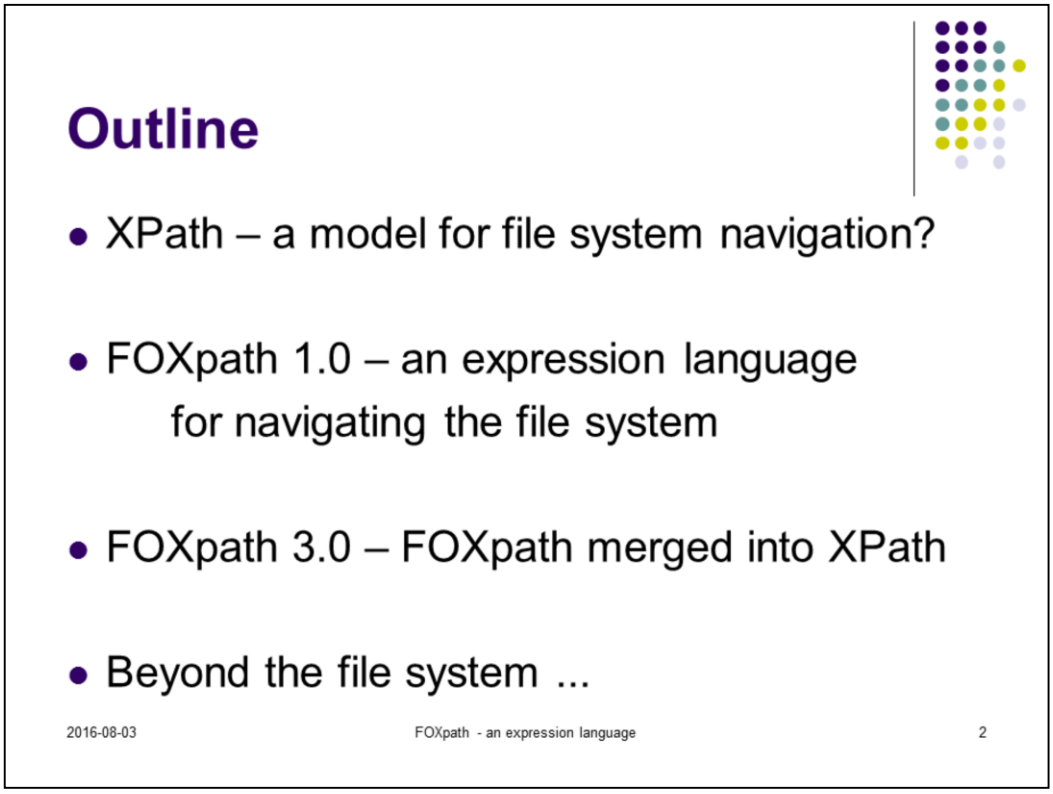
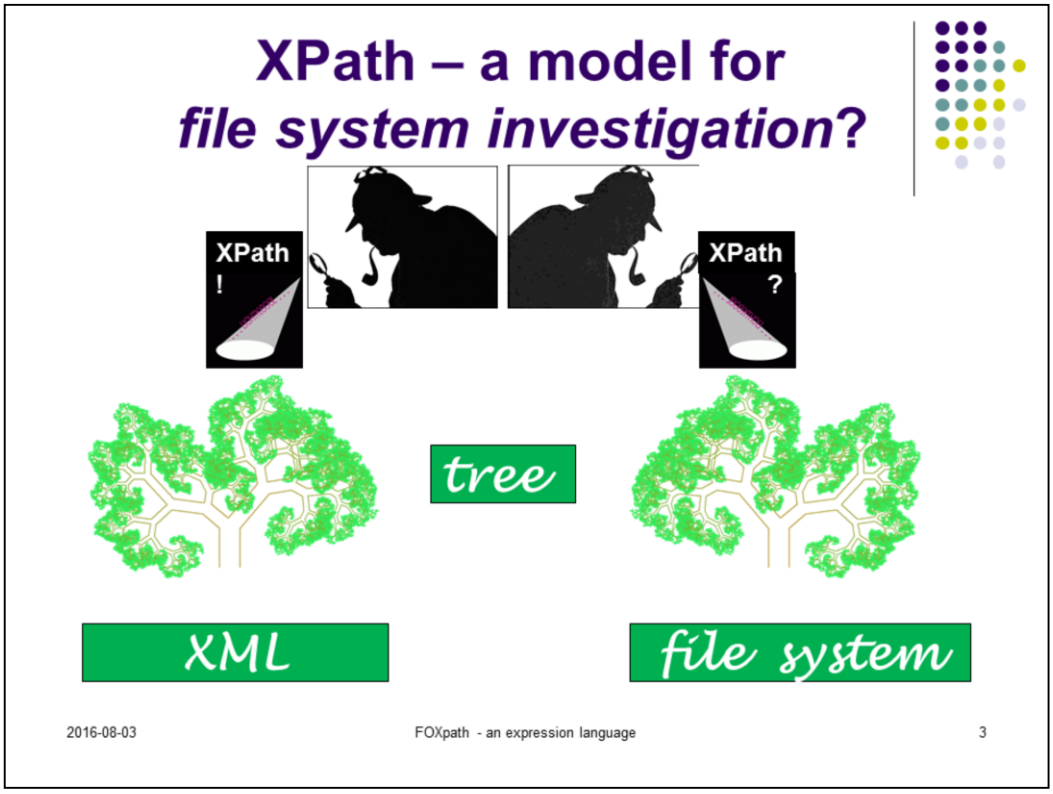
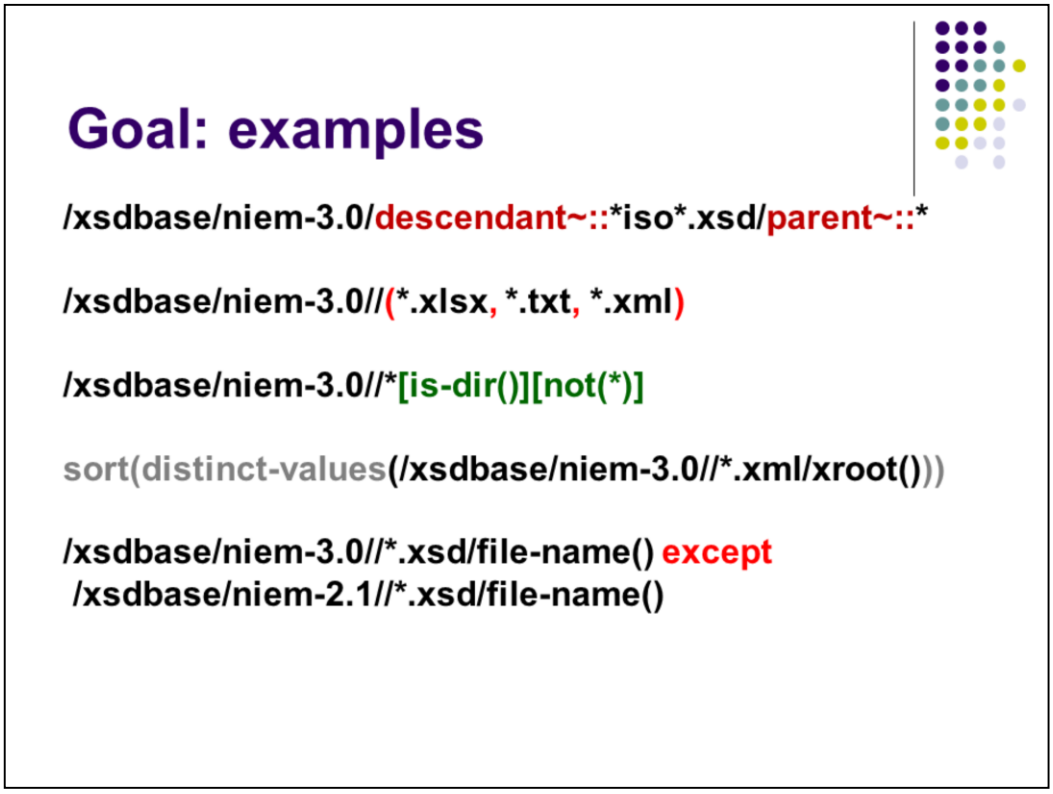
XPath is an expression language for navigating the contents of XML documents. Using XPath, we can select contents in an elegant, concise and very intuitive way. What is more: XPath is not a loose collection of expressions and rules, but a complete and fully composable expression language. Therefore there is virtually no limit to the selectiveness we may achieve, constructing XPath expressions. XML documents and a file system have much in common – they both expose a tree-structured collection of names. So – should we not have something like „XPath for files and folders“ – FOXpath for short? This presentation reports my efforts to design and implement such an expression language.



