

# XQuery Language Aware Plugin for IntelliJ

Ron Hitchens

Principal Consultant, OverStory Ltd

[ron@overstory.co.uk](mailto: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?

The screenshot shows an IDE window titled "foo.xqy - [test-xquery-project]". The main editor displays XQuery code with a completion popup for the variable "\$dork". The code includes variable declarations, function definitions, and a complex function call. The completion popup lists several options with their corresponding types.

```
declare variable $hello as xs:string := "Hello world";
declare variable $fred := "sdfsd";
declare private variable $flib := 1;

declare private function flip (
  $flob 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() { () };

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 := $
  let $blink := flip - prolog var
  let $yy := 1 $blah - let
  return
  try {
    for $forvar - for
    return $fred - prolog var
  } catch ($err) $hello - prolog var
  let $index - position ($forvar)
  return $index2 - position ($for2)
  xdmp:p1 - parameter
};
```

Completion popup options:

- \$dork - let xs:integer
- \$foo - let
- \$for2 - for
- \$forvar - for xs:int
- \$fred - prolog var
- \$hello - prolog var xs:string
- \$index - position (\$forvar) xs:integer
- \$index2 - position (\$for2) xs:integer
- \$p1 - parameter

Additional text in the popup: "sfiles if (\$quant = 3) then ()"

Footer text: "Did you know that Quick Documentation View (^) works in completion lookups as well?"

# Where Is This Thing Used?

The image shows a screenshot of an IDE window titled "sample1.xqy - [test-xquery-project] - test-xquery-project - [~/Work/overstory-dev/test-xquery-project]". The main editor displays XQuery code with the following content:

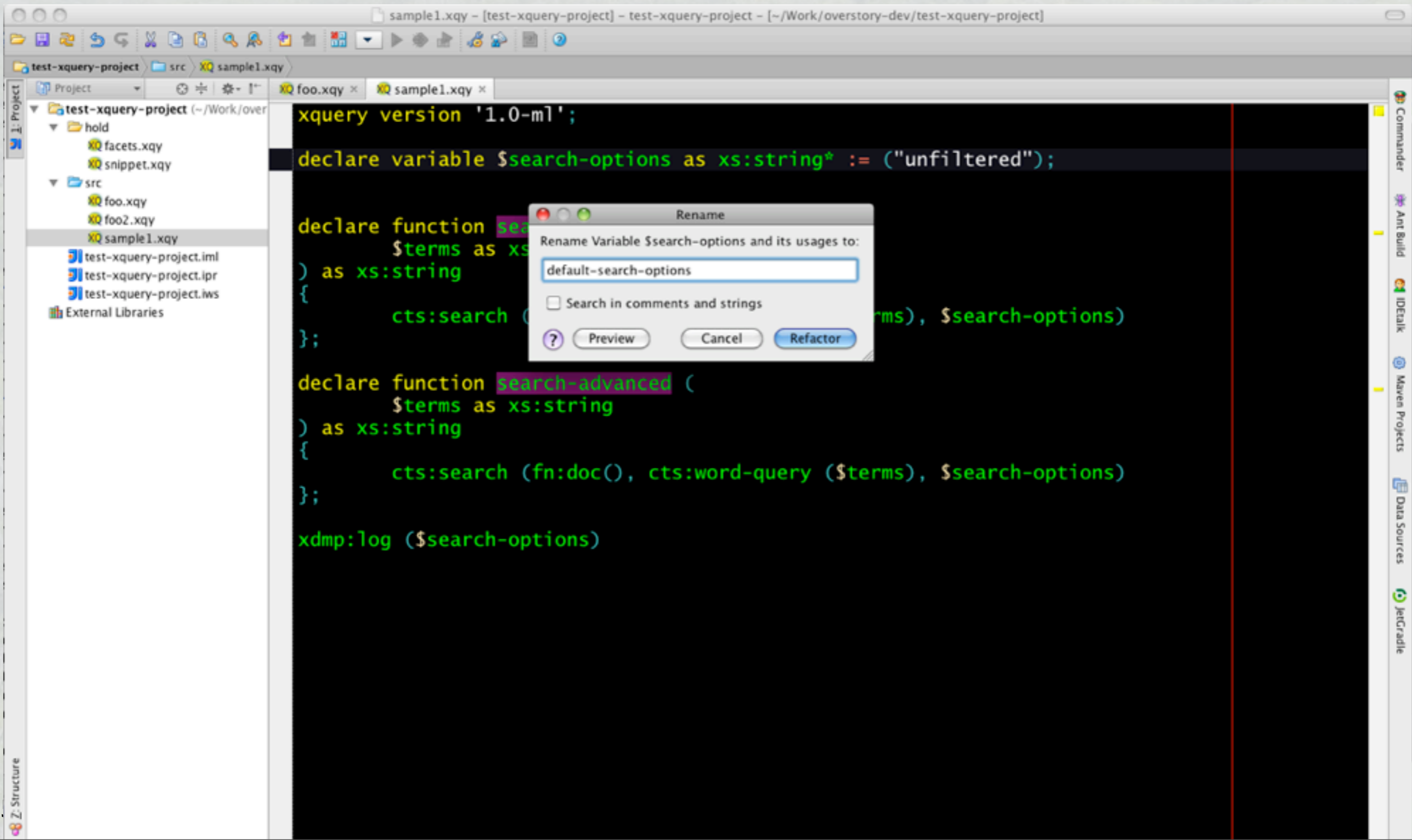
```
xquery version '1.0-m1';  
  
declare variable $search-options as xs:string* := ("unfiltered");  
  
declare function search-basic (  
    $terms as xs:string  
) as xs:string  
{  
    cts:search (fn:doc(), $search-options)  
};  
  
declare function search-advanced (  
    $terms as xs:string,  
    $options as xs:string  
) as xs:string  
{  
    cts:search (fn:doc(), $search-options)  
};  
  
xdmp:log ($search-options)
```

