XQuery
From The Trenches

MarkLogic User Group London 24/11/2015

Adam Retter

adam@evolvedbinary.com

@adamretter

EVOLVED BINARY

Adam Retter

- **Consultant**
  - Scala / Java
  - Concurrency
  - XQuery, XSLT
- **Open Source Hacker**
  - Predominantly NoSQL Database Internals
  - e.g. eXist, RocksDB, Shadoop (Hadoop M/R framework)
- **W3C Invited Expert for XQuery WG**
- **Author of the "eXist" book for O'Reilly**

EVOLVED BINARY

Talk Disclaimer

1. A work in progress...
2. How long this will take?
3. Real examples from working in a real team.
4. Experience dates from XQuery in MarkLogic 6.
5. My Opinions!
6. Maybe not even best practice!
7. Looking for interaction...

EVOLVED BINARY

A talk about code quality...

Trainwreck - woodleywonderworks (CC BY 2.0)

1. In General

EVOLVED BINARY

During Peer-Review of XQuery

1. Pragmatically Examine the Code (<u>and Tests</u>)
2. Spot Mistakes (Requirements and/or Typos)
3. Spot Bugs
4. DRY - expand common libraries, refactor, reduce LoC
5. <u>Promote Best Practice</u>
6. Redfine Best Practice doc
7. Constructive Criticism for Continuous Improvement

EVOLVED BINARY

Be Explicit

- **When reading code (linearly) I want to understand:**
  - The dependencies involved
  - The expected arguments of a function
  - The expected return type of a function
  - Ultimately <u>the intention of the author</u>

- **So when writing code I/we always:**
  - Declare module imports and namespaces in the prolog
  - Declare the types and cardinality of function arguments*
  - Declare the return type and cardinality of functions*
  - Use xqDoc *as nessecary* to document modules and functions

EVOLVED BINARY

Be Explicit

- **Explicit types and cardinality:**
  - Help with refactoring - static errors vs dynamic
  - Provide evidence to the documentation
  - Infer intention
  - Help us write <u>Unit</u> tests

EVOLVED BINARY

2. Portability

EVOLVED BINARY

Version Declaration

```
xquery version "1.0-ml";
```

ironic-fail - Dean Michael Dorman (CC BY 2.0)

EVOLVED BINARY

Version Declaration

```
xquery version "1.0-ml";
```

- **Immediately Breaks Portability**
  - Allows you to do non-standard things without realising!
  - Can you share your library?
  - Can you get the widest help available? e.g. StackOverflow
- **Do you actually need it?**
- **Don't be lazy, choose minimum**
  - Consider 1.0 first!
    - `xquery "1.0";`
  - Finally consider 3.0
    - `xquery "3.0";`

EVOLVED BINARY

Version Declaration

```
xquery version "1.0-ml";
```

- **When using standardised version**
  - You must import MarkLogic modules explicitly
    - Good Practice Anyway!

```
xquery version "1.0";

import module namespace cpf = "http://marklogic.com/cpf"
    at "/MarkLogic/cpf/cpf.xqy";

declare namespace cts = "http://marklogic.com/cts";
declare namespace xdmp = "http://marklogic.com/xdmp";
```

EVOLVED BINARY

Function Mapping

```
xquery version "1.0-ml";
```

- **If you must use 1.0-ml**
  - Make sure to disable function mapping

```
declare option xdmp:mapping "false";
```

- **Function Mapping Problems**
  - Causes implicit portability issues
  - Suddenly your code doesn't match the function docs!
  - Can lead to subtle and hard to spot bugs

```
declare variable $a := "Result1";
declare variable $b := "Result2";

declare function local:process($input as xs:string) {
  if($input)then $a else $b
};

local:process( () )
```

EVOLVED BINARY

Function Declarations

```
xquery version "1.0-ml";

declare function process() {
    <something/>
};

process()
```

