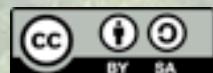


xQuery Language Aware Plugin for IntelliJ

Ron Hitchens
Principal Consultant, OverStory Ltd
ron@overstory.co.uk



Licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
© 2012 OverStory Ltd

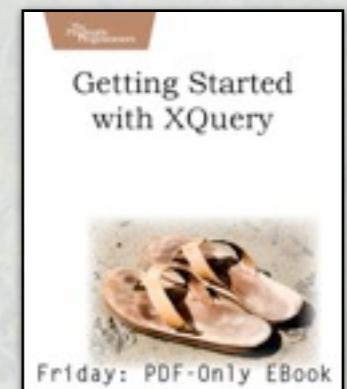
Who Am I?

Former Lead Engineer at MarkLogic (5 years)

Created XCC, way back when

Written lots of XQuery over the years

Even wrote an XQuery book (Pragmatic)



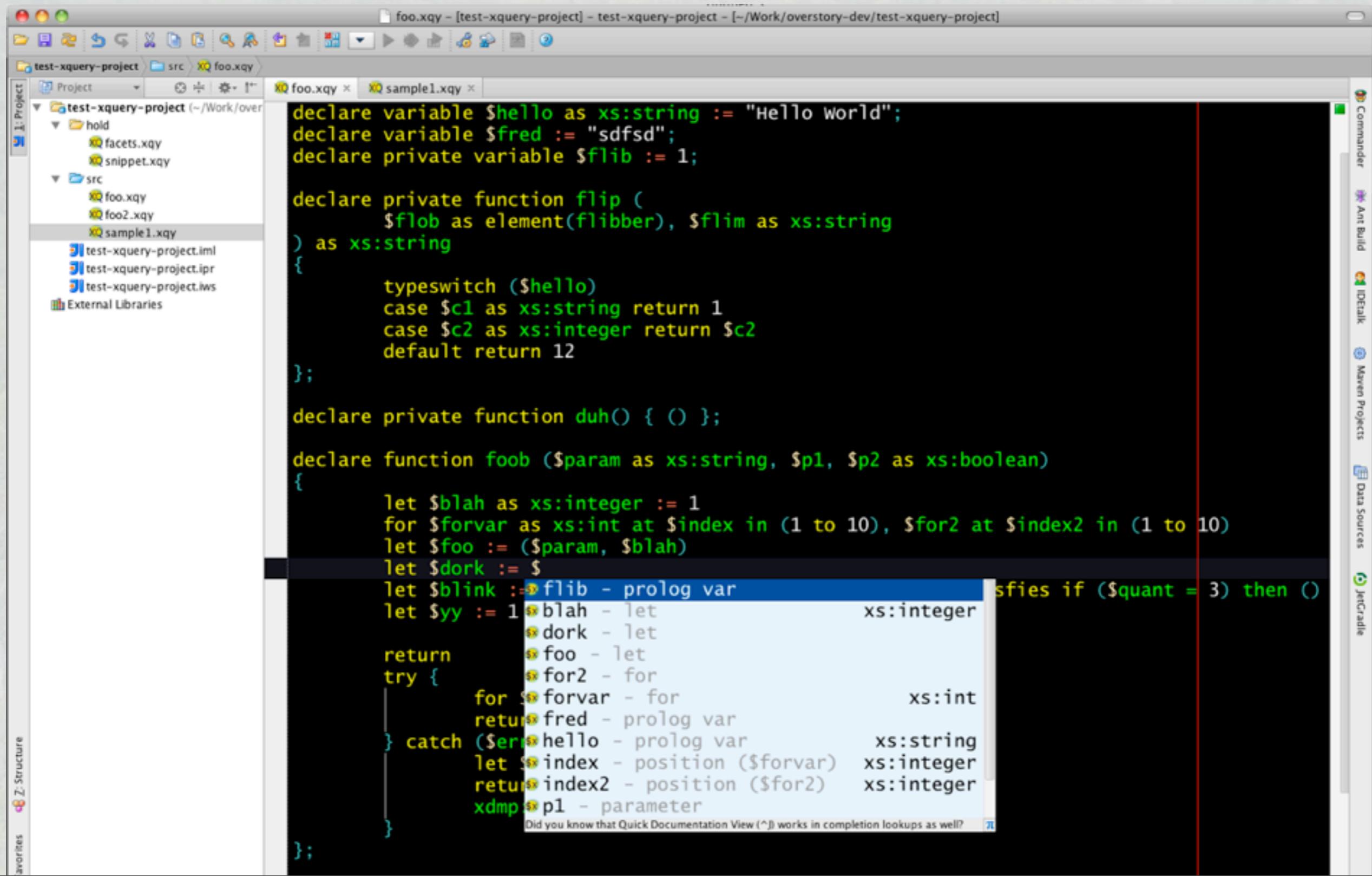
IntelliJ has always been my favorite Java IDE

Used it to create and test XCC

Always wished it could do for XQuery what it does for Java

Started a plugin while at MarkLogic, in 2005

Wouldn't This Be Nice?