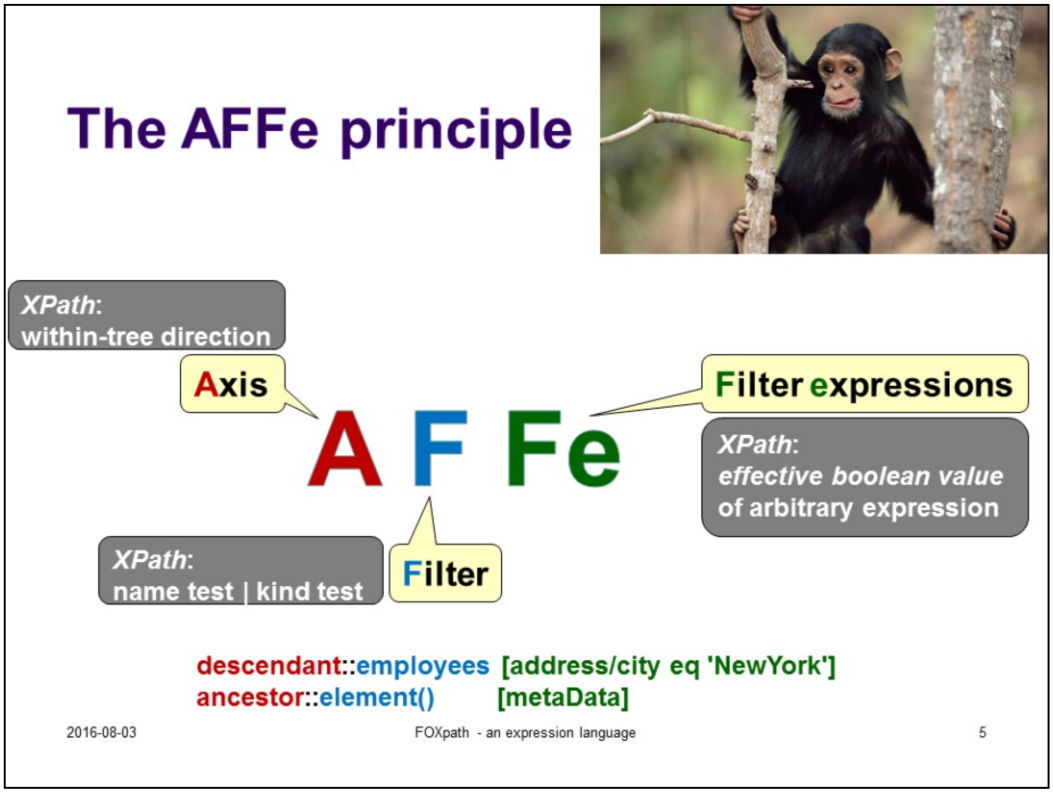
First, we check the basic assumption that XPath is an adequate model for file system navigation. Then we explore FOXpath, a new expression language for navigating the file system. We shall see that this first version of FOXpath is a modified copy of XPath. But then we merge FOXpath back into XPath, obtaining an extended version of XPath supporting file system navigation. And finally we shall see that FOXpath is not restricted to physical file systems, but can navigate logical file systems as well, as for example used to organize the contents of a NOSQL database, version control repositories or the URI collection exposed by REST-ful web services.



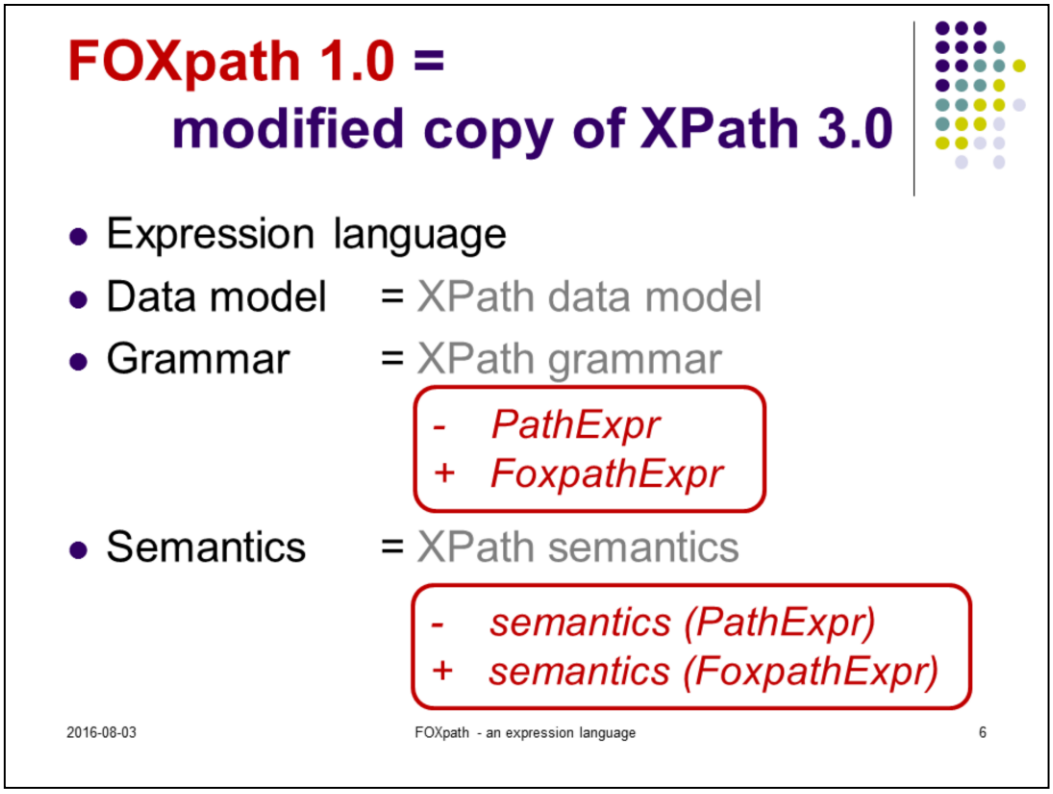
XPath îs a device for exploring and navigating XML documents. A closer look at XPath reveals core concepts not based on XML, but on the general notion of a tree-structured collection of names. XML is a particular instance of that notion, and the file system is another instance. We conclude that it should be possible to design an expression language which is similar to XPath, but deals with files and folders, rather than XML nodes.



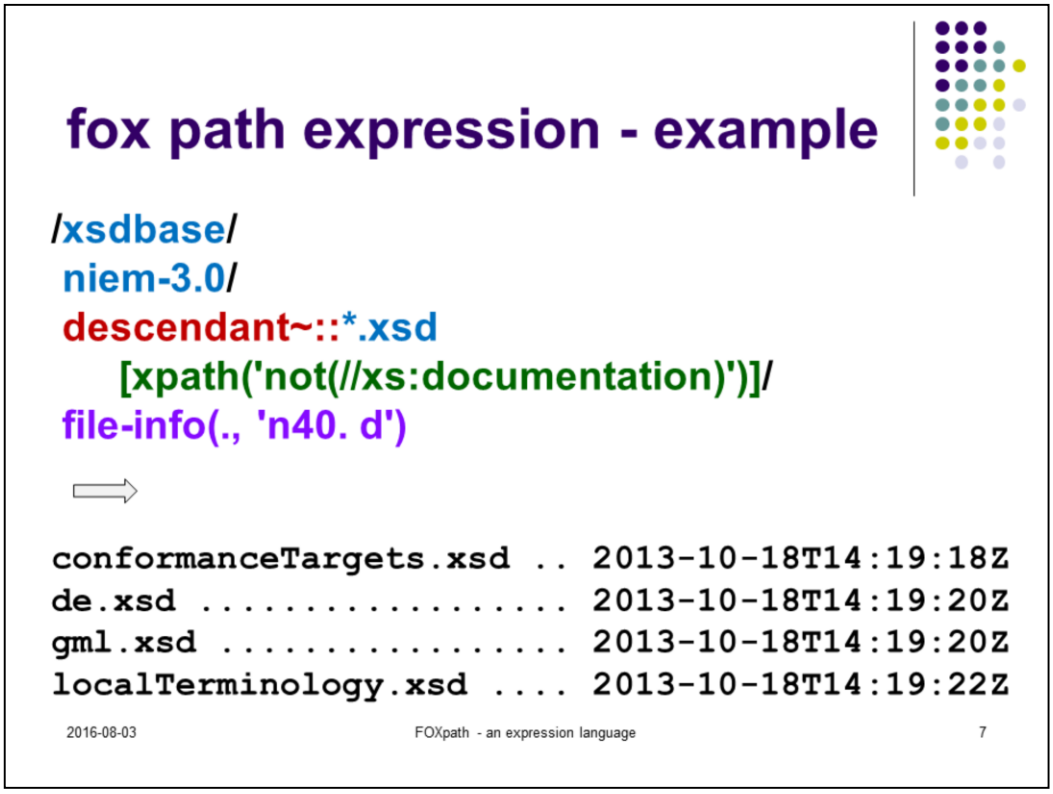
Let us start with a few examples illustrating the desired expression language. They look like XPath, and they are as powerful as XPath expressions.



