf`

The score

With the default `sort:rank`, results are ordered by a score:

```
score = 100 x tier
      - 2 x folder depth (depth counted up to 40)
      + usage bonus (0 to 60, see Frecency)
      + 1 when the first word matches with the exact letter case
```

The tier dominates: one tier is worth 100 points, while the depth penalty is at most 80 and the usage bonus at most 60. Items closer to the drive root rank above deeply nested ones, and files you use often move up. Ties go to the shorter name. The other sort orders use their own field first and the score to break ties.

Frecency

sgcFSearch remembers the files you open and copy, in `usage.json` in the data folder. Each use adds a weight that halves every 14 days:

Action	Weight
Open (`Enter`, double-click, Open menu), Show in folder, a pipe `use` request with `"kind":"open"`	1.0
Copy file (`Ctrl` + `C`), a pipe `use` request with `"kind":"copy"`	0.7

The usage bonus is `10 x ln(1 + sum of the decayed weights)`, capped at 60. A file used often and recently gets a large bonus, one used long ago fades out. Up to 4,000 paths are remembered, the lowest scores are dropped first. Copy path does not count as a use.

See Sort values for the other orders.

8 Tray menu and settings

Tray icon and menu

The tray icon tooltip shows the state, for example *sgcFSearch: Ready, 3,404,349 files, 447,577 folders*, with *, N files being read* added while the content indexer works. A left click opens the search bar. A right click opens the menu:

Item	Action
(status line)	The same text as the tooltip, for information only
Search (default item)	Opens the search bar. The active hotkey is shown next to it.
Folders...	Opens the Folders dialog
Start at login	Check mark: sgcFSearch starts when you sign in. Click to switch it on or off.
Open log	Opens `daemon.log` with the program associated with `.log` files
Quit	Saves the index and exits

Folders dialog

The dialog **sgcFSearch Folders** edits the settings in `config.json`.

Field	Description	config.json key
Content folders (the text of their files is indexed)	Folders whose files are read for content search. **Add...** picks a folder, **Remove** deletes the selected one.	`roots`
Skip folders (never indexed)	Folders the crawler never opens. Nothing below them appears in results.	`skip`
Drives (none checked = all fixed drives)	Check the drives to index. With no drive checked, every fixed drive is indexed.	`drives`
Search bar hotkey (for example Alt+Shift+Space)	One or more modifiers (`Ctrl` , `Alt` , `Shift` , `Win`) and one key (`A` to `Z` , `0` to `9` , `F1` to `F24` , `Space`), joined with `+` . An invalid entry is rejected with a message.	`hotkey`

OK saves the settings and restarts sgcFSearch in the background to apply them. **Cancel** closes the dialog without changes. The content size cap is not in the dialog, edit `content_max_size_mb` in config.json instead.

Start at login

Start at login writes the value `sgcFSearch` = `"<path>\sgcFSearch.exe" --background` under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`. At sign-in the app starts in the tray without showing the bar. Three places control the same entry: the tray menu item, the start at login task of the setup, and `sgcfs install --login` / `sgcfs uninstall` for the manual installation.

9 Configuration reference

Data folder and files

All data lives in `%LOCALAPPDATA%\sgcFSearch` (for example `C:\Users\you\AppData\Local\sgcFSearch`), unless `SGCFSEARCH_HOME` points elsewhere. The folder is created when needed.

File	Content
`config.json`	Settings. Defaults are used when it is missing.
`names.idx`	The name index with the change journal position of each drive. Saved every 5 minutes when something changed, and on exit.
`content.idx`	The trigram content index of the content folders. Saved with `names.idx`.
`usage.json`	Open and copy history for the frecency ranking (up to 4,000 paths)
`history.json`	The last 50 searches of the search bar
`daemon.log`	The log. At 2 MB it is renamed to `daemon.log.old` and a new one starts.

config.json

A file with every key at its default value:

```
{
  "roots": ["~"],
  "skip": [],
  "hotkey": "Alt+Shift+Space",
  "drives": [],
  "content_max_size_mb": 16
}
```

Key	Type	Default	Meaning
`roots`	array of strings	`["~"]`	Content folders, whose files are indexed for `grep:`, `regex:` and `sym:`. `~` is your user folder.
`skip`	array of strings	`[]`	Folders the crawler never opens. `~` is allowed.
`hotkey`	string	`"Alt+Shift+Space"`	Global hotkey of the search bar: modifiers `Ctrl` (or `Control`), `Alt`, `Shift`, `Win` (or `Windows`) plus a letter, a digit, `F1` to `F24` or `Space`. Case and spaces are ignored.
`drives`	array of strings	`[]`	Drives to index, for example `["C:\\", "D:\\"]`. Empty means every fixed drive.
`content_max_size_mb`	number	`16`	Files larger than this many MB are not content indexed.

