XPath Inspections

The XPath language implementation provides a few built-in inspections that can check for common coding mistakes when writing XPath expressions both in the interactive mode and when writing XSLT scripts. Those inspection also provide a number of configuration options which can be configured on the Inspections page of the Settings dialog box.

> Due to the way how these inspections are implemented and integrated, they only work for the on-the-fly editor highlighting and *not* if the inspections are run via **Analyse | Inspect Code** .

XPath type checking

There are two inspections that deal with type-conversion in XPath expressions: **Implicit Type Conversion** and **Redundant Type Conversion**.

Implicit type conversion

This inspection checks for any implicit conversions between the predefined XPath-types STRING, NUMBER, BOOLEAN and NODESET. While this is usually not a problem as the conversions are well-defined by the standard, this inspection can help to write XSLT scripts that are more expressive about types and can even help to avoid subtle bugs:

`<xsl:if test="foo" />` is not the same as `<xsl:if test="string(foo)" />`

The first test checks whether the element *foo* exists (`count(foo) > 0`), the latter one however is only true if the element actually contains any text (`string-length(foo) > 0`). The plugin will then offer to make the type-conversion more explicit.

There are several options to adjust the inspection to personal preferences by offering the possibility to individually enable it for implicit conversions between certain types.

The plugin can also be told to always flag explicit conversions that do not result in the actually expected type, such as `<xsl:if test="number(foo)" />` and provides a special option to ignore the conversion from NODESET to BOOLEAN by using the `string()` function as a shortcut for writing `string-length() > 0`.

Redundant type conversion

This inspection checks whether any type-conversion with the functions `string()`, `number()` or `boolean()` is redundant, i.e. whether the type of argument is the same as the functions return type or if the expected type of the expression is of type *any*. While such an explicit conversion may sometimes be intentional to emphasize the type, this can usually be safely removed.

Expression validity checks

Those inspections check whether an expressions contains any potential semantic mistakes, such as referencing element/attribute names that don't occur in instance documents or using predicates that don't potentially match anything.

Check node test

This inspection checks whether any element/attribute names that are used in XPath-expressions are actually part of an associated XML file or are defined in a referenced schema. This helps to avoid problems caused by typos in XPath-expressions that would otherwise occur when running the script and may even then not be recognized immediately.

Example:

```
<xsl:template match="h:txtarea" />
```

If the prefix *h* is bound to the XHTML namespace, the inspection will flag this part of the match-expression as an unknown element name because the correct name of the element is *textarea*.

Index zero usage

This inspection checks for any accidental use of zero in a predicate index or in a comparison with the function `position()`. Such is almost always a bug because in XPath, the index starts at one, *not* at zero.

Example:

`//someelement[position() = 0]` or `//something[0]`

Developing custom XPath inspections

The XPath inspections make use of the normal inspections API of IntelliJ IDEA. However, due to the way the XPath Language-Support is integrated, this is a bit more complicated and it's at the moment not readily possible to develop full-blown 3rd-party XPath inspections. While it's theoretically possible to develop custom inspections that make use of the XPath-PSI API and are derived from `org.intellij.lang.xpath.validation.inspections.XPathInspection`, this is not recommended and not supported.

Please contact me if there's a need for a special inspection or there is significant interest that would justify the effort to make this more pluggable.