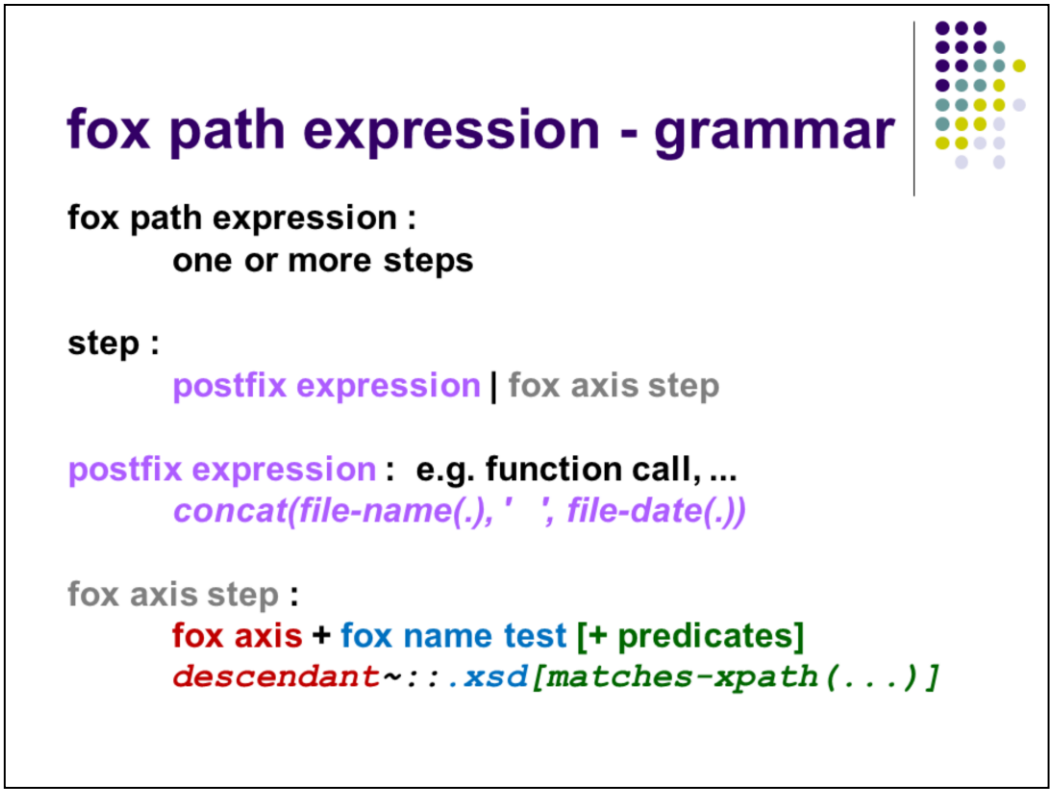
If we want to use XPath beyond XML, we must understand the core principles of XPath in an abstract way, which is independent of XML. „Affe“ is the German word for monkey, and most monkeys love to traverse trees. It‘s natural to summarize the core principles of XPath under the term AFFe. It‘s an acronym referring to the building blocks of selection: axis + filter + filter expression. The axis collects everything visited when traversing the tree in a particular direction (children, descendants, parent, ancestors, and so forth). Collected items are submitted to a simple filter, which in the case of XPath is either a name test or a kind test. If this filtering is still too crude, further filtering of unlimited selectiveness can be achieved by adding filter expressions.



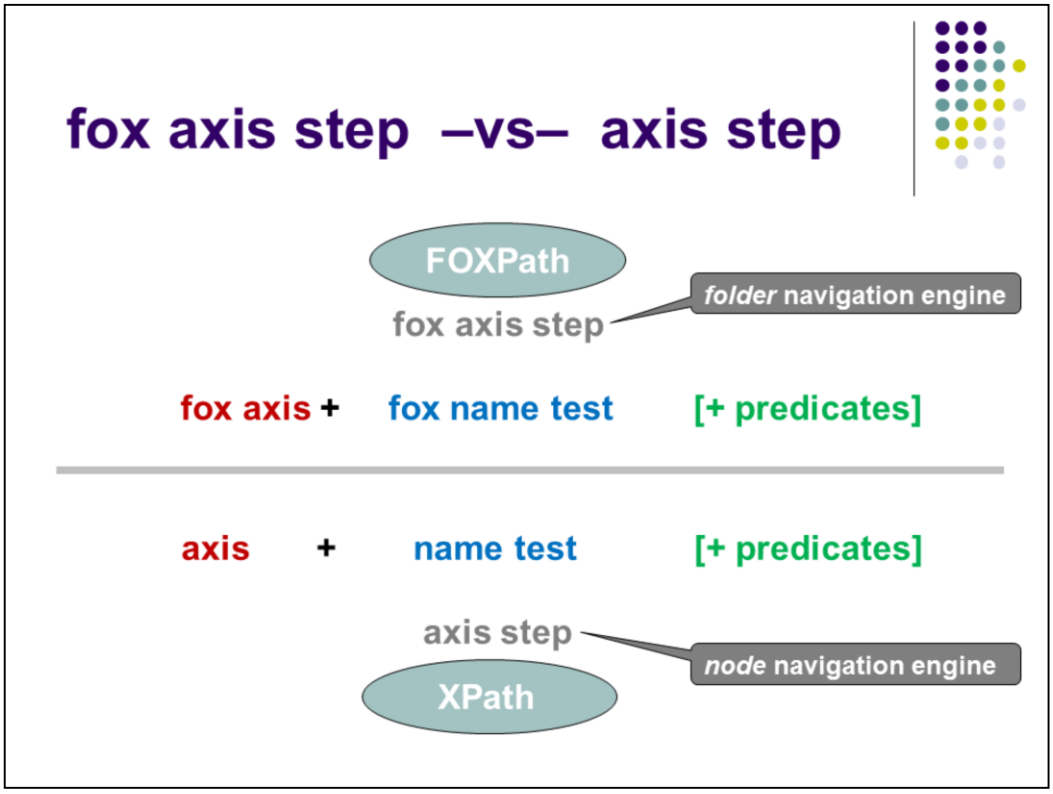
XPath is a masterly elaboration of the AFFe principle. I never cease to marvel at the expressiveness achieved by uncomprising rigour and consistency. Aspiring to a new „XPath for files and folders“, what would be more promising than to stick as closely to XPath as possible?



So the fox path expression is the only difference between XPath and FOXpath! Before becoming formal, let us look at an illustrative example. The expression reports all Niem XSDs without documentation. Ignoring the tilde and the use of wildcards within names, this could be an XPath expression.

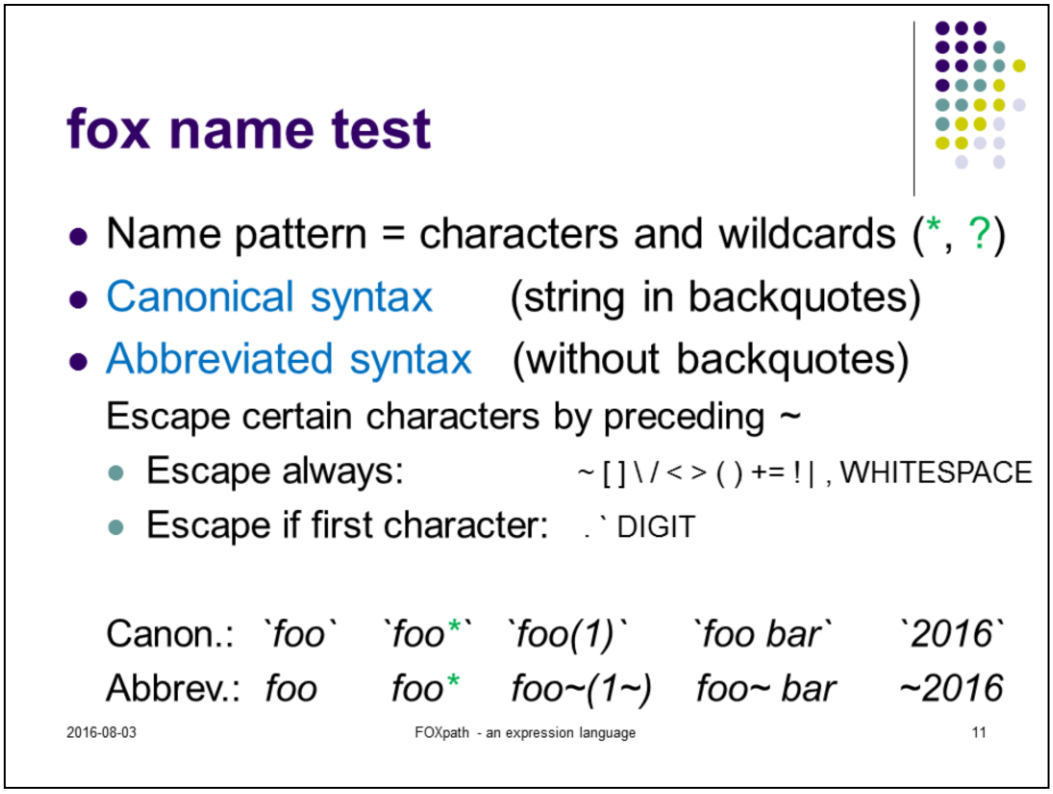


That‘s because the grammar of the fox path expression is just a modified copy of the grammar of the path expression.