Note. Backslashes must be doubled in JSON: `"C:\\Projects"`. The app reads `config.json` when it starts. After editing the file by hand, restart the app: `sgcfs stop`, then start sgcFSearch again. The Folders dialog restarts it for you.

Environment variables

Variable	Effect
`SGCFSEARCH_HOME`	Replaces the data folder. Used by the tests, and useful to keep a separate index, for example on another disk.
`SGCFSEARCH_PIPE`	Replaces the pipe name. A value that does not start with `\\` gets the prefix `\\.\pipe\`. The app and every client must see the same value.

10 Command line (sgcfs)

`sgcfs` talks to the running app through the pipe. A search, `show`, `stdio` and `bench` start `sgcFSearch.exe --background` from the folder of `sgcfs.exe` when the app is not running, print *sgcfs: starting sgcFSearch...* and wait up to 15 seconds for it to answer. `status` and `stop` never start it.

Commands and options

Command	Description
`sgcfs <words...> [-n <limit>] [-l|--long] [--json]`	Search. The words form one query with the full query syntax.
`sgcfs -- <words...>`	Search for words that look like commands or options, for example `sgcfs -- status`
`sgcfs stdio`	JSON lines bridge: each line read from stdin is sent to the pipe, each response is written to stdout
`sgcfs status`	State, version, process id, file and folder counts, content counters, indexed roots and the log file
`sgcfs stop`	Stop the app. It saves its index first. Waits up to 5 seconds and prints *sgcFSearch stopped*, or *sgcFSearch is stopping (saving the index)* if it is still busy.
`sgcfs show [text]`	Open the search bar, optionally with the text already typed
`sgcfs install [--login]`	Add the folder of `sgcfs.exe` to the user PATH. `--login` also makes sgcFSearch start at login.
`sgcfs uninstall`	Remove the login entry and the PATH entry
`sgcfs bench [n]`	Run a fixed set of five queries n times (default 20) over one connection and print the median server time, round trip time and result count of each. It loads the machine while it runs.
`sgcfs help`	Print the usage. `-h`, `--help`, `/?` and running `sgcfs` without arguments do the same. Help never starts the app.

Option	Description
`-n <limit>`	Maximum number of results. Default 100. Must be a positive number. A `limit:` filter in the query wins.
`-l`, `--long`	Long format: size (or `<DIR>`), modified time in local time (`yyyy-mm-dd hh:nn`) and path
`--json`	Print the raw JSON response line instead of the paths

A query given as one argument, such as `sgcfs "notepad ext:exe"`, is used exactly as typed, while with several arguments an argument that contains spaces stays one phrase, as in `sgcfs "annual report" in:~\Documents`. `sgcfs show` follows the same rule.

Output and exit codes

- Results go to stdout, one path per line. Folders end with `\`. Content hits are printed as `path:line: text`.
- The summary goes to stderr: *N results (x ms)*, or *N results, M shown (x ms)* when the limit cut the list. Messages and errors also go to stderr, so piping stdout gives clean paths.

