EXPath Packaging

A framework to package libraries and applications for core XML technologies

Balisage, August 4th, 2010
Montréal

Florent Georges
H2O Consulting

H2O Consulting

EXPath Packaging

- **Introduction** ←
- The problem
- How to use it?
- Write a package
- A project structure
- Going further
- Conclusion

H2O Consulting

Introduction - History

- EXSLT for XSLT 1.0
- XSLT 2.0 and needs for new extensions
- EXSLT 2.0, EXQuery & EXProc
- XML Prague 2009 – EXPath
- First modules – HTTP Client & ZIP Facility
- Summer 2009 – the Packaging System
- 2010 – the Webapp module

Introduction - Goals

- *Collaboratively defining open standards for portable XPath extensions*
- The main means is extension functions
- The main goal is defining portable specifications...
- ...and convincing vendors to endorse them
- But also providing support to open-source implementations

H2O Consulting

Introduction - Processes

- More or less formal, more or less informal *(that is a feature, not a bug)*
- The definitive goal is writing specifications
- The main tool is the mailing list
- Each module has one main maintainer, responsible of editing & achieving consensus
- More infos about processes on the wiki

EXPath Packaging

- Introduction
- **The problem** ←
- How to use it?
- Write a package
- A project structure
- Going further
- Conclusion

H2O Consulting

The import problem

- The way to import a module is dependent on the processor
- XSLT import URI
- XQuery evil: location hint
- For now, there is no standard way to import a module in XSLT, XQuery nor XProc
- No other modern programming language as this limitation

The import problem

```
(: in Saxon :)

import module namespace functx = "http://www.functx.com"
  at "../../../../../../xlibs/functx/src/functx.xq";

declare function local:hello($who as xs:string) as xs:string
{
  concat('Hello, ', functx:capitalize-first($who), '!')
};
...


(: in eXist :)

import module namespace functx = "http://www.functx.com"
  at "xmldb:exist:///db/modules/functx.xq";
...
```

The import problem

- The ideal solution would be to get rid of the location hint, and see the import URI as a name

```
(: portable :)

import module namespace functx = "http://www.functx.com";

declare function local:hello($who as xs:string) as xs:string
{
  concat('Hello, ', functx:capitalize-first($who), '!')
};
...
```

- Achievable somehow through XML Catalogs, but the install process is not uniform and thus even more painful for the user

XML Catalogs

- XML Catalogs are in the correct direction, but need automatization
- Both for the final user and for the author
- The solution needs to be used consistently, XML Catalogs does not give enough info
- Even when a catalog is shipped with a library, it needs advanced config in order to work
- And there is no standard release structure
- URI resolving is only part of the solution

H2O Consulting

EXPath Packaging

- Introduction
- The problem
- **How to use it?** ←
- Write a package
- A project structure
- Going further
- Conclusion

H2O Consulting

How to use it?

- A library user has two things to do:
  - install the package, using an automatic installer
  - import it and use it, of course, by simply using the pubic URI
- Installers can be command line tools, or a web form, or whatever is provided by the processor
- By using the public URIs (and only the public URIs), the written code is portable across different processors

H2O Consulting

Installers

- Command-line tool:

```
> xrepo list
google-apis-0.1

> xrepo install functx-1.0.xar

> xrepo list
functx-1.0
google-apis-0.1
```

- eXist's web-based install:

http://localhost:8080/exist/repo/repo.xml

- Functx-1.0 **install** | **remove**
- XQuery-Json-0.1 **install** | **remove**
- FXSL-1.0 **install** | **remove**
- example app Invoice-1.0 **install** | **remove**

H2O Consulting

Import modules

- Going back to our example:

```
(: portable :)

import module namespace functx = "http://www.functx.com";

declare function local:hello($who as xs:string) as xs:string
{
  concat('Hello, ', functx:capitalize-first($who), '!')
};
...
```

- It is now portable across processors, without imposing any configuration burden on the user

H2O Consulting

EXPath Packaging

- Introduction
- The problem
- How to use it?
- **Write a package** ←
- A project structure
- Going further
- Conclusion