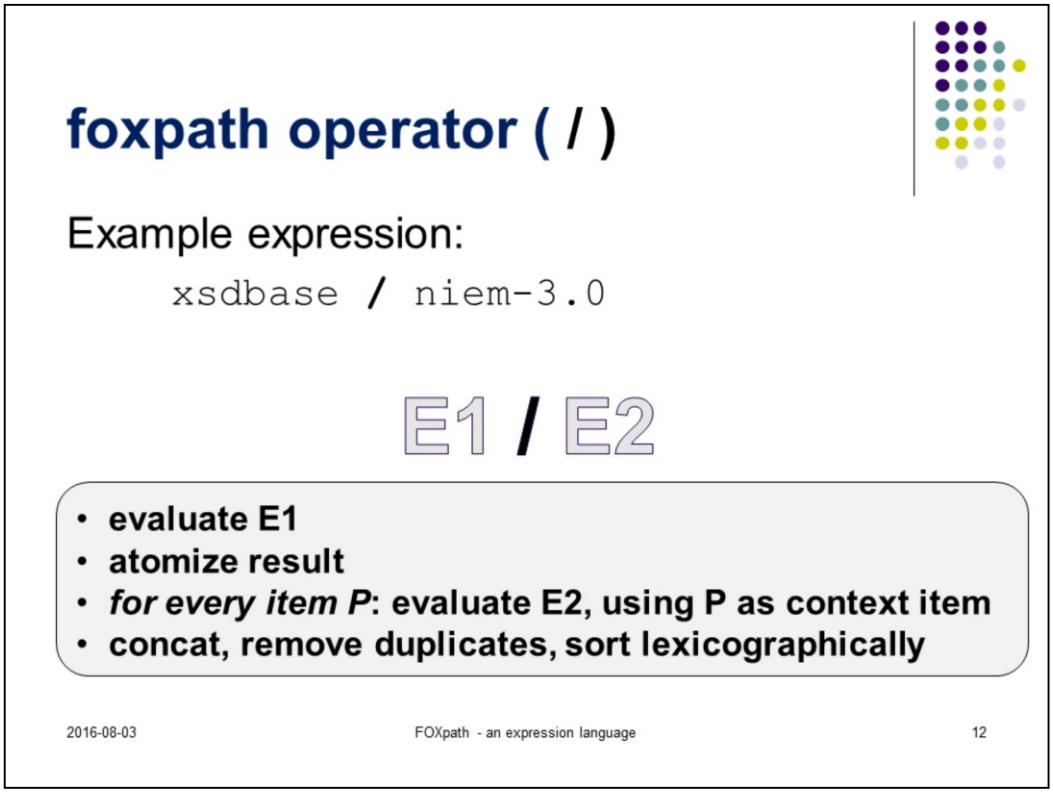


So the core functionality of folder nagivation, which is the fox axis step, is a modified copy of the axis step of XPath, which is the core functionality of node navigation..

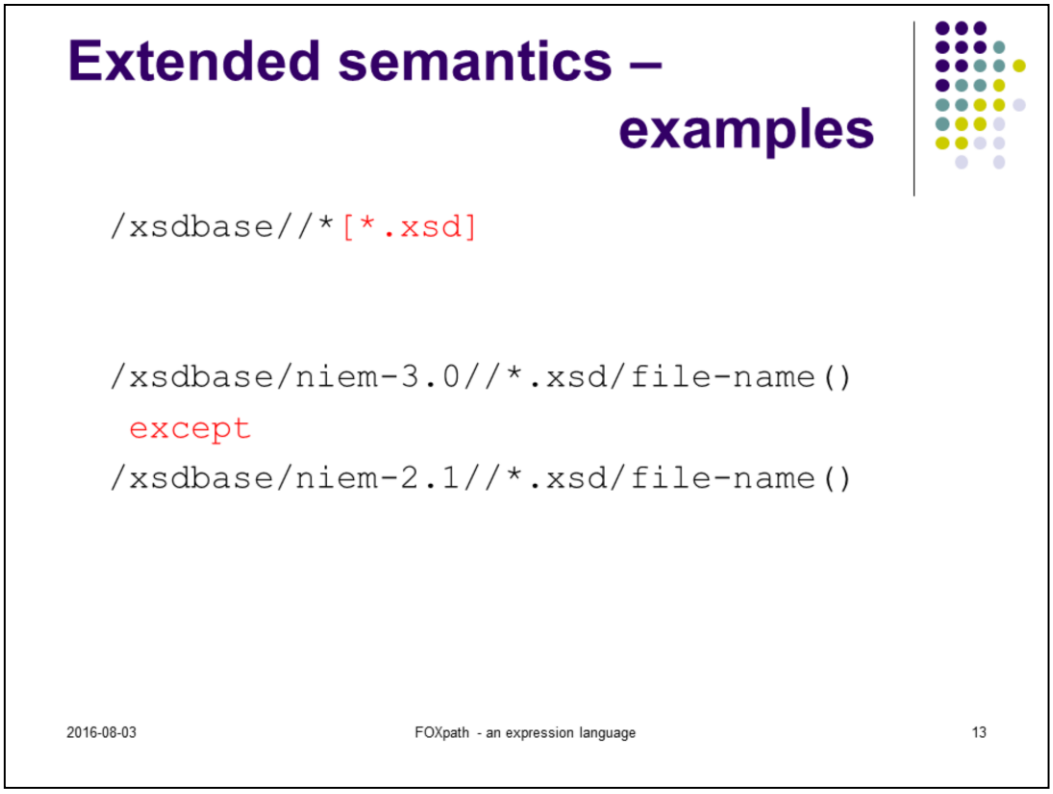
FOXpath supports most of the axes supported by XPath, excepting of course the attribute axis, as well as the preceding and following axis, which are not believed to be useful when navigating a file system. As in XPath, the child axis is the default axis.



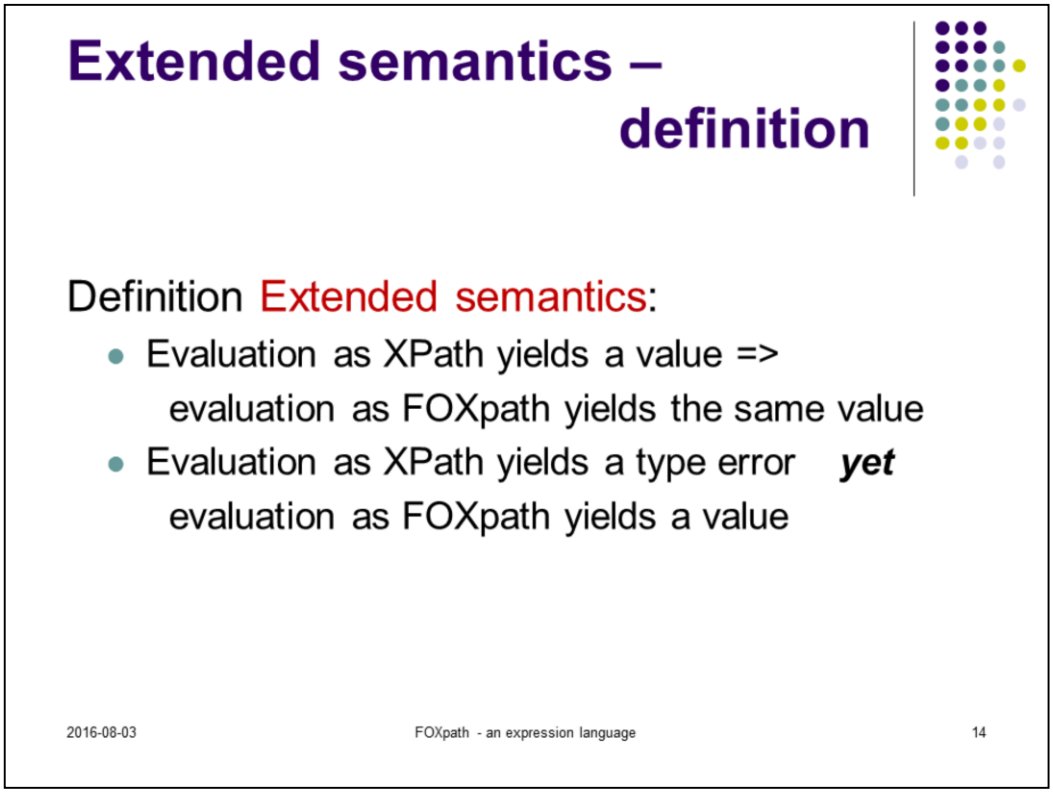
The fox name test filters by file or folder name. In contrast to the node name test of XPath, name patterns with inserted wildcards are supported. As file names can be almost arbitrary strings – not only NCNames – the syntax of a name test must make sure to avoid parsing ambiguity.