Exit code	Meaning
0	Results found, or the command succeeded
1	No results
2	Error, invalid arguments, or the app is not running and could not be started

Examples

```
sgcfs report ext:pdf mtime:month
sgcfs -l -n 20 type:videos size:>1gb sort:size
sgcfs "annual report" in:~\Documents
sgcfs grep:TODO ext:pas in:~\Projects
sgcfs sym:ParseQuery -n 5
sgcfs show ext:log mtime:today
sgcfs status
sgcfs stop
```

Use the output in other commands, for example in PowerShell:

```
sgcfs ext:tmp in:~\AppData\Local\Temp mtime:>30d | Measure-Object
sgcfs 'package.json !node_modules\ | ForEach-Object { Split-Path $_ }
```

`sgcfs status` prints one value per line (example values):

```
state            ready
version          1.0.0
pid              12345
files            3,404,349
folders          447,577
crawled          3,851,926
content pending  0
content indexed  36,000
roots            C:\, D:\
log              C:\Users\you\AppData\Local\sgcFSearch\daemon.log
```

`sgcfs bench` prints a table with the columns *query*, *server ms*, *round trip* and *results* for the queries *one word* (`config`), *typo* (`confgi`), *two words* (`system config`), *filtered* (`config ext:json,xml`) and *content grep* (`TODO` in pas, txt and md files).

JSON and stdio mode

`--json` prints the response of a search exactly as the pipe returns it (see Requests). The summary still goes to stderr.

```
sgcfs --json -n 2 report ext:pdf
```

```
{"ok":true,"total":57,"ms":3.1,"content":false,"results":
[{"path":"C:\\Users\\you\\Documents\\report.pdf","name":"report.pdf","dir":false,"size":48211,"mtime":"2
026-10-28T08:14:00Z","score":503,"match":[0,6]}, ...]}
```

`sgcfs stdio` keeps one pipe connection open, sends every non-empty line of stdin as a request and writes every response line to stdout. It is the easiest way to use the pipe API from a script that cannot open named pipes:

```
echo {"id":1,"status":true} | sgcfs stdio
```

11 Pipe API (JSON lines)

Connection and security

Item	Value
Pipe name	`\\.\pipe\sgcFSearch-<SID>`, where `<SID>` is the security identifier of the current user, for example `S-1-5-21-...-1001`. `SGCFSEARCH_PIPE` overrides it.
Access	The pipe has a protected access list that grants access to the current user only. Remote clients are rejected, so only programs of the same user on the same PC can connect.
Framing	Byte mode, UTF-8. One JSON object per line, ended by `\n` (`\r\n` is accepted). A line may be up to 16 MB.
Sessions	A connection may send many requests. Each request gets exactly one response line, in order.
Request id	Every request may carry an `"id"` (any JSON value). The response echoes it.

Requests

Search

`q` is a query with the full syntax. `limit` is optional, default 100. A `limit:` filter inside the query wins. The content fallback applies too.

```
{"id":1,"q":"report ext:pdf","limit":50}
```

```
{"id":1,"ok":true,"total":1234,"ms":0.8,"content":false,"results":[
 {"path":"C:\\Docs\\report.pdf","name":"report.pdf","dir":false,"size":12345,
  "mtime":"2026-11-03T10:15:00Z","score":512.0,"match":[0,6]}]}
```

Field	Meaning
`total`	Name results: the number of matches, also when `limit` returns fewer rows. Content results: the number of hit lines found, up to the limit.
`ms`	Search time in the app, in milliseconds
`content`	`true` when the results are content hits
`path`, `name`, `dir`	Full path, name, and `true` for folders
`size`	Size in bytes. For folders only with `sort:size` (the folder total), otherwise left out.
`mtime`	Last modified time, ISO 8601 in UTC
`score`	Ranking score (name results only)
`match`	`[position, length]` of the match in `name`, 0 based. Left out when there is none.
`line`, `text`, `tmatch`	Content results only: line number, line text and `[position, length]` of the match in `text`, 0 based. One result per hit line.

Grep

A content search with the pattern apart from the query. `q` provides the name filters. `mode` is `literal` (default), `words`, `regex` or `sym`. The response has the search shape with `"content":true`.

```
{"id":2,"grep":"TODO","mode":"literal","q":"ext:pas in:~","limit":50}
```

```
{"id":2,"ok":true,"total":3,"ms":4.2,"content":true,"results":[
 {"path":"C:\\Users\\you\\Projects\\Main.pas","name":"Main.pas","dir":false,"size":8120,
  "mtime":"2026-10-30T15:02:11Z","line":42,"text":"  // TODO: check the result","tmatch":[5,4]}]}
```

Status

```
{"id":3,"status":true}
```

```
{"id":3,"ok":true,"state":"ready","files":5080000,"folders":610000,"crawled":123456,
 "content_pending":42,"content_indexed":36000,"pid":1234,"version":"1.0.0",
 "roots":["C:\\","D:\\"],"log":"C:\\Users\\you\\AppData\\Local\\sgcFSearch\\daemon.log"}
```

`state` is one of `starting`, `loading`, `crawling`, `catchup`, `ready`, `stopping`, `stopped` (see Index states). `roots` lists the indexed drives.