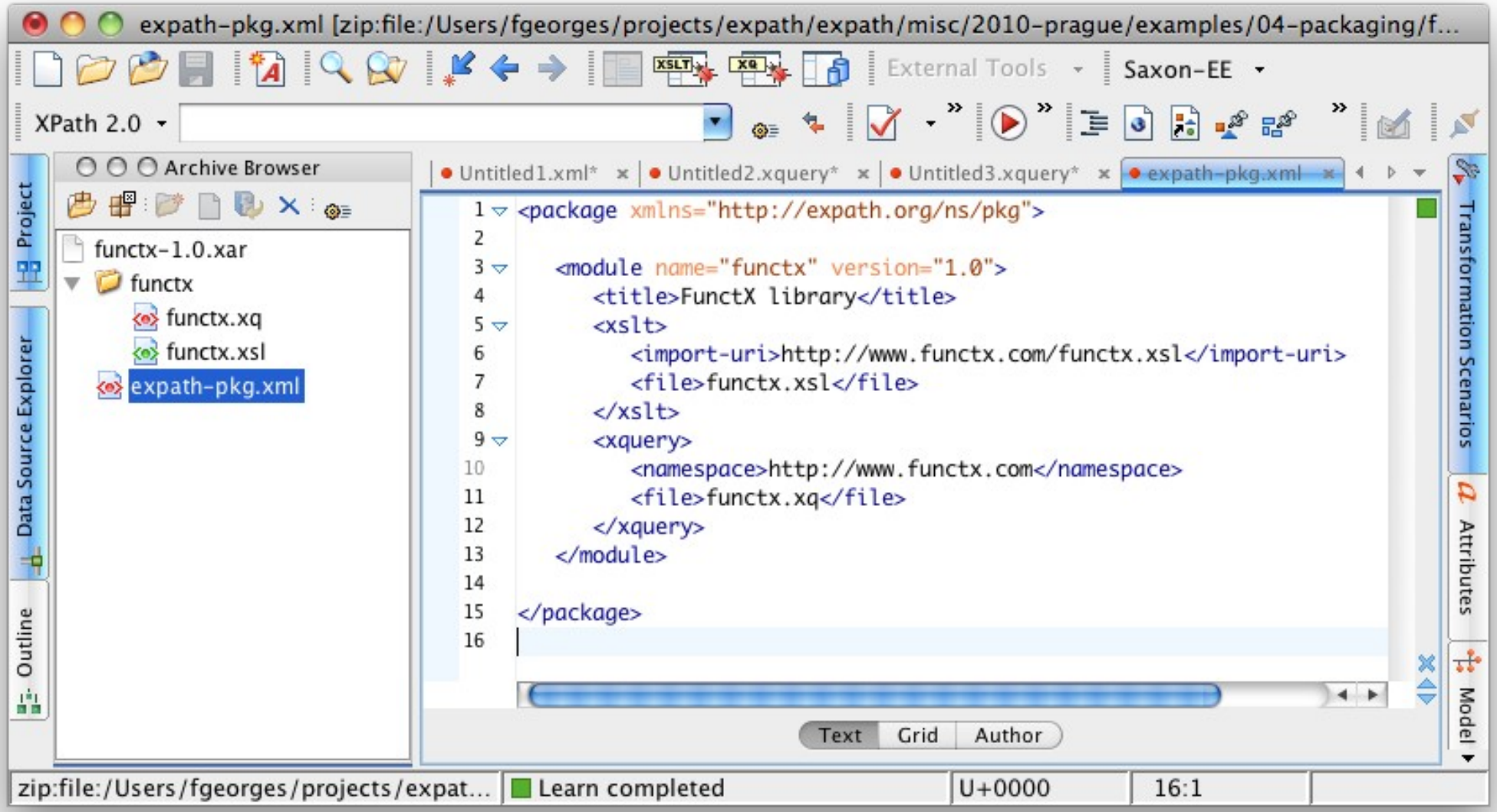
H2O Consulting

Requirements

- Building upon and going behind XML Catalogs, a packaging format must:
  - describe what is needed but is not in the X* specifications
  - be understood by most processors
  - package the components and additional informations in a single file
  - be eXtensible (to allow other specs to build upon it, and allow processor specific infos)
- Installation process can then be automated

