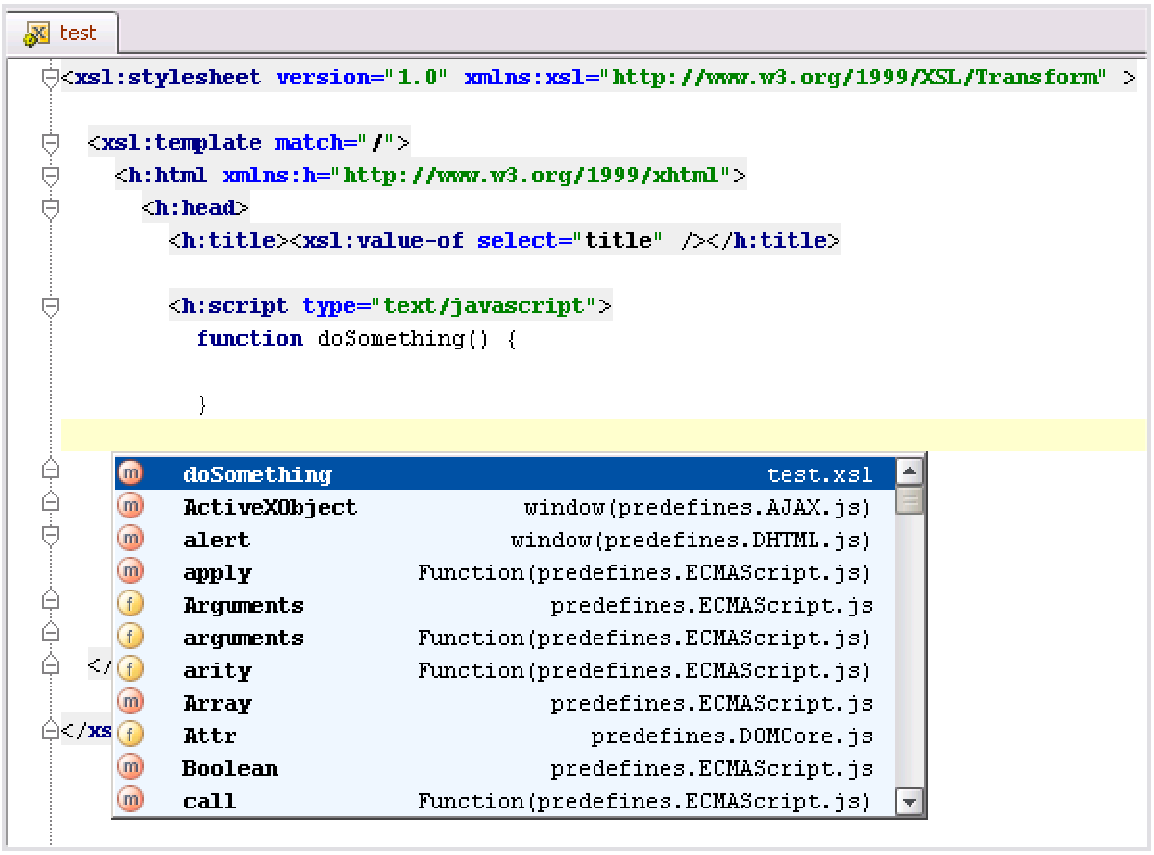
Usage Examples

Among the rather simple use-cases, like having detailed syntax checks for all kinds of "micro-languages" that are used e.g. in `Pattern.compile()`, `XPath.compile()` and so on, here are some less obvious, yet very useful examples how IntelliLang can leverage IntelliJ IDEA's support for a much better coding assistance.

The new scripting support in the upcoming Java 1.6 is another case where it will be important to get as much as possible edit-time assistance when script-code is constructed from Java code.

Extended JavaScript support

When dealing with JavaScript that's not directly embedded inside an HTML page, IntelliJ IDEA usually just treats it as plain text. Consider the following example that creates an HTML page from an XSLT script. Without the JavaScript language being injected into the *script* tag with the XHTML namespace as shown in the screenshot below, this would be treated as plain text, with no further code assistance.