Use

Reports that the user opened or copied a file, to feed the frecency ranking. `kind` is `open` or `copy`.

```
{"id":4,"use":"C:\\Docs\\report.pdf","kind":"open"}
```

```
{"id":4,"ok":true}
```

Show

Opens the search bar, optionally with text.

```
{"show":true,"q":"optional text"}
```

```
{"ok":true}
```

Stop

The app replies, then saves its index and exits.

```
{"stop":true}
```

```
{"ok":true}
```

Errors

Errors come back as one line with `"ok":false` and a message:

```
{"id":5,"ok":false,"error":"text"}
```

A request with no known operation gets `{"ok":false,"error":"unknown request"}`. An invalid regular expression in `re:` is reported as a warning and ignored, while an invalid content pattern returns an error.

Client sample

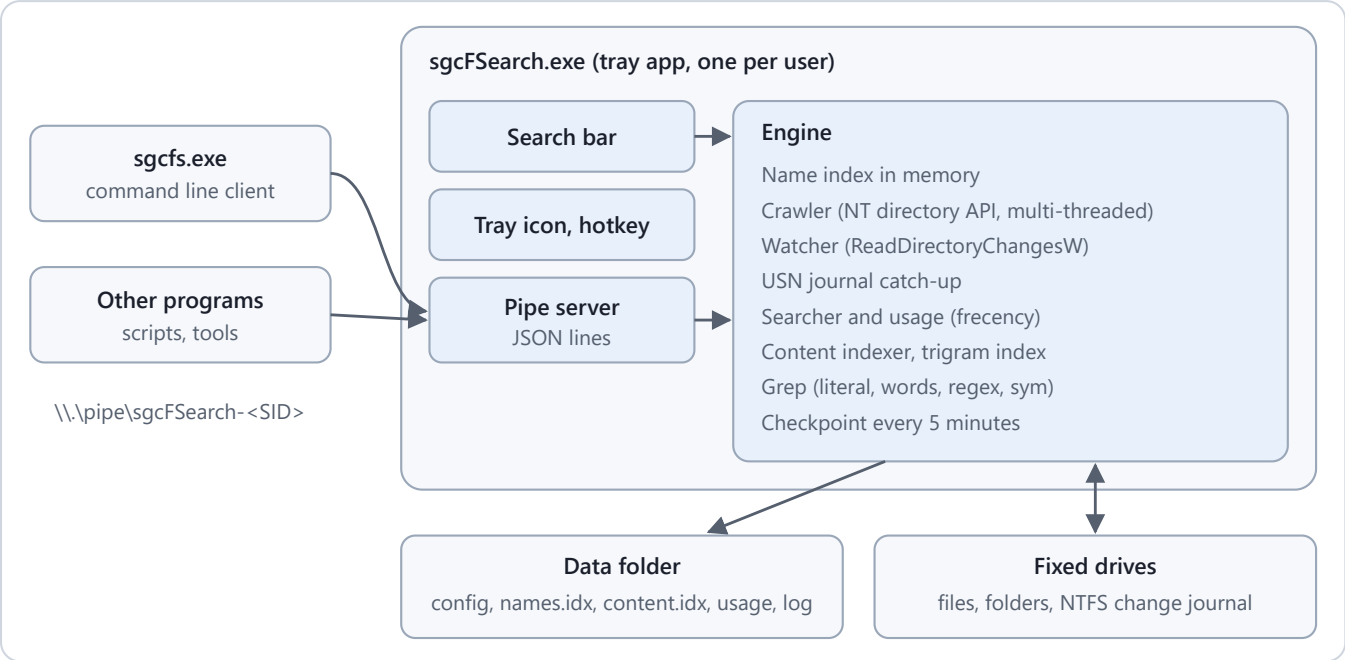
An illustrative PowerShell client that sends one search and prints the paths. It is a sample, not part of the product:

```
# Illustrative sample: one search over the sgcFSearch pipe
$sid  = [Security.Principal.WindowsIdentity]::GetCurrent().User.Value
$pipe = New-Object IO.Pipes.NamedPipeClientStream('.', "sgcFSearch-$sid",
          [IO.Pipes.PipeDirection]::InOut)
$pipe.Connect(5000)
$utf8 = New-Object Text.UTF8Encoding($false)
$w = New-Object IO.StreamWriter($pipe, $utf8)
$w.NewLine = "`n"
$w.AutoFlush = $true
$r = New-Object IO.StreamReader($pipe, $utf8)
$w.WriteLine('{"id":1,"q":"report ext:pdf","limit":10}')
$resp = $r.ReadLine() | ConvertFrom-Json
if ($resp.ok) { $resp.results | ForEach-Object { $_.path } } else { $resp.error }
$pipe.Dispose()
```

