Using Language Injections

On this page:

Learn more about language injections from the following pages in this part:

- Opening Language Injections in the Editor
- Closing an Editor for a Language Injection

Prerequisites

Before you start working with language injections, make sure that **IntelliLang** plugin is enabled.

The plugin is bundled with IntelliJ IDEA and activated by default. If not, enable the plugin as described in Enabling and Disabling Plugins.

Overview

IntelliJ IDEA makes it possible to work with islands of different languages embedded in the source code. You can *inject* other languages into *string literals*. This can be done within the source code written in most (but not all) of the supported languages (Java, JavaScript, Groovy, Python, Ruby, XML, PHP, and CSS for IntelliJ IDEA Ultimate Edition). The typical examples are HTML fragments injected into JavaScript code, SQL statements in Java or XML, and so on.

When the editor recognizes a string as a language injection:

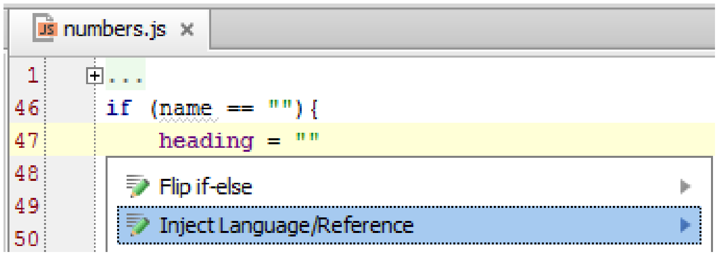
- Syntax and error highlighting and coding assistance are extended to this string.
- You can open and modify it in a separate tab in the editor, as if you were working with the source code in the corresponding language.

  To open an injection in the editor, use the **Edit <Language> Fragment** intention action.

Injecting a language into the source code and cancelling language injection

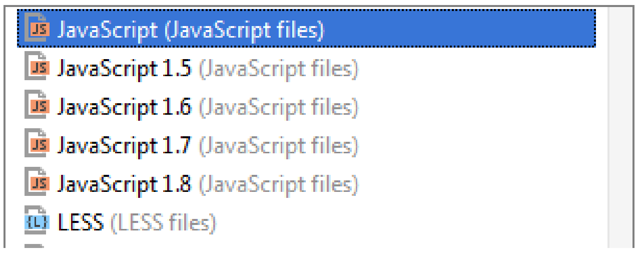
To tell IntelliJ IDEA that certain text should be treated as an embedded source code fragment, you can use:

- The **Inject Language/Reference** intention action, which is applied to a particular string:

  ```
  ...
  if (name == ""){
      heading = ""
  ```

  After that, you have to select the desired language from the menu:

  
  JavaScript (JavaScript files)
JavaScript 1.5 (JavaScript files)
JavaScript 1.6 (JavaScript files)
JavaScript 1.7 (JavaScript files)
JavaScript 1.8 (JavaScript files)
LESS (LESS files)

  Note that if you use this method, it's possible that the string literal will stay marked as a language injection only within a limited period of time. That is, IntelliJ IDEA, at a certain moment, may "forget" that the corresponding literal is a language injection. The period of the injection "persistence" will depend on the language, context and the modifications that you make in other parts of your source code.

- A dedicated @Language annotation in Java source code, for example, `@Language("JavaScript")` or `@Language("HTML")`.

- Comments such as `//language=<language_ID>`, `/*language=<language_ID>*/`, `#language=<language_ID>`, `<!--language=<language_ID>-->`, `--language=<language_ID>`. For example:

  ```
                  //language=SQL
  String s = "
                  select * from test.users";
  ```

  For more information, see Using comments.

- The Language Injection page of the Settings dialog. By creating new language injection configurations on that page, you can specify that a certain method parameter, text in an XML tag, or XML attribute should always be treated as an embedded source code fragment in another language.

  IntelliJ IDEA comes with a set of predefined injection configurations which is quite sufficient to ensure high productivity and comfortable environment. Therefore it is strongly recommended that you use the predefined injection configurations and avoid creating new ones.

To cancel language injection, choose the intention action **Un-inject Language/Reference**.

Toggling between global and project injections

IntelliJ IDEA distinguishes the *project* and *global* states of injection configurations.

- Character strings configured as *project* injections are treated as source code only within the current project.
- Character strings configured as *global* injections are treated as source code at the IntelliJ IDEA level, that is, within any IntelliJ IDEA project.

To toggle between the *project* and *global* states, use the **Move to Project/Make Global** toolbar button on the Language Injection page of the Settings dialog box.

Using comments

Here is the syntax to be used when declaring language injections by means of comments:

```
//language="<language_ID>" prefix="<prefix>" suffix="<suffix>"
```

`<language_ID>` is an ID of the injected language. The `prefix` and `suffix` are optional.

If specified, the prefix is added before the string literal and the suffix - after. The prefix, the string literal itself and the suffix are added all together to form a single injected language fragment.

Quotation marks are used if there are spaces within the language ID, prefix or suffix. Only double quotes can be used (").

In most of the cases the language ID is intuitive, e.g. CSS, HTML, SQL, MySQL, DB2, Oracle, PostgreSQL and so on. If you are not sure about the language ID, use the suggestion list for the **Inject Language** intention action as the source of information. For example, type `""` in the editor and place the cursor between " and ". Press **`Alt+Enter`** and select **Inject Language**. In the list that is shown, what precedes the opening parentheses are language IDs. For example, within the list item **AIDL (Android IDL files)**, **AIDL** is a language ID.

The comment should precede the string literal to which it applies and should be placed as close as possible to that literal.

The following examples illustrate the use of comments for language injections in Java code.

MySQL:

```
               //language=MySQL
return "
                 select * from test.employees";
```

CSS:

```
myMethod
       //language=CSS prefix="body {" suffix=}
       ("\n    color: #00ff00;\n    font-size: 120%n");
```

JavaScript 1.8:

```
String s =
       //language="JavaScript 1.8"
                 "
                 var a;\na = 1;";
```

See Also

Procedures:

- Opening Language Injections in the Editor
- Closing an Editor for a Language Injection

Reference:

- Language Injections

External Links:

- IntelliLang
- Full-featured Editor for Injected Language Fragments

Getting Started:

- Familiarize Yourself with IntelliJ IDEA Editor

Web Resources:

- Developer Community