

Portable EXPath Extension Functions

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**
- **XML Summer School Faculty (13/09/15)**



EVOLVED BINARY

A talk about incompatibility...



Phyllis Buchanan [+ Follow](#)

Playing with 3 sizes of lego

After trying to fit together 3 different incompatible sizes of lego Léon went into meltdown, not quite grasping the problem.

907
views

1
fave

3
comments

Taken on June 21, 2008

[Some rights reserved](#)

TODO...

1. The Portability Problem
2. Previous Efforts
3. Processor Varieties
4. Our Solution

Context

- **XPDL**

- XPath Derived Language e.g. XQuery/XSLT/XProc/XForms
- Typically uses F+O as Standard Library

- **Assumption: We want to write apps in XPDLs**

- Less code/impedance-mismatch
 - ~67% reduction in LoC vs Java ¹
- Serve/Process the Web
- Process structure/semi-structured data
- Process mixed-content

¹ Developing an Enterprise Web Application in XQuery
http://download.28msec.com/sausalito/technical_reading/enterprise_webapps.pdf

The Portability Problem

XPDLs are typically
specified as open
standards

...however...

Applications written in XPDLs
are rarely useable across
implementations



EVOLVED BINARY

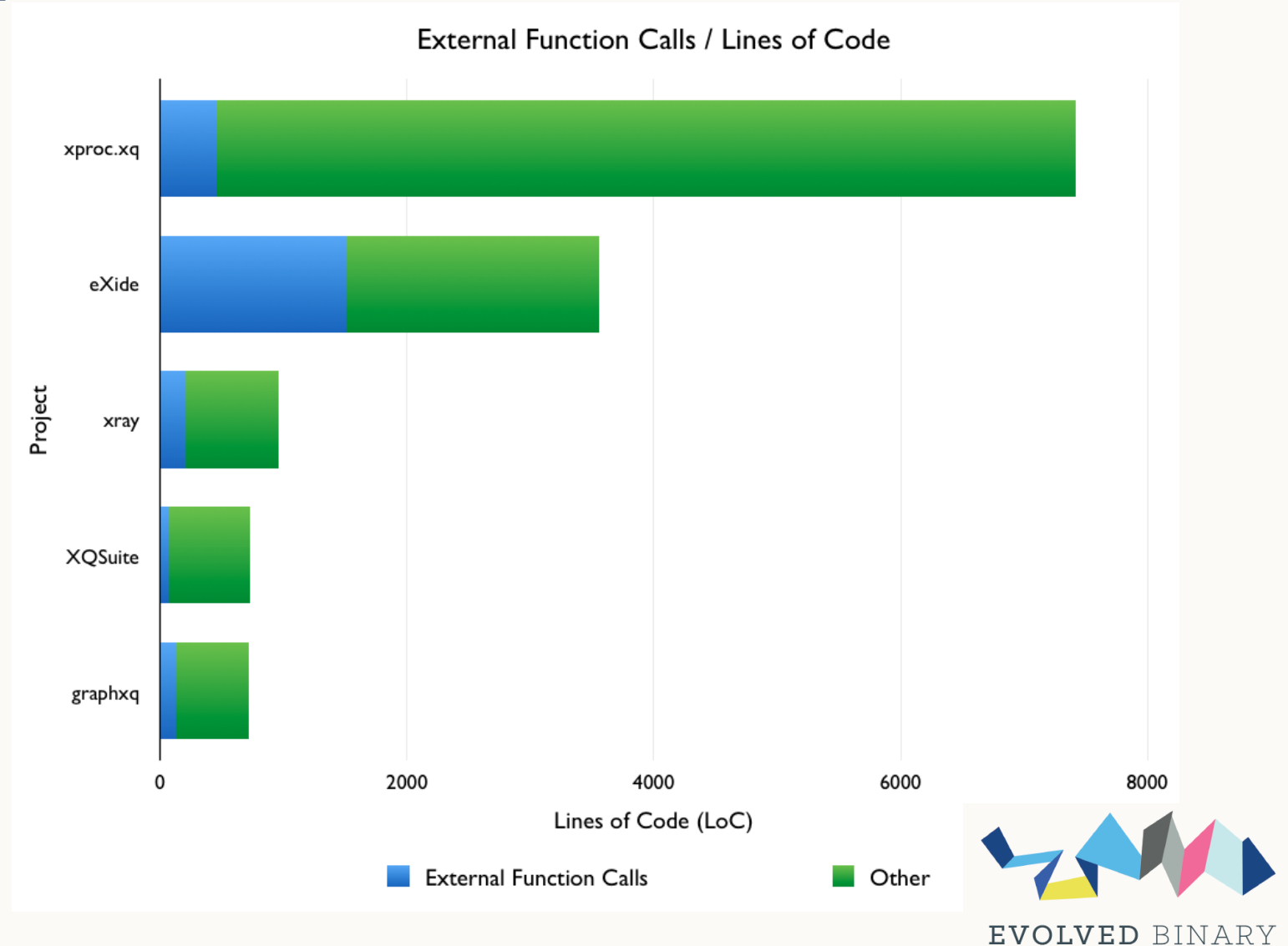
Vendor Extensions are EVIL!