A screenshot of an IDE interface showing an XQuery file named 'foo.xqy' in the editor. The code is as follows:

```
declare variable $hello as xs:string := "Hello World";
declare variable $fred := "sdfsdf";
declare private variable $lib := 1;

declare private function flip (
    $blob as element(flibber), $flim as xs:string
) as xs:string
{
    typeswitch ($hello)
    case $c1 as xs:string return 1
    case $c2 as xs:integer return $c2
    default return 12
};

declare private function duh() { 0 };

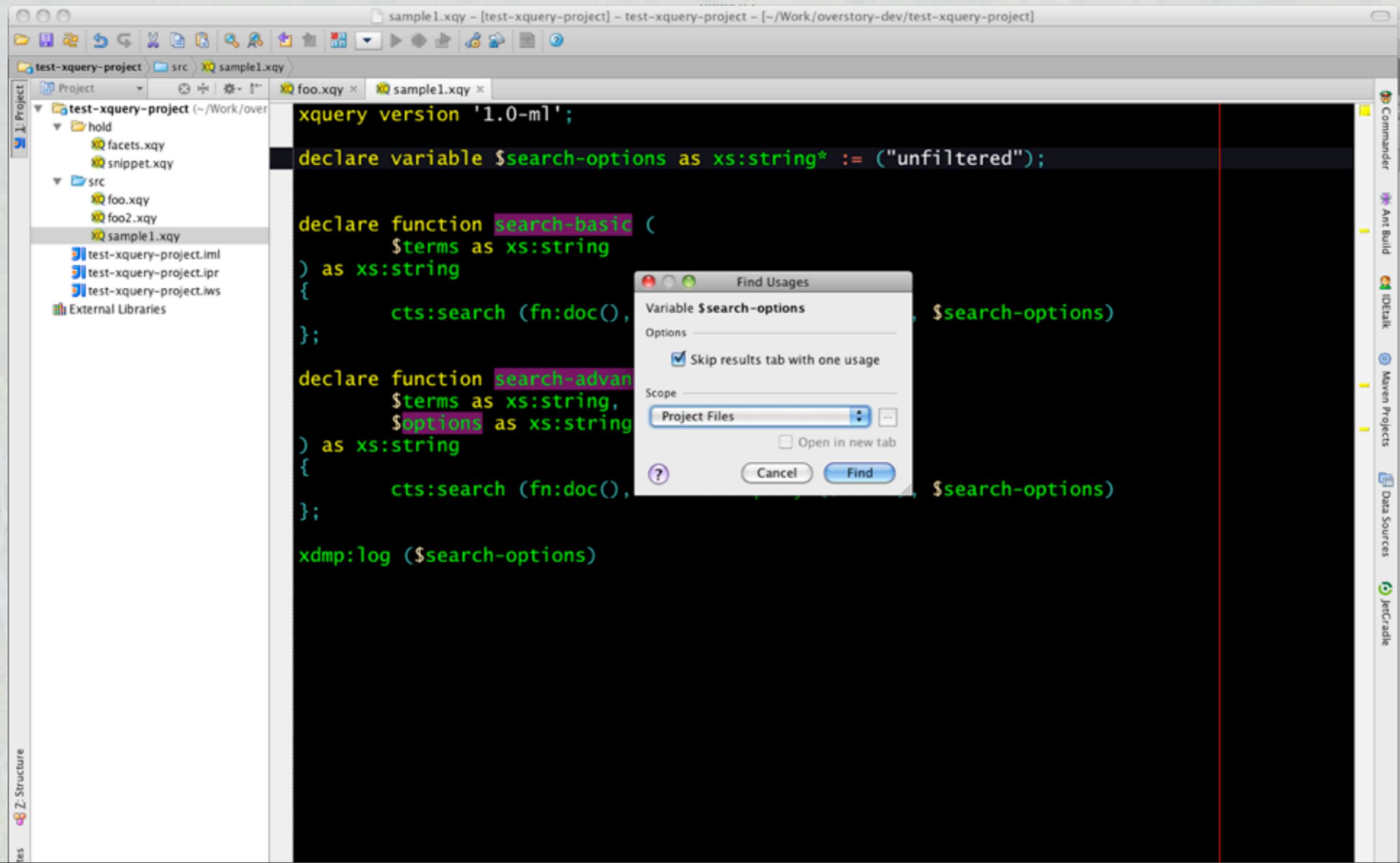
declare function foob ($param as xs:string, $p1, $p2 as xs:boolean)
{
    let $blah as xs:integer := 1
    for $forvar as xs:int at $index in (1 to 10), $for2 at $index2 in (1 to 10)
    let $foo := ($param, $blah)
    let $dork := $param
    let $blink := $lib - prolog var
    let $yy := 1
    return
    try {
        for $forvar - for
        return $fred - prolog var
    } catch ($err)
    let $index - position ($forvar)
    return $index2 - position ($for2)
    xdmp:log($p1 - parameter)
};
```