- **If you must use 1.0-ml**
  - Make sure to declare the prefix of function declarations
  - Cannot be explicitly disabled, see: https://docs.marklogic.com/guide/xquery/enhanced#id_20838
  - Consider instead:

```
xquery version "1.0-ml";

declare function local:process() {
    <something/>
};

local:process()
```

EVOLVED BINARY

Namespace Axis

- **1.0-ml provides a "*Namespace Axis*"**
  - Originates from XPath 1.0
    - ...erm, but XQuery is based on XPath 2.0!
    - Mentioned briefly: https://docs.marklogic.com/guide/xquery/xpath#id_39877
  - Useful for copying source namespace when transforming a node:

```
xquery version "1.0-ml";

declare function local:create-example($entity as element()) as element() {
  element { fn:node-name($entity) } {
    $entity/@*,
    $entity/namespace::*,
    element other {
      text { "something" }
    }
  }
};
```

~~Namespace Axis~~
Namespace Constructor

- 3.0 provides a `Computed Namespace Constructor`
  - Example of copying source namespace when transforming a node:

```
xquery version "3.0";

declare function local:create-example($entity as element()) as element() {
  element { fn:node-name($entity) } {
    $entity/@*,
    fn:in-scope-prefixes($entity) !
        namespace {.} {fn:namespace-uri-for-prefix(., $entity)},
    element other {
      text { "something" }
    }
  }
};
```

EVOLVED BINARY

Map Data Types

- **1.0-ml provides the `map:map` data type**
  - It is non-portable
  - It is a mutable data type... and therefore EVIL!
    - See: The Evils of Mutable Variables
    - *DO NOT USE IT!* Unless you have a ***REALLY*** good reason...
      - "I need to return multiple values" -> Use XML!
      - "But, I need to preserve type info" -> Use Higher-order-functions instead!
      - "Offers better performance than *X*" -> Have you tested it? Is that actually the bottleneck?
- **3.1 introduces the `map(*)` data type**
  - Sadly still unsupported in MarkLogic 8

EVOLVED BINARY

xdmp:set

- **Not *strictly* a portability issue but a gateway to it**
  - Enables mutability
  - Encourages non-functional thinking and style
  - Trying to remove it from your code base causes all sorts of unexpected breakage
  - "So... er where or how did that value change?"
  - Advice: <u>DO NOT EVER USE IT!</u>

EVOLVED BINARY

xdmp:set

- I said, " **DO NOT EVER USE IT!**"

I'll Blow Your Fucking Brains Out AAAAAAAARRRRRGGGGHHHHHH
- Surian Soosay (CC BY 2.0)

EVOLVED BINARY

3. Smaller Code

EVOLVED BINARY

Simple Map Operator

- XQuery 3.0 (also 1.0-ml!)
- Can help improve code readability
  - Diligent Use
  - Reduce Boilerplate
  - If you liked ML's Function Mapping, consider this as an explicit alternative
  - Can be used when refactoring for DRY

```
xquery version "1.0";

for $animal in $animals/animal
return
    element { $animal/type } { $animal/name }
```

```
xquery version "3.0";

$animals/animal ! element { type } { name }
```

EVOLVED BINARY

Conditional Function Calls

- **Goal: DRY (Don't Repeat Yourself)**
  - Task: Refactor to reduce repeated code
  - Let's start with:

```
if($enable-cpf)then
  $triggers/trgr:trigger ! trgr:trigger-enable(trgr:trigger-name)
else
  $triggers/trgr:trigger ! trgr:trigger-disable(trgr:trigger-name)
```

  - How can we refactor this code?

EVOLVED BINARY

Conditional Function Calls

- **Refactor Attempt 1**
  - Merge context expressions of the Simple Map Operators
  - Resulting code:

```
$triggers/trgr:trigger/trgr:trigger-name !
    (
      if($enable-cpf)then
       trgr:trigger-enable(.)
      else
       trgr:trigger-disable(.)
    )
```

  - Is it cleaner?
  - How is the readability?
  - Can we refactor this code further?

