Syntax highlight in editors

NTI/PRK, 2025, Lenka Kosková Třísková

What do we mean by syntax highlight in editor

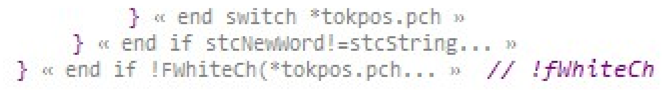
- Different coloring for the language tokens
- Bracket-balance check
- Identifying the possible errors on the fly during writing
- Increases productivity:
    - Programmers can skip keywords etc. and focus on algorithm.
    - Speeds up understanding the code (measured: https://ppig.org/files/2015-PPIG-26th-Sarkar1.pdf)
- Some editors can export highlighted code in HTML/XML/TeX
- **And the question is: How is it really done?**

Syntax highlight vs. syntax decoration

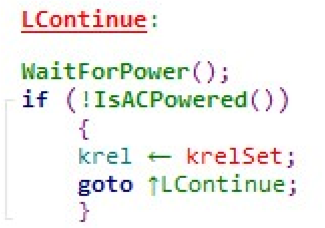
- **Syntax highlight changes the font, size, coloring:**

```
/* Hello World */

#include <stdio.h>

int main()

{

    printf("Hello World\n");

    return 0;

}
```

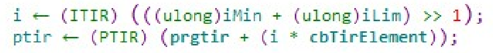
- **Syntax decoration replaces structure patterns with other symbols or graphics:**



```
        } « end switch *tokpos.pch »
    } « end if stcNewWord!=stcString... »
} « end if !FWhiteCh(*tokpos.pch... »  // !fWhiteCh
```



```
LContinue:

WaitForPower();
if (!IsACPowered())
    {
    krel ← krelSet;
    goto ↑LContinue;
    }
```



```
i ← (ITIR) (((ulong)iMin + (ulong)iLim) >> 1);
ptir ← (PTIR) (prgtir + (i * cbTirElement));
```

Example decorations taken from the SourceInsight manual (sourceinsight.com)

Standalone editor implementations

- **Regular expressions:**
  - Tokens defined by regular expressions
  - Each token has aligned highlight/decoration style
  - Limited - only lexical analysis can be done
- **Grammar based parsers:**
  - Tokens and syntax defined in the grammar
  - The grammar is aligned with the highligt/decoration style
  - Resource consuming
- Modern code editors have interfaces how to add another language support.

Syntaxes for syntax highlighting used in editors

- **TextMate** grammars - JSON based syntax, used also in VSCode.
- **Sublime** Text Syntax Definitions: JSON or YAML-based syntax definition format that is very similar in concept to TextMate's grammars, often being directly compatible or easily convertible.
- **Pygments**: Python syntax highlighting library that uses a Python-based definition format for lexers.

Shared language support: Language server protocol

- Implemented **by Microsoft in 2016 for VSCode**; now used in a lot of IDEs.

  https://microsoft.github.io/language-server-protocol/

- Open, JSON-RPC-based protocol standardizing the communication between source code editors or IDEs and language servers.
- **Server-Client principle:** Language server provides particular language support, clients in IDEs can connect and use it.
- Server, but no network: the server is mostly installed locally (it can be part of IDE/editor language plugin for example).
- Speeds up adding the new language into your IDE.
- List of implementations and editor support: **https://langserver.org/**

What is supported inside the LSP

- **Code Completion** (IntelliSense): Suggesting code as you type.
- **Go to Definition:** Jumping to the source code of a symbol.
- **Find All References:** Locating all uses of a symbol.
- **Syntax Highlighting** (Advanced): Providing more context-aware coloring.
- **Error Checking and Diagnostics** (Linters): Showing warnings and errors in your code.

- **Hover Information:** Displaying documentation or type information on hover.
- **Code Formatting:** Automatically formatting code according to style rules.
- **Rename Symbol:** Safely renaming variables and functions.
- **Signature Help:** Showing the parameters of a function.

Final set of supported features strongly depends on particular server implementation.

Python: Existing language servers

- **python-language-server (pyls):** A widely used, community-driven language server that leverages other popular Python tools like Jedi, Pyflakes, and more. (https://pypi.org/project/python-language-server/)
- **pylsp (Python LSP Server):** Another popular implementation focusing on performance and standards compliance. (https://github.com/python-lsp/python-lsp-server)
- **pyright:** A fast, static type checker written in TypeScript by Microsoft that also functions as a language server. (https://github.com/microsoft/pyright)
- **jedi-language-server:** A language server specifically built on top of the Jedi auto-completion and static analysis library for Python. Can be used in vim, emacs and vscode. (https://github.com/pappasam/jedi-language-server)

Java language servers

- **eclipse.jdt.ls**: The official language server from the Eclipse JDT project, providing comprehensive Java language support. (https://github.com/eclipse-jdtls/eclipse.jdt.ls)

  Clients for: VSCode, Emacs, Eclipse and other editors.

- **java-language-server**: Another actively developed Java language server. (https://github.com/georgewfraser/java-language-server)

  Clients for VSCode, Vim, Sublime

C/C++ language servers

- **clangd**: A fast and feature-rich language server based on the Clang compiler. It's part of the LLVM project. (https://clangd.llvm.org/)

  Plugins: VSCode, Sublime, vim

- **ccls:** Another powerful C/C++ language server focusing on performance and providing features like cross-referencing. (https://github.com/MaskRay/ccls)
- **clice:** A more modern C++ language server built from scratch, aiming to improve upon existing solutions. (https://github.com/clice-project/clice)

LSP details

- LSP focuses on the syntax of the communication between the editor/IDE and the language server.
- Communication runs in messages structured as JSON objects adhering to the JSON-RPC 2.0 specification.
- The protocol defines three main types of messages:
    - **Requests:** The client sends a request to the server and expects a response back (either a result or an error). Each request has a *unique id*, a *method* (string indicating the action to be performed), and *optional params* (a structured value containing the arguments for the method).
    - **Responses:** The server sends a response to a request, with id as the originating request. If the request was successful, the response contains a result/error field with a code, message, and optional data.
    - **Notifications:** The client or server can send notifications to the other party without expecting a response. Notifications have a method and optional params.