In XPath, the steps of a path expression are combined by the path operator, represented by a slash. In FOXpath, steps are combined by the foxpath operator, also a slash. The semantics are very similar to the path operator, but there is a key difference: what is pumped from left to right are URIs, not nodes.



As we want FOXpath expressions to be just as elegant and concise as XPath expressions, we are motivated to introduce a slight extension to the expression semantics of XPath. Consider these examples, which look elegant and intuitive, yet would produce type errors if we retained XPath semantics without any changes.



Extended semantics are defined in terms of a comparison between the outcomes of evaluating the expression as an XPath or as a FOXpath expression.

Extended semantics – rules

- Effective boolean value

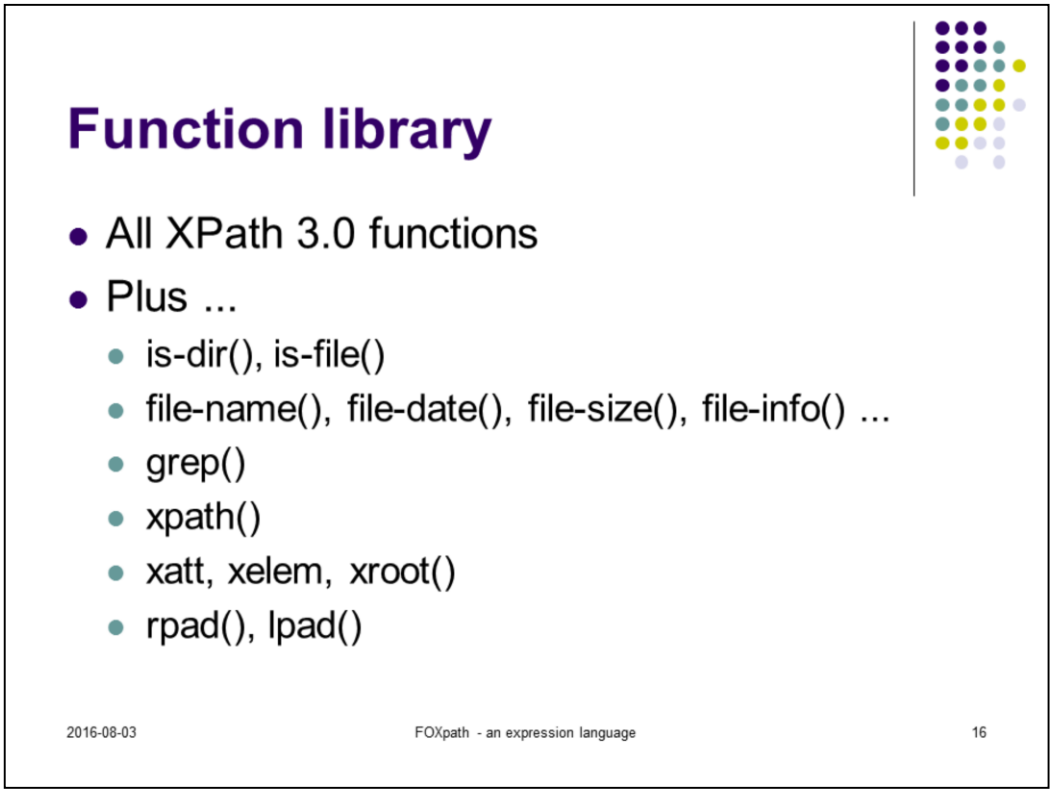
  Case: value(E) = sequence of >1 atomic items

  XPath: EBV(E) raises type error

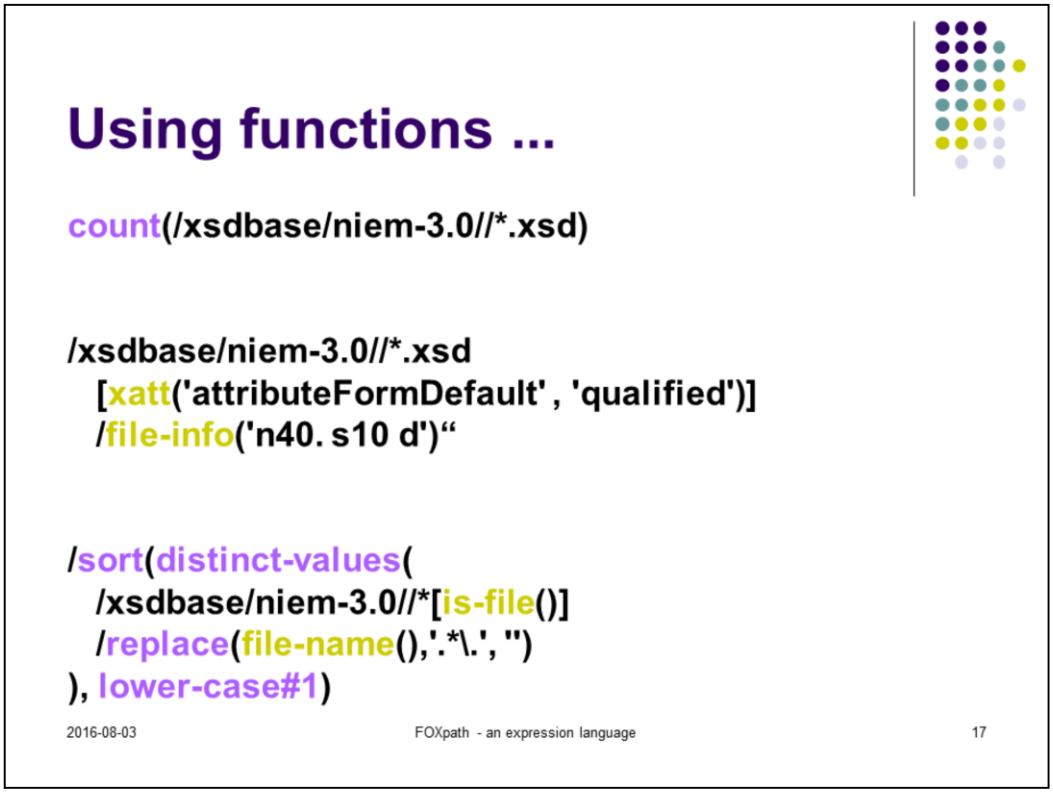
  FOXpath: EBV(E) = EBV(E[1])

- E1 or E2 contains atomic item =>
  - E1 union E2 = distinct-values ((E1, E2))
  - E1 except E2 = distinct-values (E1[not(. = E2)])
  - E1 intersect E2 = distinct-values (E1[. = E2])

FOXpath defines four semantic extensions.



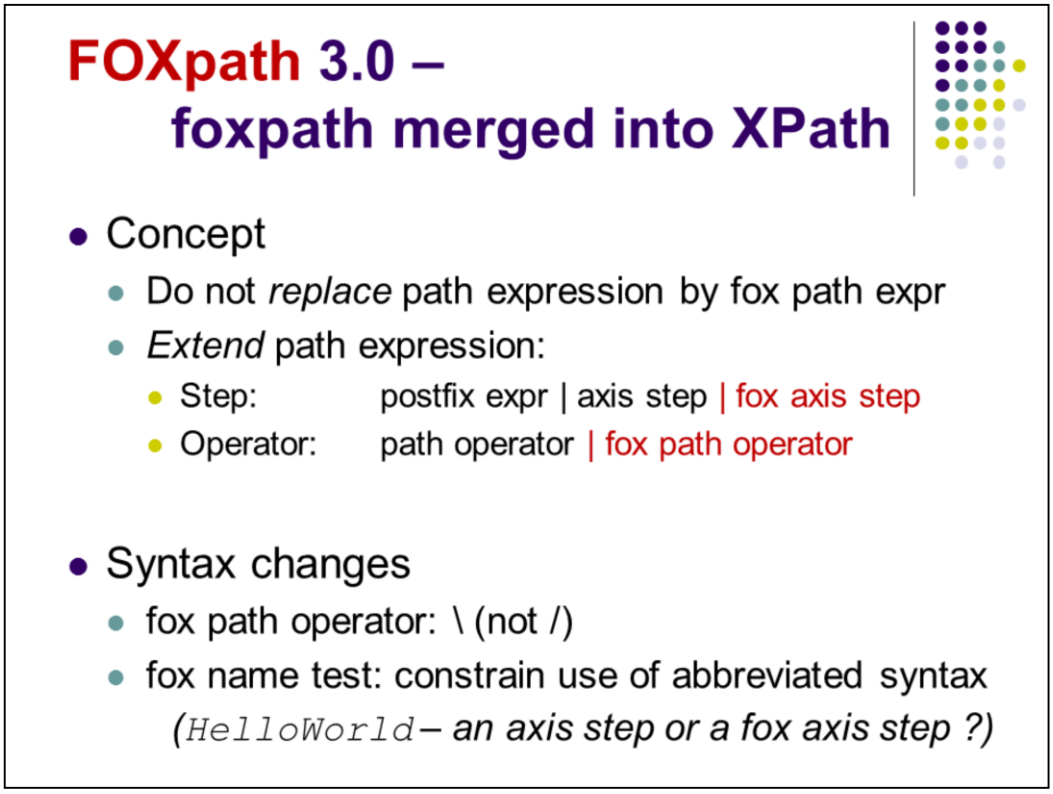
The power of XPath relies on a standard ibrary of built-in functions. FOXpath supports all XPath functions, plus a few additional functions believed to be especially useful when navigating and reporting file system contents.