EVOLVED BINARY

Conditional Function Calls

- **Refactor Attempt 2**
  - Reduce *if* expression to function invocation
    - Function references
    - xs:boolean -> xs:integer conversion
    - Dynamic function invocation
  - Resulting code:

```
$trigers/trgr:trigger/trgr:trigger-name !
    (trgr:trigger-disable#1, trgr:trigger-enable#1)
    [$enable-cpf cast as xs:integer + 1](.)
```

  - Is it cleaner?
  - How is the readability?
  - Can we refactor this code further?

EVOLVED BINARY

Function Call from Path Expr.

- **Path Expression may end with a function call**
  - An often overlooked feature of XPath
  - Function is evaluated once for each context item
  - Can often replace a FLWOR expression
  - Let's start with:

```
for $i in $some/xpath/expression
return
  local:some-function($i)
```

  - Rewrite to:

```
$some/xpath/expression/local:some-function(.)
```

  - Not the same as:

```
$some/xpath/local:some-function(expression)
```

EVOLVED BINARY

Namespaces

- **Declare Namespaces just *once* in the Module Prolog**
  - Do not declare inline on constructed or computed nodes
  - Reduces likelihood of typos and copy-paste mistakes
  - Let's start with:

```
declare namespace m = "http://sub.corp.dom.com/ns/proj/module";

declare function local:get-metadata() as element(m:metadata) {
    <metadata xmlns="http://sub.corp.dom.com/ns/proj/module">
        ...
    </metadata>
};

<metadata-container xmlns="http://sub.corp.dom.com/ns/proj/module">
{ local:get-metadata() }
</metadata-container>
```

EVOLVED BINARY

Namespaces

- **Declare Namespaces just *once* in the Module Prolog**
  - Rewrite to:

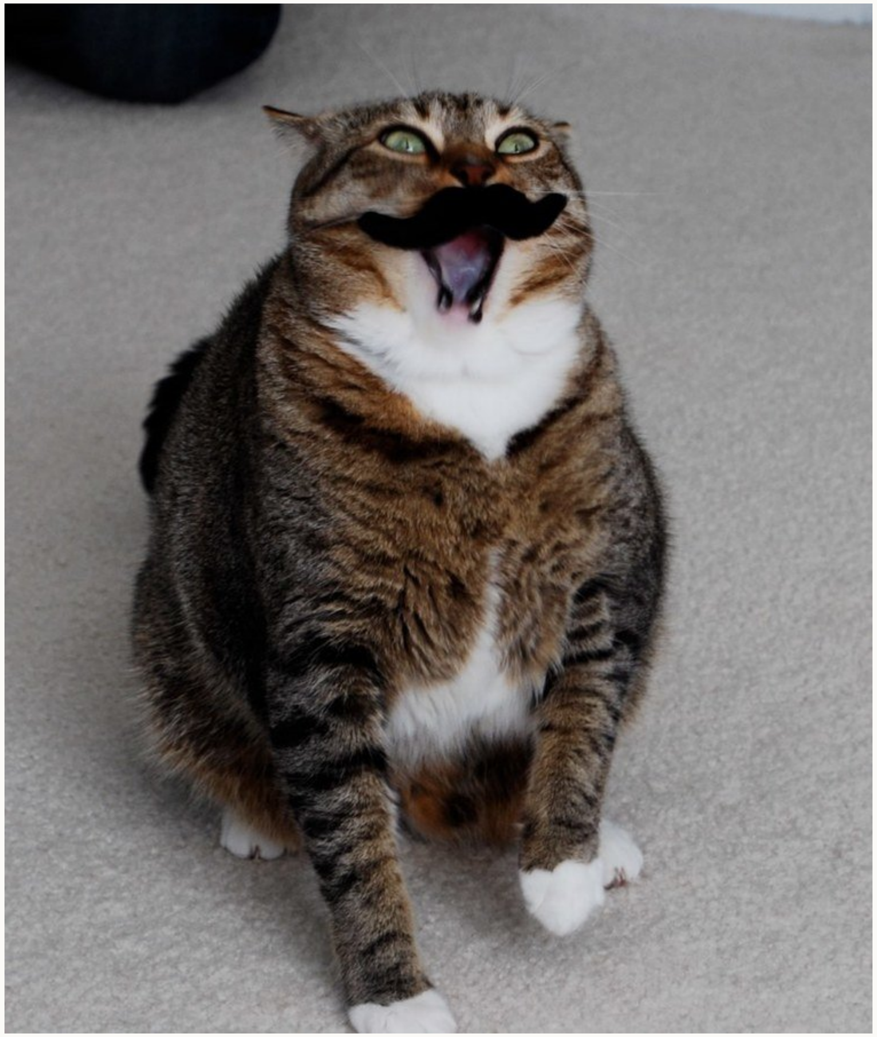
```
declare namespace m = "http://sub.corp.dom.com/ns/proj/module";

declare function local:get-metadata() as element(m:metadata) {
    <m:metadata>
        ...
    </m:metadata>
};

<m:metadata-container>
{ local:get-metadata() }
</m:metadata-container>
```