- **Seem like a good idea at the time**
 - Easy/Quick to get something done
- **Many Types**
 - Syntax extensions
 - e.g `xquery "1.0-ml"`;
 - Data Type Extensions
 - e.g `xs:binary-document`
 - Deviation from Standards
 - e.g `fn:matches($input*, $pattern)`
 - Indexes, Triggers, etc.
 - Extension Functions

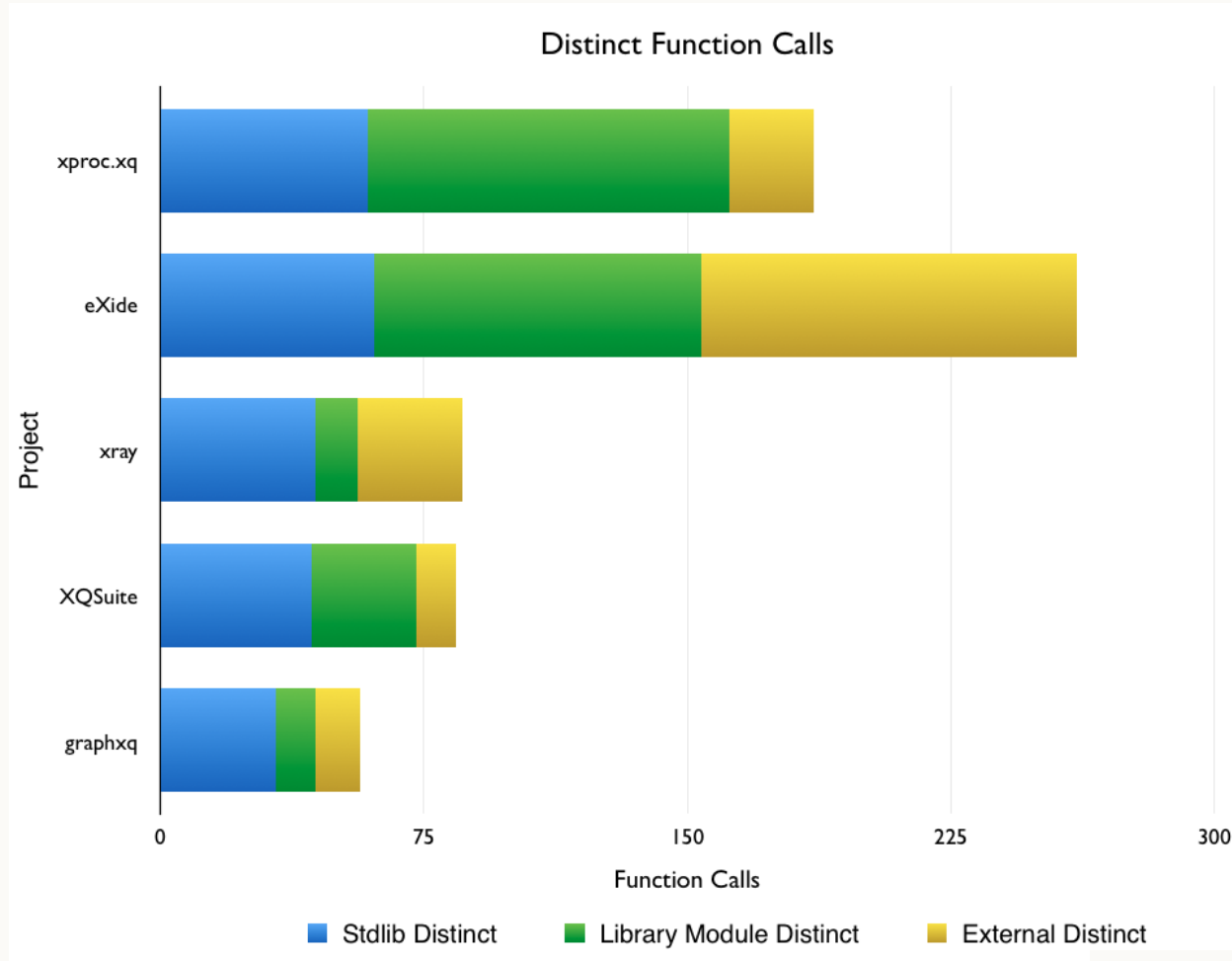
XPDL Extension Functions

- **Our focus, due to their impact**
 - Disguised by standard function call interface
 - `FunctionCall ::= EQName ArgumentList`
 - Distributed throughout an XPDL code-base
- **XPDL Extension Functions**
 - Typically implemented in lower-level language
 - C / C++ / Java / .NET etc.
 - Vendor/Processor specific
 - Consistent across processor versions?
 - EXPath
 - Requires reimplementaion for every processor
 - Not supported by all processors

Impact of Extension Functions



Impact of Extension Functions



Vendor Extensions ultimately:

- **Introduce Hurdles to Portability**
- **Restrict user freedom**
 - Vendor lock-in
 - Lesson the impact of frameworks
- **Fragment the XPD L community**
 - Create knowledge/skills silos
 - Reduce code-sharing
 - Limit code-reuse
 - Reduce collaboration
 - XPD L Processor specific forks of XPD L apps



Other Efforts to Improve Portability

- **XSLT 1.1 (2000)**

- Stated primary goal - "*improve stylesheet portability*"
- Adds `xsl:script` for extension functions