H2O Consulting

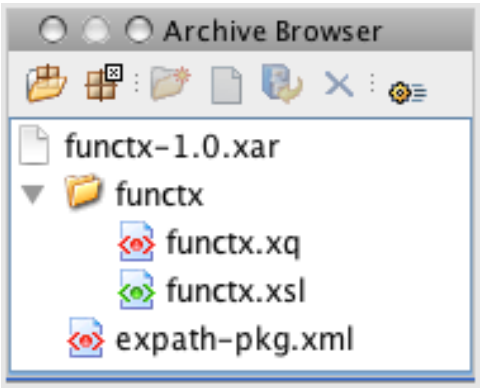
Overview



H2O Consulting

Structure

- A package is a ZIP file
- It contains exactly one subdirectory (content)
- It contains a package descriptor: `expath-pkg.xml`
- It can contain per-processor descriptors
- It can contain descriptors for other specs



The descriptor

- Record some meta infos about the package
- Record the content component's public URIs

expath-pkg.xml

```
1  <package xmlns="http://expath.org/ns/pkg">
2
3     <module name="functx" version="1.0">
4        <title>FunctX library</title>
5        <xslt>
6           <import-uri>http://www.functx.com/functx.xsl</import-uri>
7           <file>functx.xsl</file>
8        </xslt>
9        <xquery>
10          <namespace>http://www.functx.com</namespace>
11          <file>functx.xq</file>
12       </xquery>
13    </module>
14
15 </package>
```

Putting it together

- The components and the descriptor are just zipped together to make the XAR package
- The XAR file must respect the structure described in the descriptor
- Any ZIP tool can be used to achieve this goal

```
> mkdir balisage-hello

> cp hello.xql balisage-hello

> zip -r balisage-hello-1.0.xar expath-pkg.xml balisage-hello
  adding: expath-pkg.xml (deflated 39%)
  adding: balisage-hello/ (stored 0%)
  adding: balisage-hello/hello.xql (deflated 33%)
```

Standard repository layout

- How packages are installed is implementation-dependent
- The spec defines an optional repository layout
- If the implementation adopts this layout, it can share repositories with others
- Management tools in the command line have been built to manage such repositories (install, remove, list packages)
- A Java library exists for the URI resolution

EXPath Packaging

- Introduction
- The problem
- How to use it?
- Write a package
- **A project structure** ←
- Going further
- Conclusion

H2O Consulting

Too much for me!

- That's fine, but as a library author, that seems a lot of work to do again and again
- Besides, most of the steps are similar every time
- Creating new libraries usually involve copying and pasting Makefiles or other build files, and adapting them, over and over again
- By using some conventions, we can actually automate those repetitive tasks

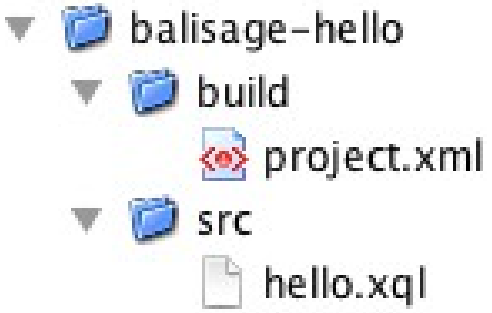
Project?

- Following a consistent structure, a project can be built automatically
- This structure use naming conventions for directories
- As well as a project descriptor for meta data (title, version, etc.)
- The public URIs are maintained within the components themselves
- An XSLT stylesheet packages the project

H2O Consulting

Project!