The same connection can send more requests before it is closed. Without a named pipe library, run `sgcfs stdio` as a child process and exchange lines through its stdin and stdout.

12 How it works

Architecture



The app owns the engine. The search bar calls it directly, every other client goes through the pipe.

The crawl

The crawler lists folders with the native NT API: each folder is opened relative to its parent handle (`NtOpenFile`) and listed in bulk, many entries per call, instead of one `FindNextFile` call per file. Several threads crawl in parallel. Junctions, symbolic links and mount points are recorded but not entered, which avoids loops and duplicates. Folders that cannot be opened are marked as denied, folders in the skip list are never opened, and cloud placeholder folders are crawled and marked as offline.

Before a drive is crawled, its change journal position is stored. After the crawl, the changes made during the crawl are applied from the journal, so nothing is missed.

The name index

The index lives in memory: one compact record per file or folder (parent, first child, next sibling, name, size, modified time, flags), with the names stored once in a shared pool. Searches scan it on several threads and keep only the best results, which is why a query over millions of names takes milliseconds. The index is saved to `names.idx` together with the journal position of each drive, and loaded in about a second at the next start.

Live updates

- **While running.** A watcher per drive uses `ReadDirectoryChangesW` on the whole drive. Changed folders are listed again and the differences applied. Renamed and moved folders keep their contents. A change is visible in searches within a fraction of a second.
- **Buffer overflow.** When too many changes arrive at once, the watcher catches up from the change journal, or rescans when that is not possible.

- **After a restart.** The NTFS change journal is read from the saved position, through the unprivileged journal interface, so no administrator rights are needed. Only the folders that changed are listed again. A drive without a usable journal is rescanned.
- **Checkpoints.** Every 5 minutes, when the index changed, the journal positions are refreshed and `names.idx` and `content.idx` are saved. They are saved again on exit. An index that was interrupted during a crawl or catch-up is never saved, so a later start does not trust a partial index.

The content index

The content indexer walks the content folders through the name index, so it does not crawl the disk again. It extracts the text of each file (see File formats) and records every trigram, a sequence of three case-folded characters, with the list of files that contain it. A content query first takes the trigrams of the pattern (for a regular expression, of its required literal parts) to find candidate files, applies the name filters, then reads each remaining candidate again on a few worker threads to find the exact lines. Files that turn out to be changed or deleted are queued for reindexing.

13 Performance

QA measurements on one PC, with a Release build. They are not a guarantee: results depend on the disk, the CPU and the number of files.

Measure	Result
First crawl of C:	63 s, for 3,404,349 files and 447,577 folders
Memory	306 MB private bytes
`names.idx` size	202 MB
One word	15.8 ms
Typo	13.8 ms
Two words	17.6 ms
Filtered	13.1 ms
Grep	0.16 ms
Live update visible	308 ms
Restart to ready	1.0 s, journal catch-up 109 ms

The query times are server times as reported by `sgcfs bench` (median of the runs). The five bench queries are described in Examples. Run `sgcfs bench` to measure your own PC.

