1. Summary

The problem of picking the best names and signatures for standard XPath functions is described.

A detailed solution is presented, that rests on fundamental XDM concepts, that is easy to implement, and has been implemented and used in practice.

2. The problem.

Here is what Michael Kay shared in the Xml.com chat:

"Naming of functions is a significant problem -- mainly because the namespace mechanism is grossly inadequate for the task of resolving ambiguous references; it's too "all-or-nothing". Yes, it's important that functions do what it says in the name (people misuse "contains" all the time because it's a poor choice of name). I'd love to find a good solution to this problem."

The number of standard XPath 3.1 functions is 197. They reside in essentially a flat space and remembering even the existence of some functions, or inferring by its name what it does, has become very challenging and unfriendly.

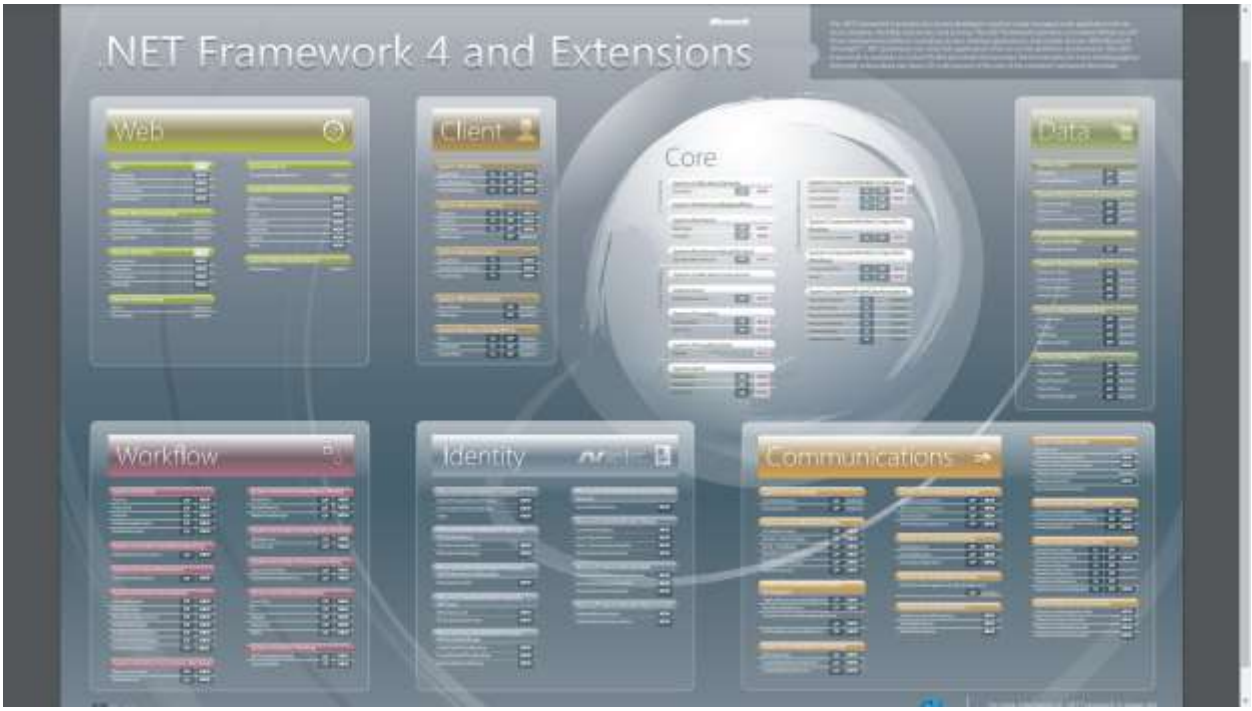
Even when using namespaces for naming, we still have a huge number of functions in the same namespace. Namespaces only allow to have flat sets of functions.

Let's look at other programming languages. A typical example is C#. In C# a namespace can contain not only individual items, such as classes, interfaces, structures, delegates,... etc., but most importantly, a namespace can contain nested namespaces within itself. Thus the namespace mechanism in C# provides a hierarchical, not flat, structuring of object's names.

It becomes possible to have to methods with the same name, here are a few examples:

Method	Class	Namespace	Remarks
`Contains`	string	System	
`Contains`	List<T>	System.Collections.Generic	
`IndexOf`	string	System	
`IndexOf`	List<T>	System.Collections.Generic	
`IndexOf`	Array	System	
`Etc …`	…	…	

This document contains a visual representation of the .NET 4.0 namespace hierarchy: http://download.microsoft.com/download/E/6/A/E6A8A715-7695-493C-8CFA-8E0C23A4BE1D/098-115952-NETFX4-Poster.pdf



Unfortunately, XML namespaces were not designed with hierarchy support in mind.

Then what to do?

3. Proposed solution.

Starting with XPath 3.1 we have a powerful mechanism of creating and accessing hierarchy of functions.

To represent something corresponding to a .NET namespace one can use a map whose keys are either names of functions or names of other maps (corresponding to a class (group of related functions) or to a namespace that is contained in the current namespace)

Thus we can have several functions named `Contains` that reside in different maps.

With this mechanism we can construct a hierarchy of any desired level of detail for

grouping just small sets of functions, so that there would be no name-conflicts within that group of functions. And we can have names like `Contains`, `Add`, `Remove`, `Get`, `Replace` etc. in as many such groups as necessary, without worrying about name conflicts, overloads, resolving ambiguous references, etc.

We can add to XPath and other languages (XSLT, XQuery) a mechanism for making available all functions and sub-maps in a given map.

For example in my work I have implemented an "extension function" `loadFunctionLibraries`, and here is a snapshot of its usage in real code:

```
(: =============== Include operators.xpath ====================:)
let $ops := Q{http://www.fxpath.org}loadFunctionLibraries#1(
concat($pBaseFXpathUri, '/f/operators.xpath')),

(: ================ Include folds.xpath ========================:)
  $folds := Q{http://www.fxpath.org}loadFunctionLibraries#1(
concat($pBaseFXpathUri, '/f/folds.xpath')
                                                ),

 (:    Special Folds

=============================================================
============:)
   $and := function ($booleans as xs:boolean*) as xs:boolean
{
  $folds?foldl_($ops?conjunction, true(), $booleans)
},
```

If the `loadFunctionLibraries` loads an XPath file that also calls `loadFunctionLibraries`, it is also invoked and loads its dependencies, and so on...

As we see this is not a pure theoretical proposal. For some time I have been creating pure XPath functions, all grouped in maps, and I have never even thought about

name conflicts, because no such problem exists between functions contained in different maps.

Conclusion:
The standard function library has surpassed a threshold where it has become challenging to name functions and give them the right signatures without conflict.

A solution to this problem was presented here, that would allow the same naming flexibility as that of modern programming languages with hierarchical namespaces.