# SOLIDWORKS PDM Search Syntax Guide

A practical reference for developers with general programming experience

---

## Overview

The PDM search system lets you build powerful search conditions using logical operators, wildcards, and variable bindings. There are two main ways to search:

| Method | Description |
|---|---|
| `FileName property` | Searches file/folder names only. Supports basic wildcards and OR grouping, but has limitations (see Section J). |
| `AddVariable2 / AddMultiVariableCondition` | Searches data card variable values. Supports the full syntax described in this guide. |

■ *Important: Each AddVariable2 / AddMultiVariableCondition call adds an extra required condition — a file must satisfy ALL of them to appear in results.*

## A. The Basics — Patterns, Wildcards, and Operators

### What is a "pattern"?

A pattern is simply the text you're searching for. By default, the search looks for values that **contain** the pattern anywhere — so `Metal` matches `Metal`, `Metallic`, `Sheet Metal`, etc.

To search for an exact phrase with spaces, wrap it in quotes: `"Ferrous Metal"`. Without quotes, `Ferrous Metal` means "contains Ferrous AND contains Metal" (in any order, anywhere).

### Wildcards

Wildcards only work in "contains" searches (without a comparison operator like = or <).

- `*` — matches any number of characters (including zero)
- `?` — matches exactly one character

```
437*         matches 437, 4370, 437156, 437-ABC, etc.
*sldprt      matches part.sldprt, 437156.sldprt, etc.
drawing?.slddrw  matches drawing1.slddrw but not drawing12.slddrw
```

■ *Without wildcards, they are added implicitly — `Metal` behaves the same as `*Metal*`.*

### Special Characters and Spaces

If your search value contains spaces or special characters (`&`, `|`, `!`, `(`, `)`, etc.), either wrap the whole thing in quotes or escape each character with a backslash `\`:

- `"Document Number"` — quotes around a name with a space
- `Document\ Number` — same using backslash escaping
- `A\&B;` or `"A&B;"` — searching for the literal string A&B;

To search for a literal quote, escape it: `\"`. To search for a literal backslash: `\\`.

## Comparison Operators

| Operator | Meaning |
|---|---|
| `=` | Exactly equals |
| `<> or !=` | Does not equal |
| `<` | Less than |
| `>` | Greater than |
| `<=` | Less than or equal to |
| `>=` | Greater than or equal to |

■ *Numbers are only compared numerically if the variable's data type is numeric. Same for dates.*

## Logical Operators

| Operator | Shortcut | Meaning |
|---|---|---|
| `AND` | `&` | Both conditions must be true |
| `OR` | `\|` | Either condition can be true |
| `NOT` | `!` | Condition must not be true |

■ *AND, OR, NOT as plain words are treated as literal search strings in basic mode — NOT as operators. Always use &, |, and ! for logic.*

**Order of operations** (without parentheses): `!` first, then `&`, then `|`. Use parentheses to override.

**AND is implicit** — writing two patterns side by side means AND: `Ferrous Metal` = `Ferrous & Metal`.

```
sldprt | sldasm          contains sldprt OR sldasm
437* & !prototype        starts with 437, does NOT contain prototype
(sldprt | sldasm) & 437* contains sldprt or sldasm, AND starts with 437
```

## B. Single-Value vs. Multi-Value Search Logic

This is one of the most important concepts to understand, as it changes how conditions are evaluated.

### Single-Value Search (Default)

All conditions in a single expression must be satisfied by the **same variable value**. For example: `56 & 7` applied to Document Number means "find a Document Number value containing both '56' and '7'." The value `567` satisfies this; `56` alone does not.

### Multi-Value Search

Activated by placing `:` or `@:` at the start of your condition. Different parts of the condition can be satisfied by *different* variable values or different variables.

For example: `:=5 =10` means "one variable equals 5 AND another equals 10." No single value needs to equal both.