14 Troubleshooting and FAQ

The hotkey does not open the bar

Another program may own `Alt` + `Shift` + `Space`. sgcFSearch then falls back to `Win` + `Shift` + `F`, then to `Ctrl` + `Alt` + `F`. To see which one is active, open the tray menu (the hotkey is shown next to **Search**) or the log:

```
app: hotkey Alt+Shift+Space not available, error 1409
app: hotkey Win+Shift+F active
```

Two other messages can appear: `app: invalid hotkey "..."` when the setting cannot be parsed, and `app: no global hotkey available` when none of the three could be registered.

Choose a free combination in the Folders dialog. Without any hotkey, open the bar from the tray icon or with `sgcfs show`.

Where is the log?

`%LOCALAPPDATA%\sgcFSearch\daemon.log`. The tray menu item **Open log** opens it and `sgcfs status` prints its path. It holds starts and stops, the hotkey, watcher and journal events, and errors. At 2 MB it moves to `daemon.log.old`.

How do I rebuild the index?

1. Quit sgcFSearch from the tray menu, or run `sgcfs stop`.
2. Delete `names.idx` from the data folder to rebuild the name index. Delete `content.idx` to rebuild the content index.
3. Start sgcFSearch again. It crawls the drives as on the first run.

What happens when I start sgcFSearch twice?

Only one instance runs per user. A second start does not build another index: it asks the running instance to show its search bar and exits. Started with `--background`, the second instance exits silently.

A file I know exists is not found

- Check the state. During the first crawl the index is still partial.
- Check the skip list and the Drives setting in the Folders dialog. Only fixed drives are indexed, network shares and removable drives are not part of the default set.
- Folders your account cannot open (access denied) are listed, but their contents are not.
- Junctions, symbolic links and mount points are not followed. Search for the file at its real location.
- Words match the name, not the path. Use a path token, `in:` or `path:` for folder names.
- `ext:`, `type:` (except folders) and `size:` (except `size:empty`) match files only. `in:` matches the contents of the folder, not the folder itself.
- Look at the status line for a warning about an ignored filter value.

A text is not found inside a file

- The file must be below a content folder (default: your user folder).

- It must not be larger than `content_max_size_mb` (default 16 MB).
- It must not be inside an excluded folder such as `AppData`, `node_modules` or `.git` (see Limits and exclusions).
- Online-only cloud files are not read. Make the file available offline to have it indexed.
- PDF and legacy Office files need a Windows IFilter for their extension.
- After a start, the content indexer may still be reading files: watch for *N files being read* in the tray tooltip, or `content pending` in `sgcfs status`.
- An uppercase letter makes the search case-sensitive (smart-case).
- Plain words next to a content filter must match the file name.

Why does the bar show lines from inside files?

No file or folder name matched your words, so the content fallback searched the content folders instead. The status line says so.

sgcfs says "sgcFSearch is not running"

`status` and `stop` do not start the app. Start `sgcFSearch.exe`, or run any search or `sgcfs show`, which start it in the background. If the start fails, check that `sgcFSearch.exe` is in the same folder as `sgcfs.exe`.

My manual changes to config.json have no effect

The settings are read at start. Run `sgcfs stop` and start sgcFSearch again, or use the Folders dialog, which restarts it for you.

15 Version history

1.0.0, November 2026

First release.

- Name index of every fixed drive, built by a multi-threaded crawler on the NT directory API and saved to `names.idx`.
- NTFS USN journal catch-up at startup and live change watcher.
- Search bar with a global hotkey (`Alt` + `Shift` + `Space`, with fallbacks), ranked results, icons, filter chips and history.
- Query syntax: name modifiers, path tokens, `ext:`, `type:`, `kind:`, `in:`, `path:`, `size:`, `mtime:`, `re:`, `limit:` and `sort:`.
- Content index of text files under the content folders, with `grep:`, `regex:` and `sym:` content search.
- Frecency ranking from opened and copied results.
- Named pipe JSON lines API restricted to the current user.
- `sgcfs` command line client: search, JSON output, stdio bridge, status, stop, show, install, uninstall and bench.
- Tray icon with the index status, Folders dialog, Start at login and Open log.
- Single instance: a second start shows the search bar of the running instance.
- Distributed as a signed per user installer (recommended) and as a zip. The setup offers the PATH entry and Start at login as tasks; with the zip, `sgcfs install` adds them.