EVOLVED BINARY

xdmp:invoke / xdmp:eval


DSC_0335 copy - Jeff Rock (CC BY 2.0)

EVOLVED BINARY

xdmp:invoke / xdmp:eval

- **Dynamic evaluation of code**
  - Should only be used when:
    - Changing Database Context
    - Changing Transaction Context
  - For any other need, consider Higher Order Functions
    - Advantage of being statically checked

- **Prefer xdmp:invoke**
  - Seperate main module with injectable parameters
  - Code can be statically checked by linters etc.
  - Can make testing easier

EVOLVED BINARY

xdmp:invoke / xdmp:eval

- **How to pass a sequence as a parameter value?**
  - Consider its function signature:

```
xdmp:invoke(
  $path as xs:string,
  [$vars as item()*],
  [$options as node()?]
) as item()*
```

  - `$vars` " *This must be a sequence of even length, alternating QNames and items*"

- **Sequences of Sequences are flattened!**
  - Options:
    - Consider XML first
    - David Cassel - String Concatenation, see: http://blog.davidcassel.net/2010/01/passing-a-sequence-to-xdmpeval/

EVOLVED BINARY

xdmp:invoke / xdmp:eval

- **Higher Order Functions to the rescue!**
  - Passing a sequence as a parameter value:

```
xquery version "3.0";

declare namespace xdmp = "http://marklogic.com/xdmp";

xdmp:invoke(
  "http://example.com/foo.xqy",
  (xs:QName("local:param1"), function() { ("v1", "v2", "v3" ) }),
  <options xmlns="xdmp:eval">
    <isolation>different-transaction</isolation>
    <database>{xdmp:database("SOME-OTHER-DATABASE")}</database>
    <prevent-deadlocks>true</prevent-deadlocks>
  </options>
)
```

  - `foo.xqy`:

```
xquery version "3.0";

declare variable $local:param1 external;
count($local:param1())
```

EVOLVED BINARY

xdmp:invoke / xdmp:eval

- **Higher Order Function can also be a Closure!**
  - Passing a sequence (from the environment) as a parameter value:

```
xquery version "3.0";

declare namespace xdmp = "http://marklogic.com/xdmp";

let $my-values := ("v1", "v2", "v3")
return

  xdmp:invoke(
    "http://example.com/foo.xqy",
    (xs:QName("local:param1"), function() { $my-values }),
    <options xmlns="xdmp:eval">
      <isolation>different-transaction</isolation>
      <database>{xdmp:database("SOME-OTHER-DATABASE")}</database>
      <prevent-deadlocks>true</prevent-deadlocks>
    </options>
  )
```

EVOLVED BINARY

4. Safer Code

EVOLVED BINARY

Revisiting xdmp:eval

- **So... What is wrong with xdmp:eval?**
  - Consider its function signature:

```
xdmp:eval(
  $xquery as xs:string,
  [$vars as item()*],
  [$options as node()?]
) as item()*
```

  - Leads to:

```
xdmp:eval(
  "declare variable $local:param1 as xs:interger external;
<sum>{$local:param1 + 999}</sum>",
  (xs:QName("local:param1"), 10)
  <options xmlns="xdmp:eval">
    <isolation>different-transaction</isolation>
    <database>{xdmp:database("SOME-OTHER-DATABASE")}</database>
    <prevent-deadlocks>true</prevent-deadlocks>
  </options>
)
```

KaBoom!