### Using Braces {} to Force Single-Value Checking

In multi-value mode, wrap a sub-condition in `{}` to require it to be satisfied within a single value:

```
:{>=1 <=10}    a single value must be between 1 and 10
:>=1 <=10      one value >= 1, AND another value <= 10 (can be different values)
```

■ *In single-value mode (the default), braces don't change anything — conditions are already implicitly single-value.*

### Practical Example

Given these files in the vault:

- File A: Document Number = 567, Comment = xyz
- File B: Document Number = 567, Comment = ijk
- File C: Document Number = 123, Comment = abc

```
AddVariable2("Document Number", "56 & 7")  // matches 567 → Files A and B
AddVariable2("Comment", "xy | z NOT a")    // matches xyz only → File A

Result: File A only
```

## C. Negation — What Gets Returned and What Doesn't

Negation can be subtle, especially around documents with no variable values at all.

| Expression | What it returns |
|---|---|

| | |
|---|---|
| `!P` | Documents that do NOT contain P anywhere, including documents with no variable values |
| `{!P}` | Documents that HAVE at least one variable value, but that value doesn't match P |
| `!{P}` | Same as !P — everything not matched by P, including valueless documents |
| `P & !P` | Nothing — impossible to satisfy |
| `P \| !P` | Everything |

■ *Tip: If you're getting unexpected results from a negated search, check whether you want !P (includes valueless documents) or {!P} (excludes valueless documents).*

## D. Variable Bindings with @

The `@` operator lets you target a specific variable within a condition string — especially useful with `AddMultiVariableCondition`.

### Basic Variable Binding

```
@Comment xyz           Comment must contain xyz
@Author != Pete        Author must not equal Pete
@"Document Number" 56  Document Number must contain 56
```

### Binding to Multiple Variables

```
@(Author | Creator) != Pete    Either Author or Creator must not equal Pete
```

### Special Variables

- `""` or `0` — represents **all variables** in the database
- `_Name` — represents the **file or folder name**

■ *Important: `_Name` cannot be used as a binder inside a condition string. It must be passed in the variable names array of AddMultiVariableCondition. See Section F.*

### Binding with Parentheses

```
@Version P Q      Version must contain P; Q can be in any other variable
@Version(P Q)     Version must contain BOTH P and Q
```

### Combining Variable Logic

```
@("" & !Comment) P    Any variable except Comment must contain P
```

## E. Configuration Bindings with @@

`@@` works exactly like `@` but targets configurations (think: Default, Large, Small) instead of variables.

- `@@"@"` — the default configuration (named "@")
- `@@" "` — configuration 1 (always named with a single space)
- `@@""` or `@@0` — all configurations (this is the default)

```
@@ "@" P    Search only the default configuration for P
```

# F. Advanced Mode — @: and :

### Why Use Advanced Mode?

Basic syntax silently ignores errors and has limited variable targeting. Advanced mode gives you more power and better error reporting. Place `@:` or `:` at the very start of your condition string to activate it.

| Prefix | What it adds |
|---|---|
| `:` | Multi-value search logic |
| `@:` | Multi-value logic + variable/config binding inside the condition + syntax error reporting |

### Syntax Errors in Basic vs. Advanced Mode

In basic mode, a malformed condition silently returns no results. In `@:` mode, errors are reported and retrievable via `GetSyntaxErrors()`. **Always use `@:` during development.**

### Using _Name Correctly

`_Name` cannot be used as a binder inside a condition string. Pass it in the variable names array instead:

```
// CORRECT
string[] varNames = { "_Name" };
_search.AddMultiVariableCondition(varNames, ":437* & (*sldasm | *sldprt)");

// INCORRECT — causes a syntax error
_search.AddMultiVariableCondition(null, "@:_Name: 437* & (*sldasm | *sldprt)");
```

# G. Colon-Attached Specifiers

Adding a colon directly after certain keywords (no space before the colon) gives them the **lowest priority** — they apply to the entire expression that follows, not just the next term.

```
INT: >=1 <=10     INT applies to the whole expression (both conditions)
INT >=1 <=10      INT applies only to >=1
```

| Specifier | Effect |
|---|---|
| `TEXT:` | Treat all values as text |
| `INT:` | Treat all values as integers |
| `FLOAT:` | Treat all values as decimals |
| `DATE:` | Treat all values as dates |

| | |
|---|---|
| `AND:` | Spaces between terms mean AND (this is the default) |
| `OR:` | Spaces between terms mean OR |
| `NOT:` | Applies NOT to the whole expression |