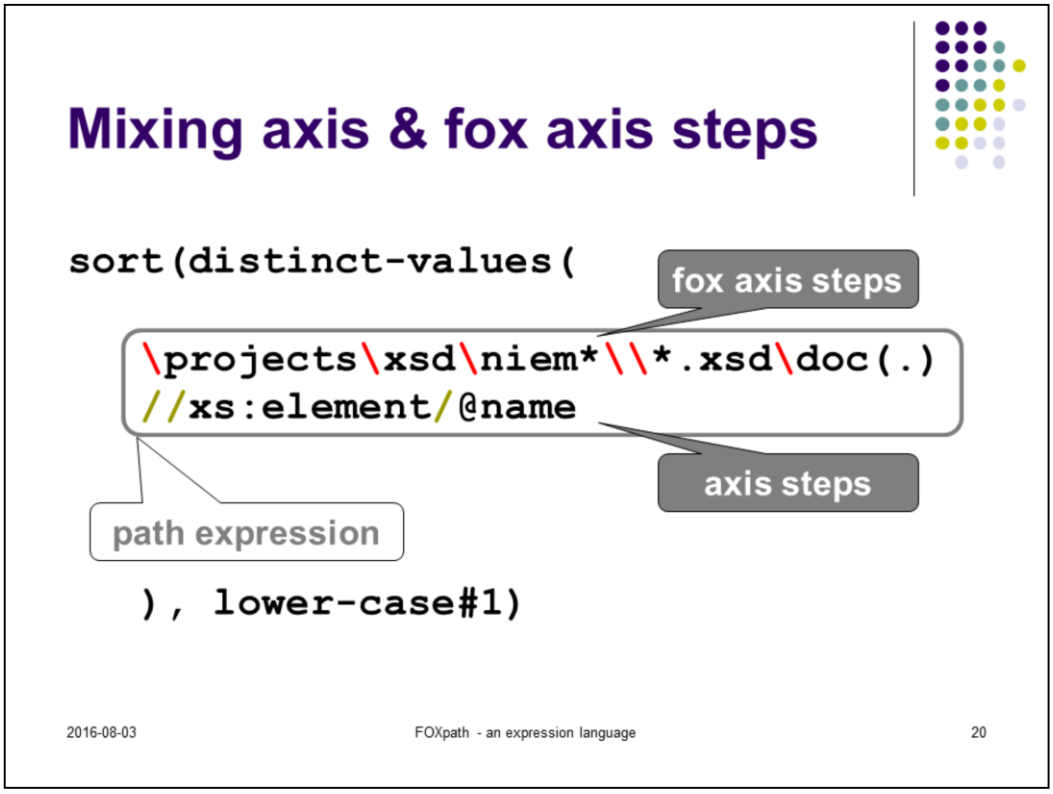
A few examples illustrate the use of XPath as well as FOXpath functions.

Bringing It All Back Home

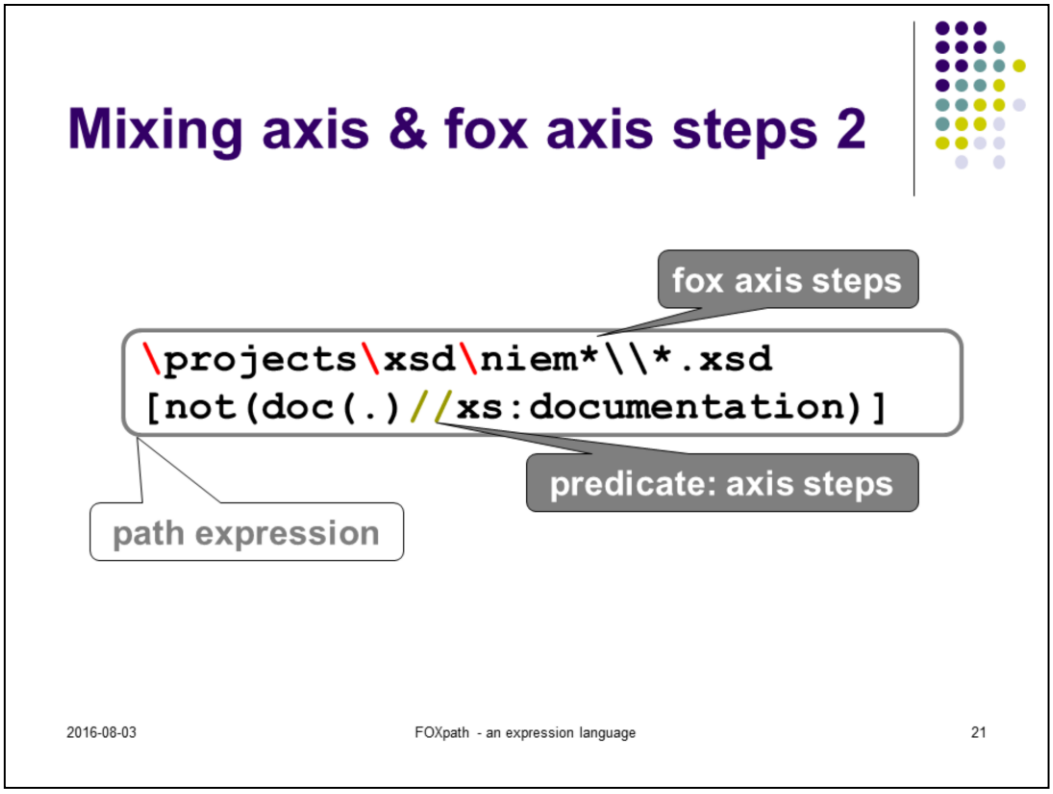
If FOXpath is almost identical to XPath, it is natural to ask if we cannot merge FOXpath back into XPath, obtaining an extended version of XPath, rather than a modified copy..



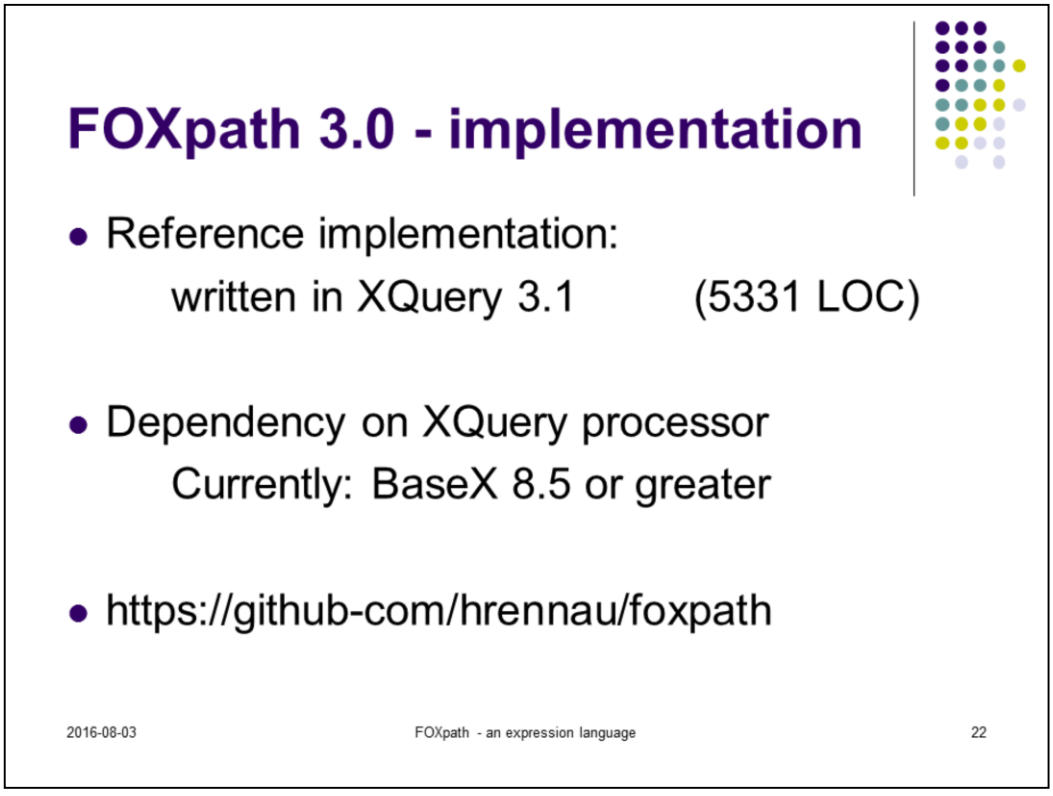
Indeed – it‘s possible! This new version of FOXpath is called FOXpath 3.0, as it is an extended version of XPath 3.0.