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

  <xsl:template match="/">
    <h:html xmlns:h="http://www.w3.org/1999/xhtml">
      <h:head>
        <h:title><xsl:value-of select="title" /></h:title>

        <h:script type="text/javascript">
          function doSomething() {

          }

  </
</xs
```

```
doSomething         test.xsl
ActiveXObject       window(predefines.AJAX.js)
alert               window(predefines.DHTML.js)
apply               Function(predefines.ECMAScript.js)
Arguments           predefines.ECMAScript.js
arguments           Function(predefines.ECMAScript.js)
arity               Function(predefines.ECMAScript.js)
Array               predefines.ECMAScript.js
Attr                predefines.DOMCore.js
Boolean             predefines.ECMAScript.js
call                Function(predefines.ECMAScript.js)
```

Support for JSP custom cags

With IntelliLang it's also possible to make the content and attributes of custom JSP tags being treated as another language. This can be useful e.g. for server-side scripting using JavaScript or any other Language implementation available for IntelliJ IDEA.

One thing that's important to know is that the taglib's URI which supplies a custom tag should be used as the namespace URI of the XML tag to inject a language into. The namespace-textfields contain a list of all known taglib URIs in the project.

Unfortunately, at the moment the support for refactoring and navigation inside JSP custom tags seems to be broken and attempting to use code completion may result in exceptions thrown in IntelliJ IDEA core. See also the known issues below.

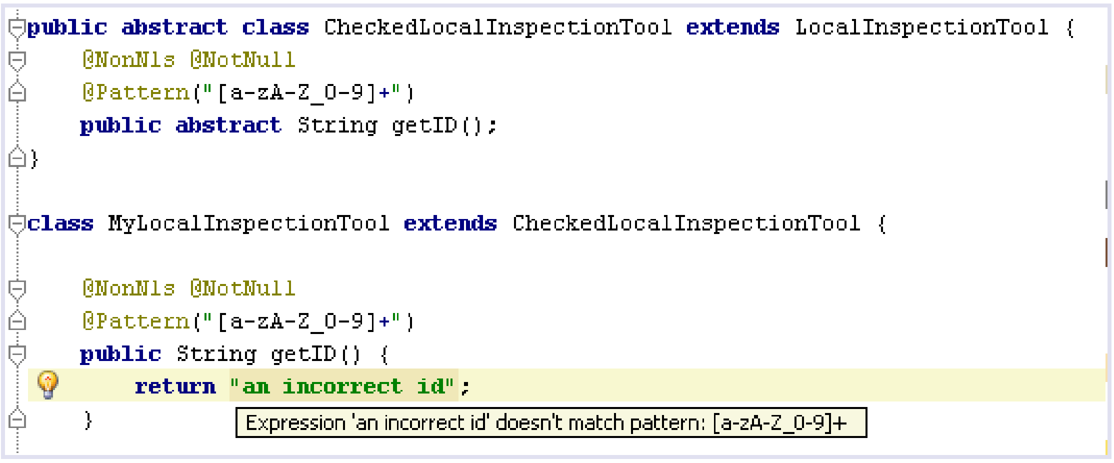
Pattern validation

Here's an obvious example right from IntelliJ IDEA's OpenAPI:

```
/** com.intellij.codeInspection.LocalInspectionTool

     * @return descriptive name to be used in suppress comments and annotations,
     *         must consist of [a-zA-Z_0-9]+
     */
    @NonNls @NotNull public String getID() {
      return getShortName();
    }
```

The contract of the method `getID()` is that it should only return strings that match the pattern "[a-zA-Z_0-9]+". The short note in the JavaDoc can be easily overlooked though because the contract isn't specified in an automatically verifiable way.

However, if this method were annotated as `@Pattern("[a-zA-Z_0-9]+")`, any attempt to return a string that doesn't match that pattern would be flagged in the editor:

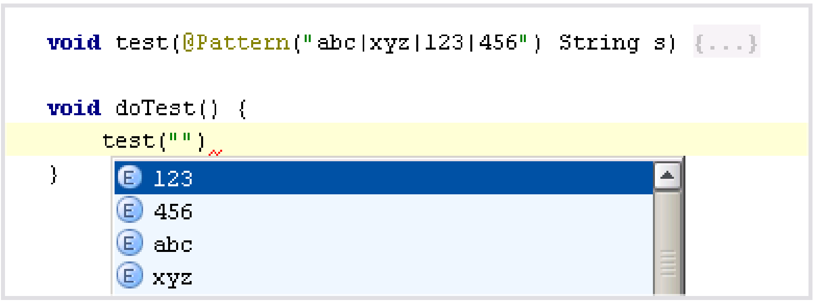