A completion dropdown is open at the bottom of the code editor, showing suggestions for variables and functions. The dropdown includes:

- \$lib - prolog var
- \$blah - let
- \$dork - let
- \$foo - let
- \$for2 - for
- \$forvar - for
- \$fred - prolog var
- \$hello - prolog var
- \$index - position (\$forvar)
- \$index2 - position (\$for2)
- xdmp:log(\$p1 - parameter)

The IDE interface includes a toolbar, a project tree on the left, and various toolbars and panels on the right.

Where Is This Thing Used?



The screenshot shows an IDE interface with a dark theme. The left sidebar displays a project structure for 'test-xquery-project' with files like 'facets.xqy', 'snippet.xqy', 'foo.xqy', 'foo2.xqy', and 'sample1.xqy'. The main editor window shows an XQuery script:

```
xquery version '1.0-m1';

declare variable $search-options as xs:string* := ("unfiltered");

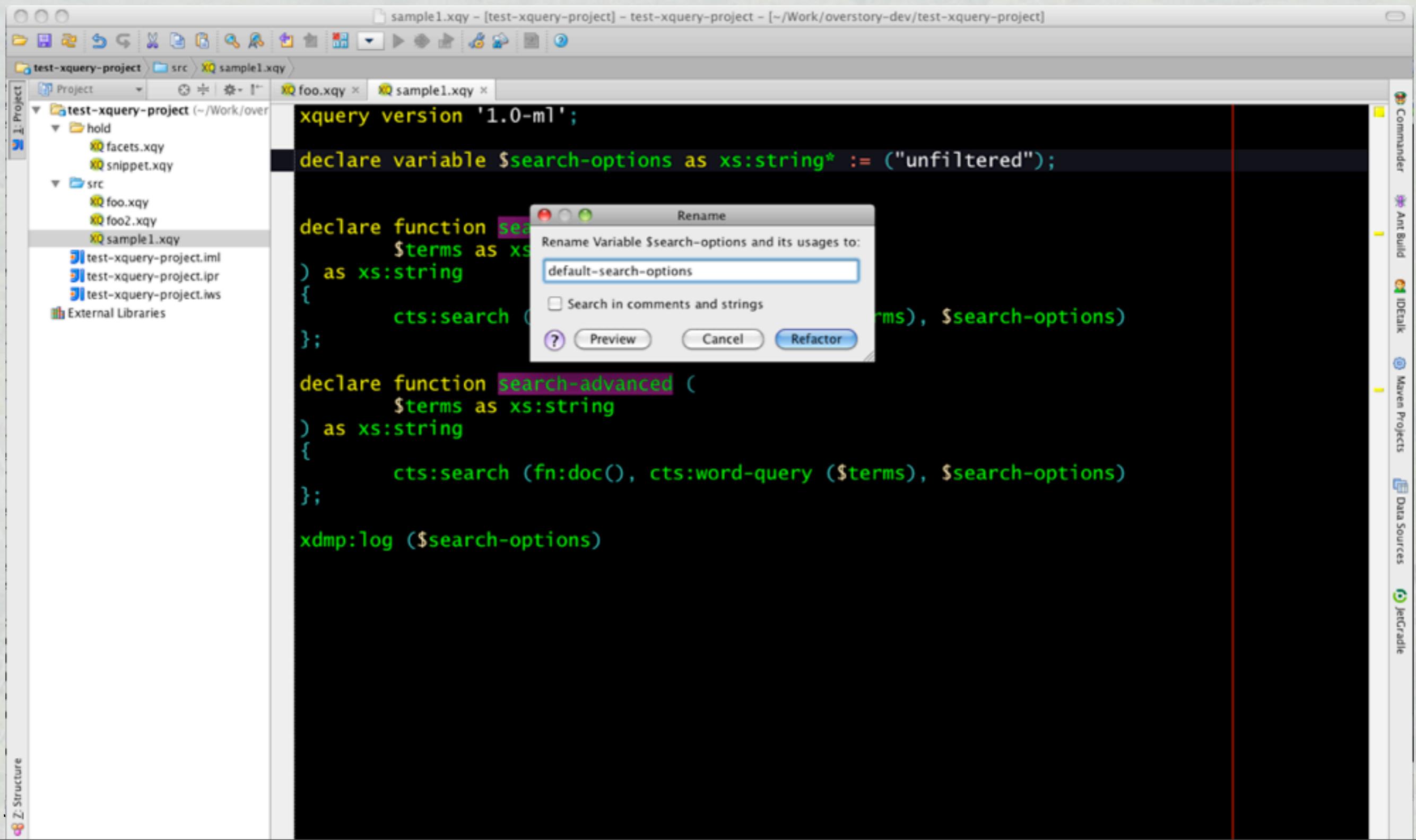
declare function search-basic (
    $terms as xs:string
) as xs:string
{
    cts:search (fn:doc(),
};

declare function search-advan
    $terms as xs:string,
    $options as xs:string
) as xs:string
{
    cts:search (fn:doc(),
};

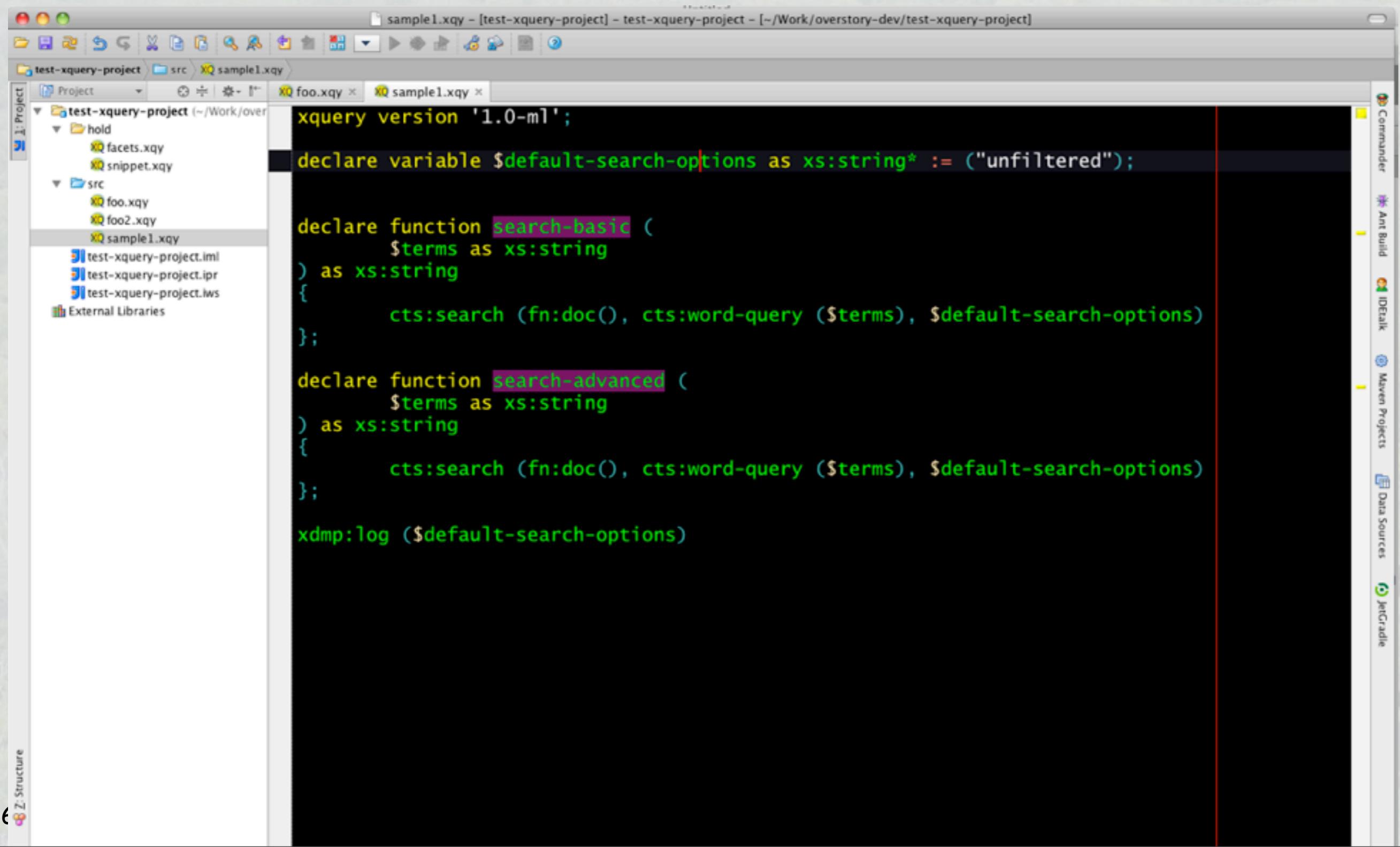
xdmp:log ($search-options)
```