Revisiting xdmp:eval

- **Fixing xdmp:eval with Higher Order Functions**
  - Refactored to this:

```
declare function local:sum($n as xs:integer) as element(sum) {
  <sum>{$n + 999}</sum>
};

xdmp:eval(
  "declare variable $local:f external;
$local:f()",
  (xs:QName("local:f"), function() { local:sum(10) }),
  <options xmlns="xdmp:eval">
    <isolation>different-transaction</isolation>
    <database>{xdmp:database("SOME-OTHER-DATABASE")}</database>
    <prevent-deadlocks>true</prevent-deadlocks>
  </options>
)
```

  - Code is now statically checked before main execution
  - Reduces nasty XQuery in Strings
  - Ensures a fail-fast approach

EVOLVED BINARY

A better xdmp:eval

```
declare function local:call-in(
    $database as xs:string?,
    $isolate as xs:boolean,
    $f) as item()*
{
  xdmp:eval(
    "declare variable $local:f external;
$local:f()",
    (xs:QName("local:f"), $f),
    <options xmlns="xdmp:eval">
      <isolation>{
        ("same-statement", "different-transaction")
            [$isolate cast as xs:integer + 1]
      }</isolation>
      <database>{
        (function() { xdmp:database($database) }, xdmp:database#0)
            [empty($database) cast as xs:integer + 1]()
      }</database>
      <prevent-deadlocks>true</prevent-deadlocks>
    </options>
  )
};
```

- Like a special version of `fn:apply` for ML ;-)

EVOLVED BINARY

5. Testable Code

EVOLVED BINARY

Writing Testable Code

- **The problem with (immutable) global state**
  - Can it be injected by the test runner?
  - What is my "*Unit*"?

- **The problem with side-effects**
  - Pre/Post-determined state affects test outcome
  - Test cannot run in isolation
  - Test(s) cannot be run in parallel
  - Possible dependencies on external systems (<u>Ouch</u>!)

- **Can we make our code more testable?**

EVOLVED BINARY

Writing Testable Code

- **Higher Order Functions to the rescue!**
  - By refactoring, we can (later) test with mock functions:

```
declare variable $local:uri-prefix external;

declare function local:store-base-data(
    $entities as document-node(element(entities)),
    $uri-prefix as xs:string,
    $store as function(element(entity)) as empty-sequence())
    as empty-sequence() {
  $entities/entities/entity[fn:starts-with(uri, $uri-prefix)] ! $store(.)
};

declare %private function local:store-entity(
    $entity as element(entity)) as empty-sequence() {
  xdmp:document-insert(
    $entity/uri,
    $entity/xml/*,
    $entity/permissions/*,
    $entity/collections/collection)
};

local:store-base-data($blah, $local:uri-prefix, local:store-entity#1)
```

Writing Testable Code

- **Injecting mock functions**
  - An XRay Unit Test:

```
declare
    %test:case
function local:store-base-data-limits-by-uri() {
  let $test-entities :=
      <entities>
        <entity>
          <uri>some/uri</uri>
        </entity>
      </entities>
  let $uri-prefix := "some"
  let $mock-store :=
      function($entities as element(entity)) as empty-sequence() {
        fn:error(
            $asset-lookup-error,
            "Should never be called when the uri-prefix is invalid",
            $entities)
      }
  return

    (: function under test :)
    local:store-base-data($test-entities, $uri-prefix, $mock-store)
};
```

n. And I could go on...

Cardiac Arrest (1) - 松林 L (CC BY 2.0)

EVOLVED BINARY

Don't fail me again - Xuilla (CC BY 2.0)

Any Questions?