How it works

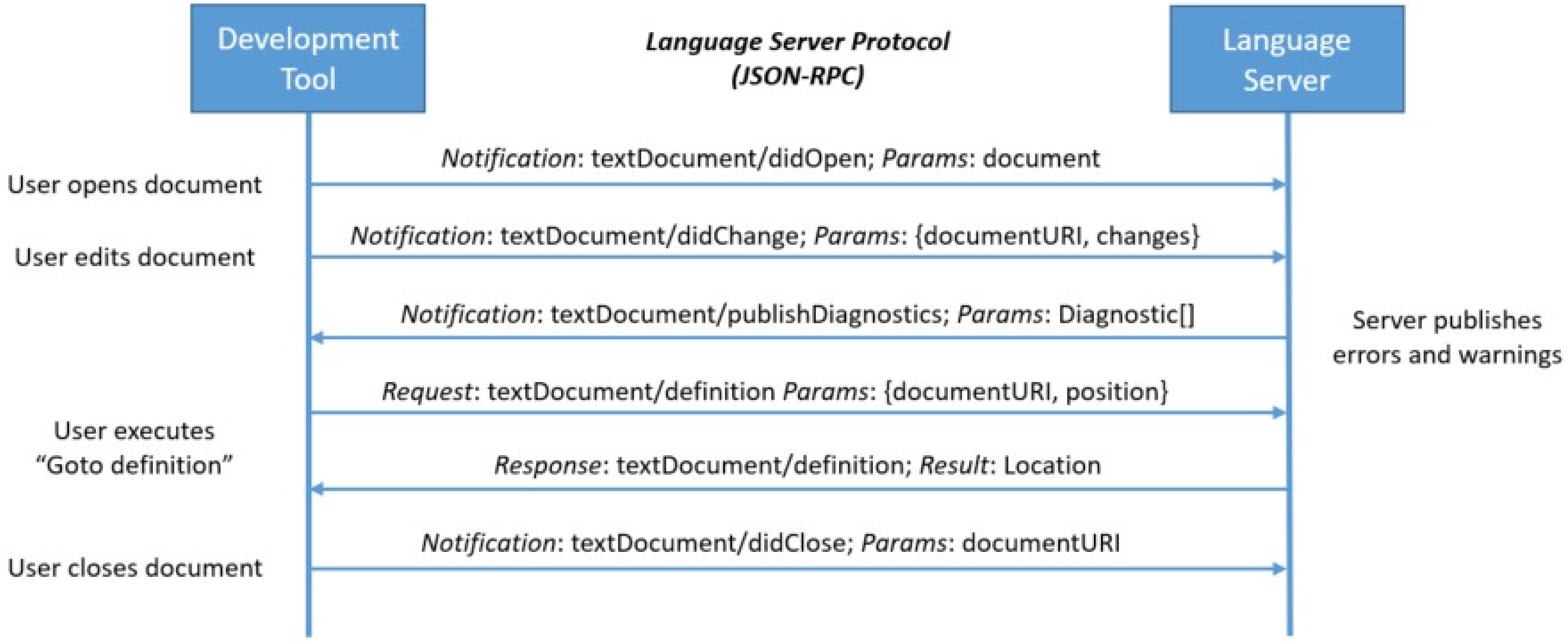

Development Tool
Language Server Protocol (JSON-RPC)
Language Server
Notification: textDocument/didOpen; *Params*: document
User opens document
Notification: textDocument/didChange; *Params*: {documentURI, changes}
User edits document
Notification: textDocument/publishDiagnostics; *Params*: Diagnostic[]
Server publishes errors and warnings
Request: textDocument/definition *Params*: {documentURI, position}
User executes "Goto definition"
Response: textDocument/definition; *Result*: Location
Notification: textDocument/didClose; *Params*: documentURI
User closes document

Image comes from LSP official page

Implementing your own language server - todo list

- **Text Document Management:** Handling events like opening, closing, changing, and saving text documents.
- **Diagnostics:** Analyzing code for errors and warnings and reporting them to the client. This often involves integrating with existing linters or static analysis tools for the language.
- **Completions:** Providing code completion suggestions based on the current context.
- **Go to Definition:** Locating the definition of a symbol (variable, function, class, etc.) in the codebase. This requires symbol resolution and indexing.
- **Hover Information:** Providing contextual information (e.g., documentation, type signatures) when the user hovers over a symbol.
- **Signature Help:** Displaying information about function parameters as the user types.
- **Formatting:** Implementing code formatting according to language-specific style guides.
- **Rename Symbol:** Implementing safe renaming of identifiers across the project.
- **Semantic Highlighting (Optional but Advanced):** Providing syntax highlighting based on the semantic meaning of code elements, often requiring more advanced parsing.

Syntax analysis within a language server

You need a language parser returning the AST. Select one of following (the most suitable

- Use existing parsers for the target language (if available as a library).
- Generate parser well suited for usage in the editor (incremental parsing, error tolerance) such as Tree-Sitter (https://tree-sitter.github.io/tree-sitter/).
- Generate a parser using standard tools like ANTLR, flex/yacc, or others.
- Write a parser manually, often using techniques like recursive descent.