- Highly contentious. Abandoned!

- **EXSLT (2001)**

- Extended the XSLT 1.0 Standard Library
- Just a Specification
- Each vendor implemented for own processor

Other Efforts to Improve Portability

- **FunctX (2006)**
 - A Library of >150 useful common functions
 - Implementations in both XQuery and XSLT
- **EXQuery (2008)**
 - Just one specification to date: RESTXQ
 - Common implementation in Java
- **EXPath (2009)**
 - Standards for extension functions
 - Some common implementations in Java

Lessons Learnt

- **Standards are nice, but require implementations**
 - Really need >50% of market-share to implement
- **Vendors are lazy/limited**
 - Standards are often retrospective!
- **Implementation Type Mapping (XSLT 1.1)**
 - Showed great promise for integration
 - Must be implementation language agnostic
- **No single language for low-level implementation**
 - Won't be accepted by developers
 - Won't be accepted by vendors

Lessons Learnt

- XPD L Processors are surprisingly similar!



```
interface StandardFunc {  
    Item item(QueryContext qc, InputInfo ii) throws QueryException;  
}
```



```
interface BasicFunction {  
    Sequence eval(Sequence[] args, Sequence contextSequence)  
        throws XPathException;  
}
```



```
interface ExtensionFunctionCall {  
    SequenceIterator call(SequenceIterator[] arguments, XPathContext context)  
        throws XPathException;  
}
```