- The basic project structure has a `build/` directory with a project file, and a `src/` directory with the project source files



xproj

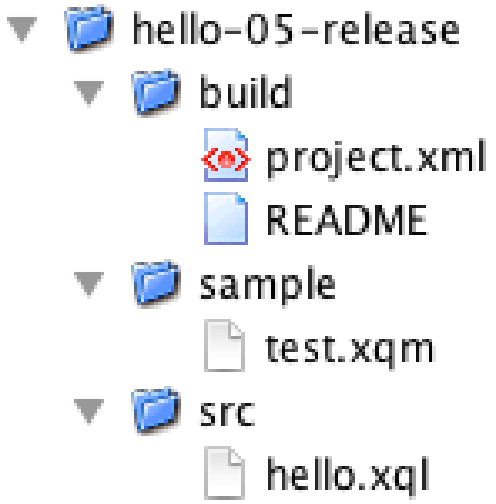
- A simple script wraps the stylesheet invocation
- You call it from the project directory to build the project package:

```
> pwd
~/2010-balisage/examples/balisage-hello/

> xproj build
Building project ~/2010-balisage/examples/balisage-hello/
Generated XAR file build/balisage-hello-1.0.xar

> ls build
balisage-hello-1.0.xar
packager.xml
```

H2O Consulting

Releasing

- Why did we put the test file outside the project? Let's include it.
- And let's put a nice README file

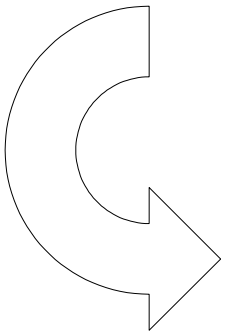


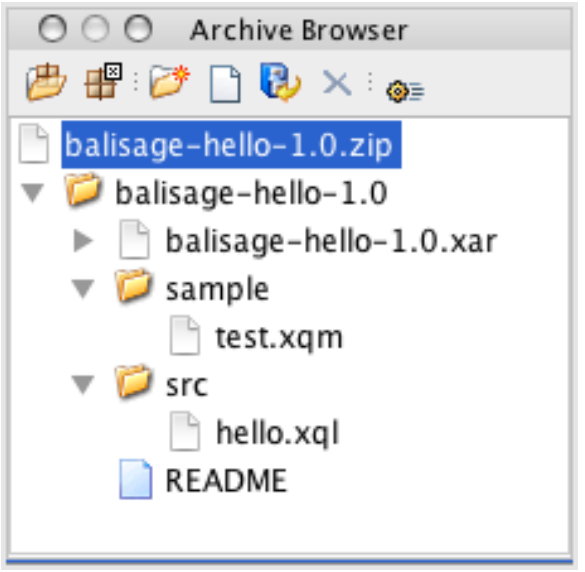
- Can we create automatically a proper release?

H2O Consulting

xproj, relaunched

```
> xproj build
Building project ~/2010-balisage/examples/balisage-hello/
Generated XAR file build/balisage-hello-1.0.xar

> xproj release
Releasing project ~/2010-balisage/examples/balisage-hello/
Generated ZIP release file build/balisage-hello-1.0.zip
```





(let's make it clear)

- Before going further, let's make it clear this project structure stuff is already behind the packaging system itself
- It is useful, or even crucial, for the user experience
- But it is behind the packaging spec
- The spec is the minimal common piece to conform to
- Tools and specs can then be built upon it

EXPath Packaging

- Introduction
- The problem
- How to use it?
- Write a package
- A project structure
- **Going further**
- Conclusion

←

H2O Consulting

Repository functions

- A set of functions to manage a package repository directly from within XPath expression
- Not part of the spec (but could be in a v.Next)
- This is the approach followed by eXist:
  - repo:list() as xs:string*
  - repo:install($href as xs:string) as xs:boolean
  - repo:remove($name as xs:string) as xs:boolean
- Could be used to build convenient managers, not dependent on the processor

CXAN

- http://cxan.org/
- Comprehensive XML Archive Network
- Follow the same principle as well-known CPAN for Perl and CTAN for (La)TeX
- Work in progress
- Collect existing packages
- Make them accessible and searchable from a single one location

H2O Consulting

CXAN

- CXAN is composed of:
  - the package base set on the server
  - the website to browse and search within this package set
  - a command line tool to install packages by downloading them directly from the server

H2O Consulting

CXAN

```
> cxan install functx
Downloading functx from http://cxan.org/xar/functx/latest...
Going to install functx...
```

- Some challenges are still to solve:
  - versioning
  - dependencies between packages
  - website interface