With FOXpath 3.0, you can not only use fox axis steps, you can also mix axis and fox axis steps within a single path expression. This enables expressions accomplishing a two-phase navigation: initial steps select resources, following steps navigate into their contents.



Another example of mixing is the use of fox axis steps with predicates containing axis steps, which means resource selection based on resource contents.

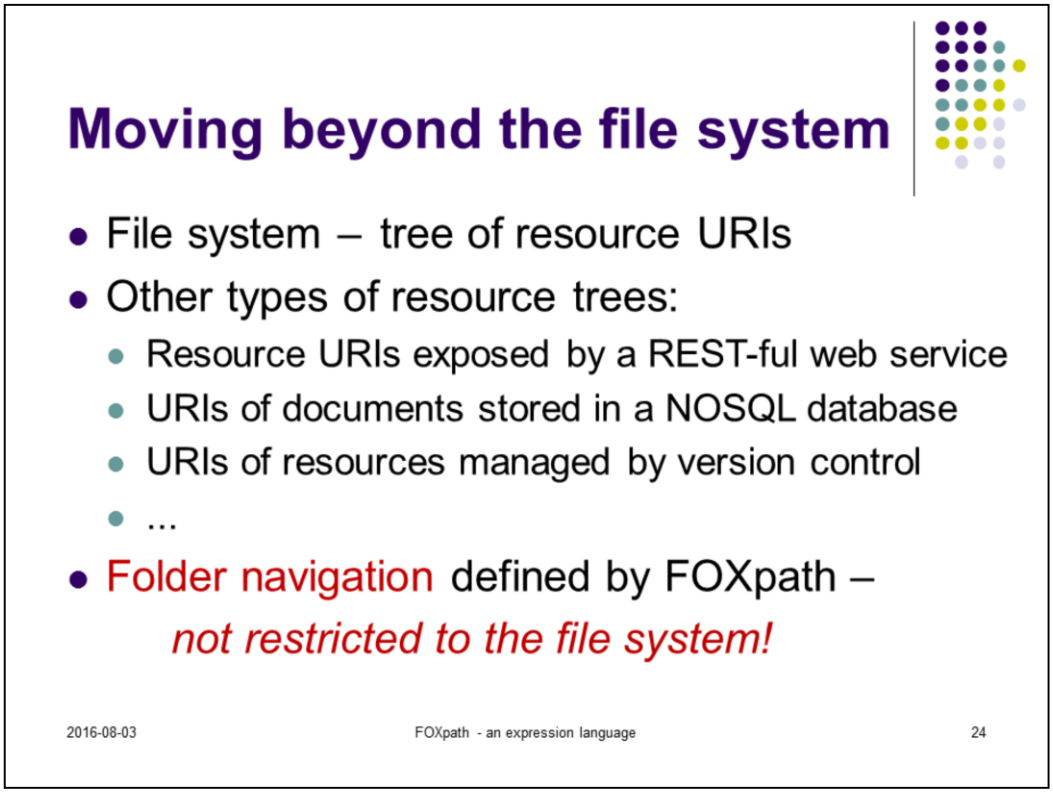


A reference implementation of FOXpath 3.0 is available.

```
for $d in /xsdbase/niem-3.0//*[is-dir()] return
  concat(
    for $i in 1 to (count($d/ancestor~::*) - 2) return '. ',
    file-name($d))
```

```
. niem
. . adapters
. . . edxl-cap
. . . . 3.0
. . . edxl-de
. . . . 3.0
. . . edxl-have
. . . . 3.0
. . . geospatial
. . . . 3.0
. . appinfores
. . . 3.0
. . codes
. . . ansi_d20
. . . . 3.0
```

The implementation in action – a complex expression producing a tree representation of the folder tree found under the niem-3.0 folder.



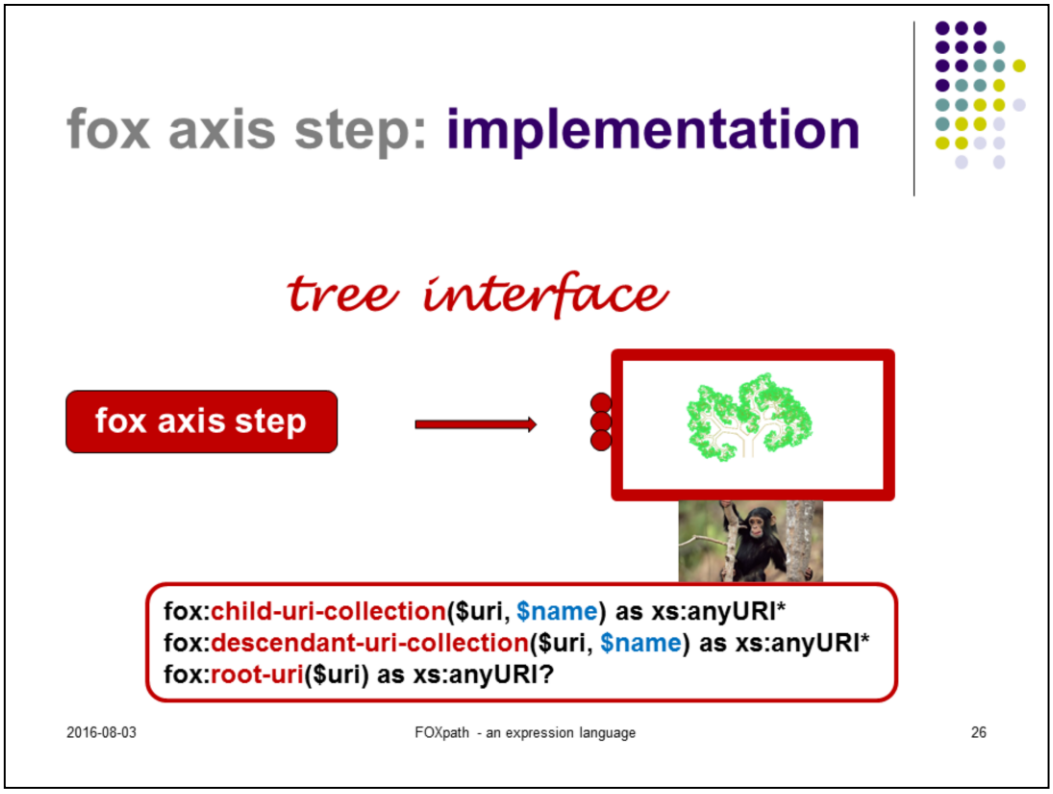
The folder navigation supported by FOXpath is not restricted to physical file systems – it can also deal with other kinds of resource trees, which may be regarded as logical file systems.

Folder navigation

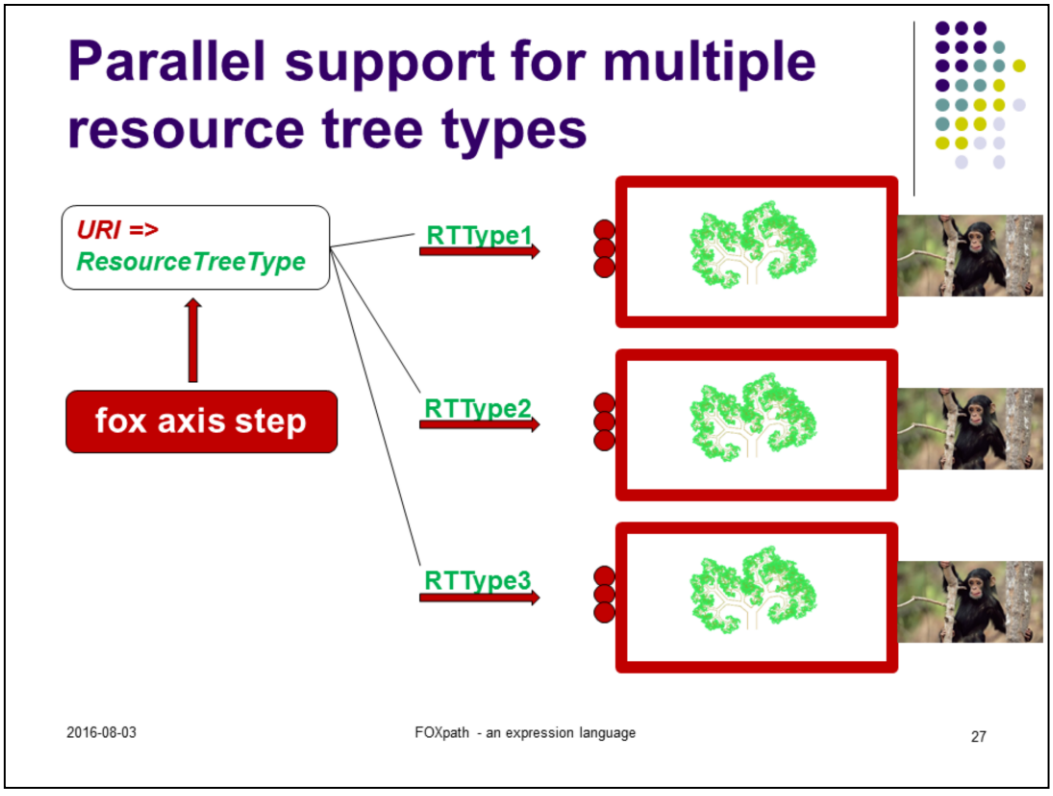
Basic building block of folder navigation:

fox axis step

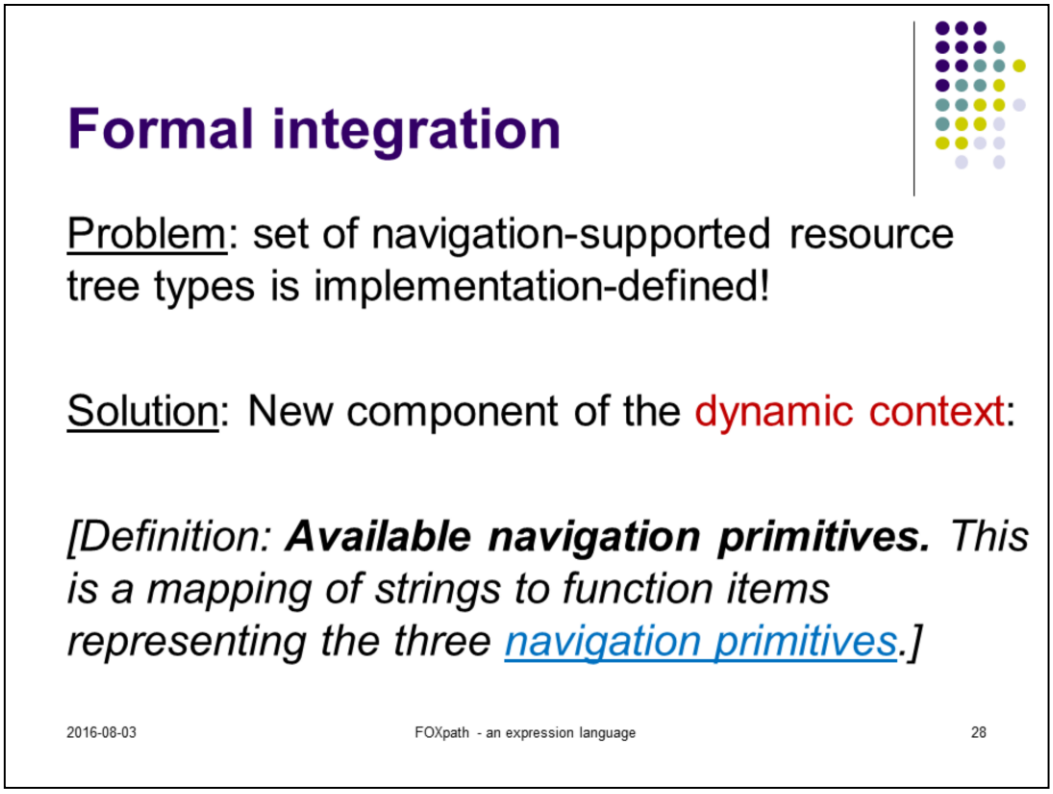
It is important to recognize the basic building block of folder navigation – which is the fox axis step. The question arises whether the fox axis step can deal with logical file systems as well.



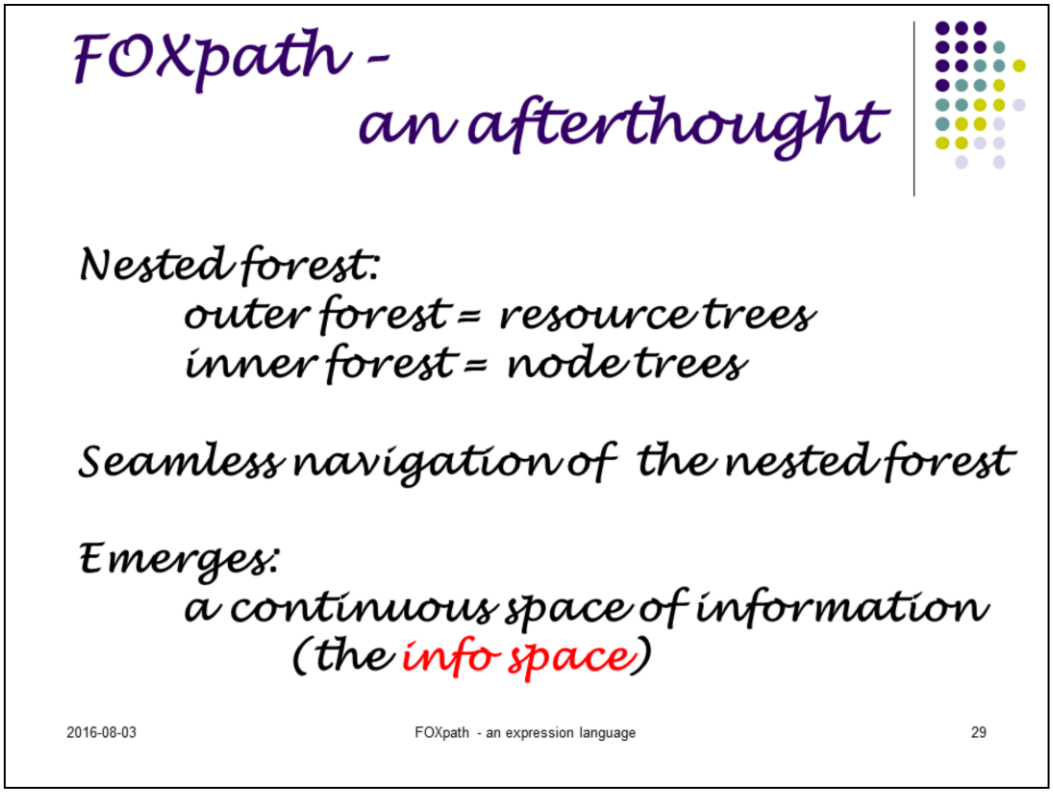
As you remember, the fox axis step combines a fox axis with a fox name test. A complete implementation can be built upon navigation primitives, three functions mapping an input URI to output URIs – root, child and descendant URIs. The functions constitute a sufficient interface to the tree, as far as navigation is concerned. This interface completely hides the nature of the resource tree – physical file system, SVN repository, etc. In principle, FOXpath can navigate any type of resource tree, for which implementations of these three navigation primitives are available.



What is more: FOXpath can support in parallel any number of resource tree types. This presupposes that for a given URI, the type of containing resource tree can be determined. If this is the case, for a given context URI, the appropriate instances of the navigation primitives can be selected and the fox axis step can be executed.

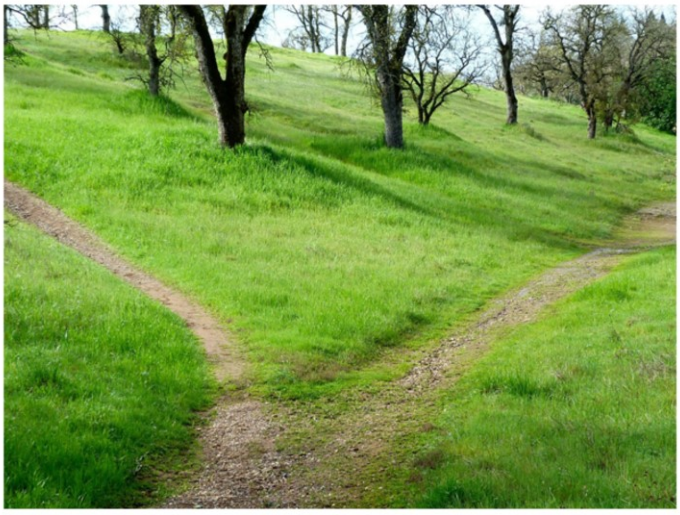


Support for various resource tree types will certainly be implementation dependent. How to integrate this variability cleanly into the highly standardized XPath language?



An afterthought: FOXpath complements tree navigation of resource contents with tree navigation of resource collections. FOXpath encourages us to perceive a nested forest of information: an outer forest consisting of resource trees, an inner forest consisting of the node trees corresponding to leaves of the resource trees. FOXpath enables us to navigate the complete structure in a seamless way, and therefore we may experience this nested forest as a single, continuous space – the info space.

The end! (The end?)



There is a choice to be made: either XPath remains an ingenious tool for navigating XML documents, or it will be extended to become the engine of the info space.