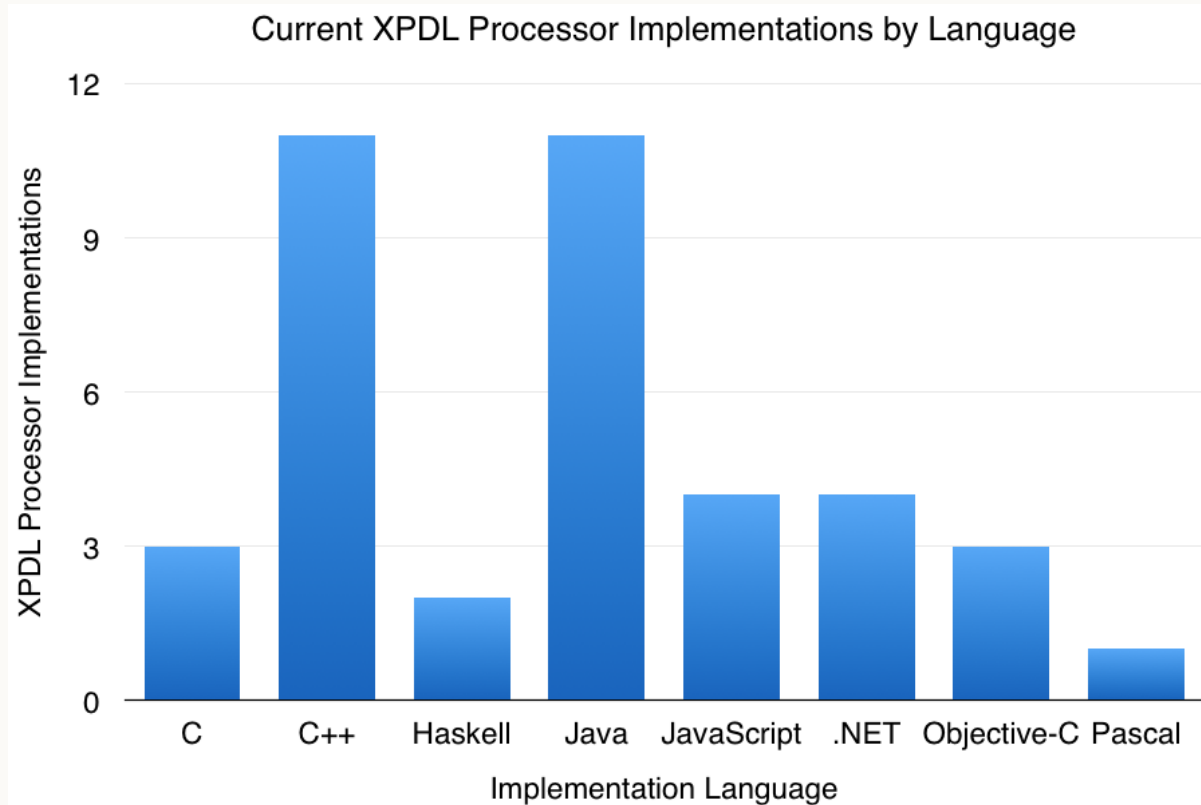


```
class XQFunction {  
    public:  
        Sequence createSequence(DynamicContext* context, int flags=0) const;  
};
```



Processor Varieties

- We want to support XPDL Extension Functions
 - For all XPDL processors
 - What XPDL processor implementations exist?