### Colon as a Variable Binder Shortcut

Instead of `@(Author|Comment) john`, you can write:

```
Author:Comment: john

// For configurations, use double colons:
"@":: 1:: john
```

## H. Data Types

By default, the data type used for comparison is inherited from the variable's definition in the vault. You can override it explicitly:

```
TEXT abc                    compare as text
INT 100                     compare as integer
FLOAT 3.14                  compare as decimal
DATE >= "Jan 15, 2005"      compare as date
```

Data type specifiers have the same priority as `!`. Use parentheses to extend their scope:

```
INT >=1 <=10       INT applies only to >=1
INT(>=1 & <=10)    INT applies to both conditions
```

When searching across multiple variables with mixed types, the type is automatically resolved:

- Text mixed with non-text → TEXT
- Dates mixed with non-dates → TEXT
- Integers mixed with decimals → FLOAT

# I. Practical Examples

### Find a file by exact name

```
_search.FileName = "437156.sldprt";
```

### Find files starting with "437" — single extension

```
_search.FileName = "437*.sldprt";
```

### Find files starting with "437" — two extensions using OR grouping

OR grouping works when there is no wildcard before the group:

```
_search.FileName = "437156.(sldprt|sldasm)";   // works (no * before group)
```

OR grouping **does not** work when combined with a wildcard:

```
// Does NOT work:
_search.FileName = "437*.(sldprt|sldasm)";

// Use AddMultiVariableCondition with _Name instead:
string[] varNames = { "_Name" };
_search.AddMultiVariableCondition(varNames, ":437* & (*sldasm | *sldprt)");
```

### Find files where a variable matches a range

```
_search.AddVariable2("Revision", ">=A & <=F");
```

### Find files matching on two variables

```
// Document Number contains both "56" and "7" (e.g. "567")
// AND Comment contains "xyz"
_search.AddVariable2("Document Number", "56 & 7");
_search.AddVariable2("Comment", "xyz");
```

### Complex multi-variable search

```
// Find files where:
//   (Comment = abc AND any variable contains xyz)
//   OR (Document Number or filename contains 123 or ab, but not five)
string[] varNames = { };
_search.AddMultiVariableCondition(varNames,
    "@: @Comment=abc & @\"\"(xyz) | @(\"Document Number\" | _Name)(123 | ab & !five)");
```

### Check for syntax errors after a search

```
string[] errors = _search.GetSyntaxErrors();
if (errors != null)
    foreach (string e in errors)
        Console.WriteLine(e);
```

■ *Syntax errors are only reported when your condition starts with @:. In basic mode, malformed conditions silently return no results.*

## J. Known Limitations and Gotchas

| Issue | Details and Workaround |
|---|---|
| Wildcards and OR grouping don't mix in FileName | `437*.(sldprt\|sldasm)` returns zero results. Use AddMultiVariableCondition with _Name in the variable array, or run two separate searches. |
| _Name must go in the variable names array | Putting _Name inside the condition string causes a syntax error. Pass it as "_Name" in the string[] array argument to AddMultiVariableCondition. |
| AND/OR/NOT as plain words are literal strings | Use &, \|, and ! for logic. Plain-word versions are treated as search terms in basic mode. |
| Basic mode silently swallows errors | A condition like `56 && 7` is invalid but returns no error — just no results. Use @: prefix during development to surface errors via GetSyntaxErrors(). |
| AddVariable2 is implicitly single-value | Each condition is automatically wrapped in {} — all parts must be satisfied within a single variable value. Use AddMultiVariableCondition with : if you need conditions to span multiple values. |
| Multiple calls are ANDed together | Each AddVariable2 / AddMultiVariableCondition call adds a required condition. A file must satisfy all of them to appear in results. |