A 'Find Usages' dialog is open in the center, with the variable '\$search-options' selected. The dialog includes options for 'Skip results tab with one usage' and 'Scope' set to 'Project Files'. The 'Find' button is highlighted.

How About A Name Change



Variable Renamed



The screenshot shows an IDE interface with a dark theme. The title bar reads "sample1.xqy - [test-xquery-project] - test-xquery-project - [/Work/overstory-dev/test-xquery-project]". The left sidebar shows a project structure for "test-xquery-project" with "src" and "sample1.xqy" selected. The main editor window contains the following XQuery code:

```
xquery version '1.0-ml';

declare variable $default-search-options as xs:string* := ("unfiltered");

declare function search-basic (
    $terms as xs:string
) as xs:string
{
    cts:search (fn:doc(), cts:word-query ($terms), $default-search-options)
};

declare function search-advanced (
    $terms as xs:string
) as xs:string
{
    cts:search (fn:doc(), cts:word-query ($terms), $default-search-options)
};

xdmp:log ($default-search-options)
```

The variable `$default-search-options` is highlighted in green, indicating it has been renamed. The code is syntax-highlighted, with keywords in blue, strings in green, and variables in purple.

XQuery Language Plugin

Hooks into IntelliJ's Program Structure Interface

XQuery lexer/parser builds an internal model

IntelliJ calls into the plugin to manipulate the model

The plugin also traverses the model

Validation, type-ahead suggest, usage checks, etc

Refactoring of various flavors

Common editor, customized to your preferences

Switch easily between languages in one IDE

Current State

Basic Functionality is in Place

XQuery-specific syntax highlighting

XQuery & XML-specific bracket/brace matching

Jump to definition, find usages for vars and functions

Auto-suggest in-scope variables and functions

Warn unused, error undefined vars and functions

Rename variables and functions

Comment/uncomment in the XQuery style

DEMO

Where It Stands Now

Basic functionality is mostly there

Auto-suggest, API info, usages, rename, warnings, etc

Syntax-aware code traversal

Parser/Lexer still rather brittle

Many features stop working with invalid xQuery

XML parsing still a bit flaky

Not quite ready for prime time

Soon, I hope

Parser State

Uses the Grammar Kit from JetBrains

Generates a parser from a BNF spec

Somewhat customizable, but poorly documented

Not very forgiving of malformed XQuery, yet

Features that use the model stop working when the parser can't build a complete model

Editor-based features, like brace matching, still work

Parser improvement is the next priority

Intermediate Plans

Contextual auto-complete, BNF-driven

Next legal XQuery construct at cursor, with look-ahead

More and better annotations

Type mis-matches in assignments and parameters

Variables hiding other variables

Unused imports and namespace declarations,

Invalid values for options/parameters, where known

Matching function parameters to signatures

Longer Term Plans

Resolve namespace values across modules

Contextual documentation

Code Intentions

Full refactoring

Extract, introduce, etc

Warn about unresolvable import paths

Beyond Editing

Execution from IntelliJ

Send request, capture result in an edit buffer

Jump to error source line for syntax exceptions

Profiling

Capture history to plot code improvements

Correlate performance with code checkins

Debugging

Testing framework integration

Help Me!

If you want to contribute to the project, especially if you love tinkering with parsers & lexers, please contact me:

ron@overstory.co.uk

<http://github.com/overstory>

Questions?

Ron Hitchens
Principal Consultant, OverStory Ltd
ron@overstory.co.uk



Licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
© 2012 OverStory Ltd