Our Requirements

- **Focus on Extension Function Implementation**
 - Standardisation is alive in W3C and EXPath
 - Ideally implement just once (ever!)
 - Ideally compatible with any XPDL processor
- **Polyglot**
 - Must support at least Java and C++ implementations
 - Ideally also C for integration with other languages
- **Specify an Implementation Type Mapping**
 - XDM types to/from XPDL processor implementation language types

Our Solution

- **Source-to-source Compilation**
 - Using the Haxe cross-platform toolkit
 - Haxe Lang for high-level implementation
 - Similar to ECMAScript
 - Haxe cross-compiler for target implementation
- **XDM Implementation Type Mapping to Haxe Lang Interfaces**
- **Function Implementation Type Mapping to Haxe Lang Interfaces**
 - Based on: XPath 3.0 *Function Call*
 - Based on: XQuery 3.0 *Function Declaration*



Haxe XDM Impl. Type Mapping

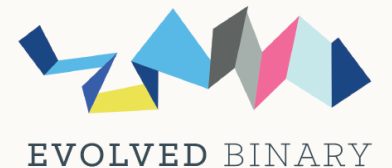
```
interface Item {
    public function stringValue() : xpdl.xdm.String;
}

interface AnyType {}

interface AnyAtomicType extends Item extends AnyType {}

class Boolean implements AnyAtomicType {
    var value: Bool;
    public function new(value) {
        this.value = value;
    }
    public function stringValue() {
        return new xpdl.xdm.String(Std.string(value));
    }
    public function haxe() {
        return value;
    }
}

class String implements AnyAtomicType {
    var value: HString;
    public function new(value) {
        this.value = value;
    }
    public function stringValue() {
        return this;
    }
    public function haxe() {
        return value;
    }
}
```



Haxe Function Implementation

Type Mapping

```
interface Function {  
  
    public function signature() : FunctionSignature;  
  
    public function eval(arguments: Array<Argument>, context: Context) : Sequence;  
}  
  
class FunctionSignature {  
    var name: QName;  
    var returnType: SequenceType;  
    var paramLists: Array<Array<Param>>;  
  
    public function new(name, returnType, paramLists) {  
        this.name = name;  
        this.returnType = returnType;  
        this.paramLists = paramLists;  
    }  
}
```

- <https://github.com/exquery/xpdl-extension-lib>



EVOLVED BINARY

Proof-of-concept

- **Implementation of EXPath File Module**

- Implemented in Haxe Lang
- Coded to XDM Implementation Type Mapping Interfaces

- **Focused on just `file:exists` function**

- `file:exists($path as xs:string) as xs:boolean`
- Function Call Type + Function Implementation Type
- `xs:string`
- `xs:boolean`

- **Status**

- Runnable on **any** processor that supports Haxe Implementation Type Mapping



file:exists in Haxe

```
class ExistsFunction implements Function {

    private static var sig = new FunctionSignature(
        new QName("exists", FileModule.NAMESPACE, FileModule.PREFIX),
        new SequenceType(Some(new ItemOccurrence(Boolean))),
        [
            [
                new Param(new QName("path"),
                    new SequenceType(Some(new ItemOccurrence(xpdl.xdm.Item.String))))
            ]
        ]
    );

    public function new() {}

    public function signature() {
        return sig;
    }

    public function eval(arguments : Array<Argument>, context: Context) {
        var path = arguments[0].getArgument().iterator().next().stringValue().haxe();
        var exists = FileSystem.exists(path);
        return new ArraySequence( [ new Boolean(exists) ] );
    }
}
```



Proof-of-concept: Processor

- **Added support to eXist**

- Static mapping of Haxe XDM types
- Dynamic mapping of Haxe function call interfaces
 - Bytecode generation of classes and objects: cglib
- Currently ~300 lines of Java code

- **Status**

- <https://github.com/eXist-db/exist/tree/xpdl-extensions/src/org/exist/xpdl>
- Supports Haxe XDM Function Implementation Type Mapping
- Supports Haxe XDM Implementation Type Mapping



Conclusion

- **Implement Once**
- **Cross-Compile and Compile Once**
- **Supports any processor**
 - Requires Vendor to (*just once*) implement:
 - XDM Implementation Type Mapping
 - Function Implementation Type Mapping

Win!