H2O Consulting

Webapps

- Using X* technologies end-to-end for web applications
- Most existing XML databases provide proprietary framework for that (eXist, MarkLogic, Sausalito, etc.)
- Then again, we are stuck with processor-locked applications
- A standard would allow to write portable web applications, libraries and frameworks

Request / response

```
<web:request servlet="name" path="/path" method="get">
  <web:uri>http://example.org/my-app/path/one?p=v</web:uri>
  <web:authority>http://example.org</web:authority>
  <web:context-root>/my-app</web:context-root>
  <web:path>
    <web:part>path/</web:part>
    <web:match name="which">one</web:match>
  </web:path>
  <web:param name="p" value="v"/>
  <web:header name="connection" value="keep-alive"/>
  ...
</web:request>

<web:response status="200" message="Ok">
  <web:header name="..." value="..."/>
  ...
  <web:body content-type="text/html" method="xhtml"/>
</http:response>
```

Entry point

- Either a:
  - XQuery function
  - main XQuery module
  - XSLT function
  - XSLT named template
  - XSLT stylesheet
  - XProc pipeline
- Must accept two parameters
  - the request element
  - the sequence of bodies (possibly empty)

Entry point

```
declare function my:home-servlet(
    $request as element(web:request),
    $bodies  as item()*
  )
  as element()+
{
   <web:response status="200" message="Ok">
      <web:body content-type="text/html" method="xhtml"/>
   </web:response>
   ,
  <html xmlns="http://www.w3.org/1999/xhtml">
     ...
  </html>
};
```

H2O Consulting

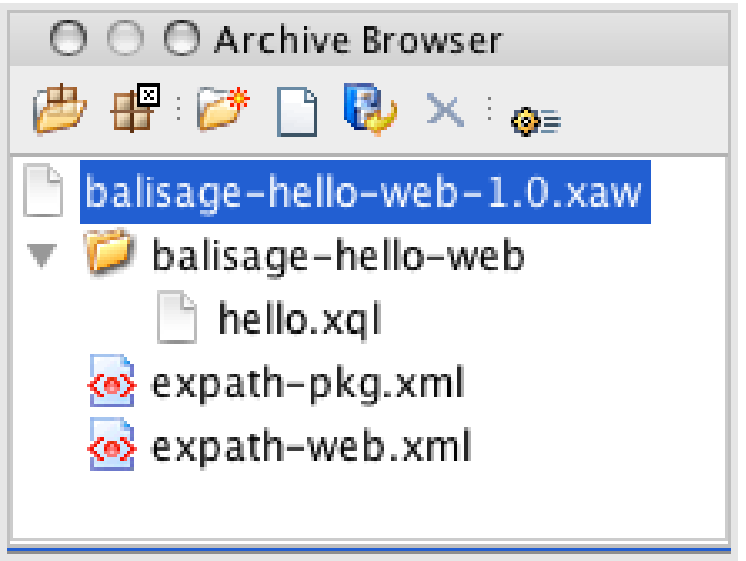
Web descriptor

- Map requests to entry points
- Based on URI matching

```
<webapp xmlns="http://expath.org/ns/webapp/descriptor"
        xmlns:h="http://fgeorges.org/tmp/hello"
        name="http://fgeorges.org/tmp/hello"
        abbrev="balisage-hello-web"
        version="1.0">

   <title>Balisage sample webapp</title>

   <servlet name="home">
      <xquery function="h:home-servlet"/>
      <url pattern="/"/>
   </servlet>

   <servlet name="hello">
      <xslt function="h:hello-servlet">
         <import-uri>http://fgeorges.org/tmp/hello.xsl</import-uri>
      </xslt>
      <url pattern="/hello"/>
   </servlet>

</webapp>
```

Packaging

- A webapp is packaged as any standard project
- The web descriptor is inserted next to the package descriptor
- All the resolution mechanism is already there



Building block

- Once again, the webapp spec follow the same principle than packaging: defining only the strict minimum low-level mapping between an HTTP request and an X* component (and its response back to HTTP)
-

EXPath Packaging

- Introduction
- The problem
- How to use it?
- Write a package
- A project structure
- Going further
- **Conclusion**

←

H2O Consulting

- Join the mailing list and browse the website:

http://expath.org/