A "Find Usages" dialog box is open over the code, showing the following details:

- Variable: `$search-options`
- Options: (empty)
- Skip results tab with one usage
- Scope: Project Files
- Open in new tab
- Buttons: Cancel, Find

The IDE interface includes a project explorer on the left showing the "test-xquery-project" structure, and a vertical toolbar on the right with icons for Commander, Ant Build, IDEtalk, Maven Projects, Data Sources, and JetCradle.

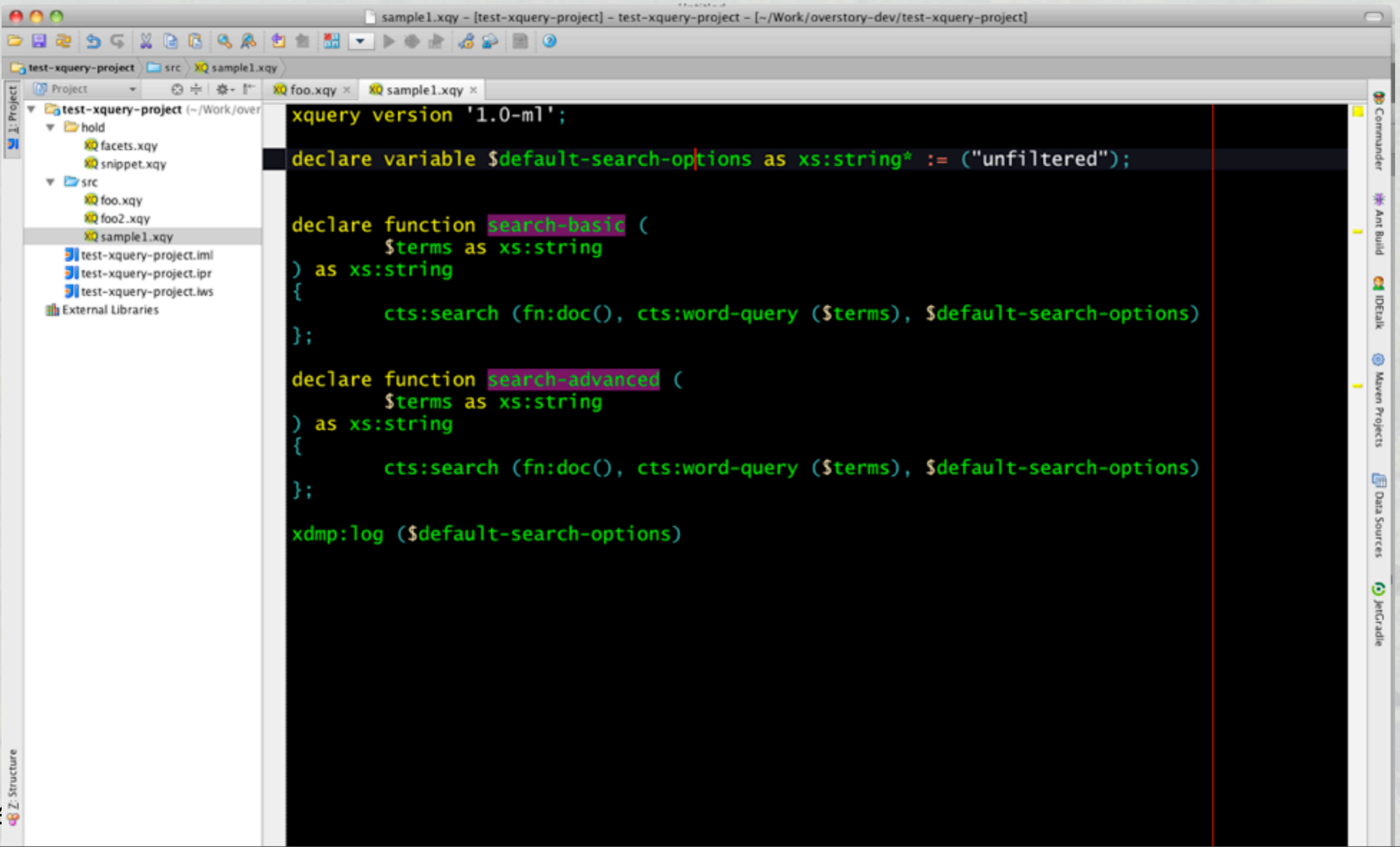
# How About A Name Change



The screenshot shows an IDE window with a project named 'test-xquery-project'. The main editor displays XQuery code. A 'Rename' dialog box is open, prompting the user to rename the variable '\$search-options' to 'default-search-options'. The dialog includes a text input field with the new name, a checkbox for 'Search in comments and strings', and buttons for 'Preview', 'Cancel', and 'Refactor'. The code in the background includes:

```
xquery version '1.0-m1';  
declare variable $search-options as xs:string* := ("unfiltered");  
  
declare function search-advanced (  
    $terms as xs:string  
) as xs:string  
{  
    cts:search (fn:doc(), $terms), $search-options)  
};  
  
declare function search-advanced (  
    $terms as xs:string  
) as xs:string  
{  
    cts:search (fn:doc(), cts:word-query ($terms), $search-options)  
};  
  
xdmp:log ($search-options)
```

# Variable Renamed



The screenshot shows an IDE window with the following XQuery code:

```
xquery version '1.0-m1';  
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 IDE interface includes a project browser on the left showing a project named 'test-xquery-project' with subfolders 'hold' and 'src'. The 'src' folder contains files 'foo.xqy', 'foo2.xqy', and 'sample1.xqy'. The main editor displays the XQuery code, and the right sidebar contains various tool icons like 'Commander', 'Ant Build', 'IDEAalk', 'Maven Projects', 'Data Sources', and 'JetCradle'.

# 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](mailto:ron@overstory.co.uk)

<http://github.com/overstory>

# Questions?

Ron Hitchens

Principal Consultant, OverStory Ltd

[ron@overstory.co.uk](mailto:ron@overstory.co.uk)



Licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

© 2012 OverStory Ltd