```
public abstract class CheckedLocalInspectionTool extends LocalInspectionTool {
    @NonNls @NotNull
    @Pattern("[a-zA-Z_0-9]+")
    public abstract String getID();
}

class MyLocalInspectionTool extends CheckedLocalInspectionTool {

    @NonNls @NotNull
    @Pattern("[a-zA-Z_0-9]+")
    public String getID() {
        return "an incorrect id";
    }
```

Expression 'an incorrect id' doesn't match pattern: [a-zA-Z_0-9]+

Pattern completion

If a regular expression pattern represents an enumeration of different literal values, the plugin offers completion for those values:

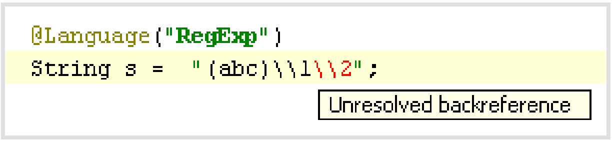


```
void test(@Pattern("abc|xyz|123|456") String s) {...}

void doTest() {
    test("")
}
```

Regular expression editing

Here are some examples of the enhanced coding support for regular expression patterns:

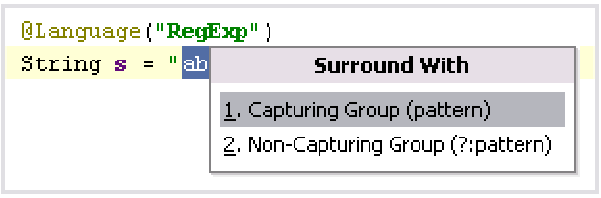
Backref validation



```
@Language("RegExp")
String s =  "(abc)\\1\\2";
```

Unresolved backreference

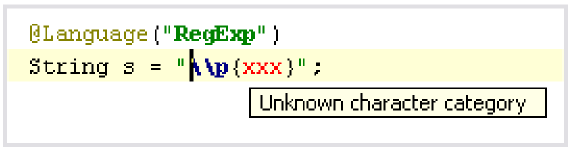
Surround with



```
@Language("RegExp")
String s = "ab
```

Surround With
1. Capturing Group (pattern)
2. Non-Capturing Group (?:pattern)

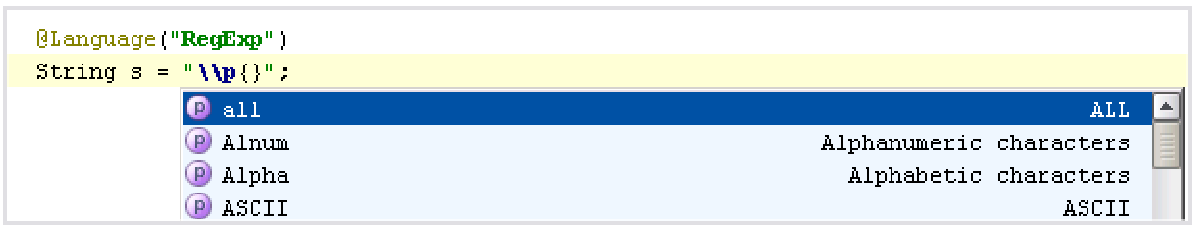
Character category validation



```
@Language("RegExp")
String s = "\\p{xxx}";
```

Unknown character category

Character category completion



```
@Language("RegExp")
String s = "\\p{}";
```

```
all                                          ALL
Alnum                       Alphanumeric characters
Alpha                         Alphabetic characters
ASCII                                        ASCII
```