Should you use namespace wildcards in XPath?

December 12, 2011 by *Evan Lenz*

Have you ever wished you could just skip having to deal with namespaces in your content? One way to do this is to avoid using namespaces altogether (i.e. avoid any xmlns or xmlns:\* declarations in your XML content). But given that namespaces are in widespread use, both in standard XML vocabularies and in custom application data, that isn't always an option. XPath does provide a convenient feature, namely "local name tests" (or "namespace wildcards"), which let you avoid having to type your content's namespace declaration in your query. In fact, you might be tempted to use it all the time, to save the typing. But I'm here to tell you: don’t. That would be a bad idea. Keep reading if you want to know when it might be safe to use them and when it's not a good idea.

What exactly am I talking about? See item #4 in the following table. There are four kinds of name tests in XPath, and three of them are wildcards:

| **What it matches** | **Example(s)** |
| --- | --- |
| 1. Match a specific QName | foo, xyz:bar, etc. |
| 2. Match any name | \* |
| 3. Match any name in a specific namespace | xyz:\* |
| 4. Match a specific local name, regardless of namespace | \*:foo |

In XPath 1.0 (pre-XQuery), only the first three kinds were supported. If you wanted to select a <foo> element regardless of its namespace, you'd have to write something like this:

|  |  |
| --- | --- |
| 1 | \*[**local-name**(.) **=** 'foo'] |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex01.xqy)[**ex01.xqy**](https://gist.github.com/evanlenz/1468934#file-ex01-xqy) hosted with ❤ by **[GitHub](https://github.com/)**

One rationale behind this perhaps obvious omission was that such a language feature might encourage some bad practices. The idea of a namespace is that it identifies a distinct set of names. Local names in different namespaces shouldn't necessarily be related to each other (<head> means one thing in HTML and quite another in, say, AnatomyML). Of course, that still didn't prevent people from using namespaces for things like versioning, where each new version of a vocabulary gets a new namespace URI.

In any case, local name tests (or "namespace wildcards") were added to XPath 2.0 (and thus XQuery):

|  |  |
| --- | --- |
| 1 | **collection**()//\*:foo |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex02.xqy)[**ex02.xqy**](https://gist.github.com/evanlenz/1468934#file-ex02-xqy) hosted with ❤ by **[GitHub](https://github.com/)**

The above query selects all elements with local name "foo", regardless of namespace. Even if you know these elements are in just one namespace, it can be a convenient shortcut. It saves you from having to write out the namespace declaration:

|  |  |
| --- | --- |
| 12 | **declare** **namespace** xyz**=**"http://example.com";  **collection**()//xyz:foo |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex03.xqy)[**ex03.xqy**](https://gist.github.com/evanlenz/1468934#file-ex03-xqy) hosted with ❤ by **[GitHub](https://github.com/)**

But there are two problems with using namespace wildcards like \*:foo. One is that the intentions are unclear. Did you really mean that? Are there really elements named <foo> in more than one namespace? Or were you just being lazy? The other problem is a performance one. MarkLogic indexes elements by QName, not by local name. That means namespace wildcards won't utilize the index and will require a lot of filtering. We can prove this by using our friend xdmp:plan() (or [its cousin xdmp:query-trace()](http://developer.marklogic.com/blog/learning-with-query-trace)):

|  |  |
| --- | --- |
| 123 | **xdmp:plan**(  **collection**()//\*:foo  ) |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex04.xqy)[**ex04.xqy**](https://gist.github.com/evanlenz/1468934#file-ex04-xqy) hosted with ❤ by **[GitHub](https://github.com/)**

The output shows how many "fragments" (equivalent to documents, unless you've enabled fragmenting) will have to be read in order to resolve this query. Normally, MarkLogic uses its Universal Index to minimize the number of document reads it has to make. In this case, we can see from the output that the "\*:foo" step ("Step 2" below) is problematic:

|  |  |
| --- | --- |
| 123 | <qry:info-trace>Analyzing path: fn:collection()/descendant::\*:foo</qry:info-trace>  <qry:info-trace>Step 1 is searchable: fn:collection()</qry:info-trace>  <qry:info-trace>Step 2 does not use indexes: descendant::\*:foo</qry:info-trace> |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex05.xml)[**ex05.xml**](https://gist.github.com/evanlenz/1468934#file-ex05-xml) hosted with ❤ by **[GitHub](https://github.com/)**

Looking further down the output, we see the number of fragments that would have to be opened for the filtering stage:

|  |  |
| --- | --- |
| 12 | <qry:info-trace>Selected 14944 fragments</qry:info-trace>  <qry:result estimate="14944"/> |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex06.xml)[**ex06.xml**](https://gist.github.com/evanlenz/1468934#file-ex06-xml) hosted with ❤ by **[GitHub](https://github.com/)**

This is not the number of documents that have a <foo> element. This is the total number of documents in my database. So obviously, this query is going to run very slowly, because it's forcing all of those fragments to be read from the disk.

In contrast, let's look at the plan with the case where we specify the exact QName:

|  |  |
| --- | --- |
| 1234 | **declare** **namespace** xyz**=**"http://example.com";  **xdmp:plan**(  **collection**()//xyz:foo  ) |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex07.xqy)[**ex07.xqy**](https://gist.github.com/evanlenz/1468934#file-ex07-xqy) hosted with ❤ by **[GitHub](https://github.com/)**

In this case, we see that the path is "fully searchable." In other words, all the steps contribute index constraints that can be used to narrow down the possible number of matching documents:

|  |  |
| --- | --- |
| 1234 | <qry:info-trace>Analyzing path: fn:collection()/descendant::xyz:foo</qry:info-trace>  <qry:info-trace>Step 1 is searchable: fn:collection()</qry:info-trace>  <qry:info-trace>Step 2 is searchable: descendant::xyz:foo</qry:info-trace>  <qry:info-trace>Path is fully searchable.</qry:info-trace> |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex08.xml)[**ex08.xml**](https://gist.github.com/evanlenz/1468934#file-ex08-xml) hosted with ❤ by **[GitHub](https://github.com/)**

And we see further down that MarkLogic knows *a priori*, from the Universal Index, that *no* <xyz:foo> elements exist in the database:

|  |  |
| --- | --- |
| 12 | <qry:info-trace>Selected 0 fragments</qry:info-trace>  <qry:result estimate="0"/> |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex09.xml)[**ex09.xml**](https://gist.github.com/evanlenz/1468934#file-ex09-xml) hosted with ❤ by **[GitHub](https://github.com/)**

So simply by specifying the namespace part of the QName, we've gone from having to read *all* the documents in the database to *none* of them.

To summarize, you should generally avoid namespace wildcards like \*:foo for two reasons:

* performance, and
* clarity.

Are there ever any cases where it's okay to use \*:foo? Performance is not nearly as big an issue when you're processing documents that you're already committed to opening. For example, if you're processing a single zip file manifest (the result of xdmp:zip-manifest()), then using \*:part because you're too lazy to declare the zip namespace isn't a problem as far as performance goes, because you're not searching among thousands or millions of documents and the index doesn't even come into play. Still, in production code, it's a good idea to declare the namespace and use zip:part so your intentions are clearly documented. Of course, when your intentions actually are to select an element with a specific local name but any number of namespaces, then you can use \*:foo, but, again, be sure it's not when you're searching across the database. In that case, if it's possible, you should enumerate all the QNames, so MarkLogic can most effectively narrow down the result set based on what it knows from its indexes:

|  |  |
| --- | --- |
| 1 | //(abc:foo**|**def:foo**|**xyz:foo) |

[**view raw**](https://gist.github.com/evanlenz/1468934/raw/ex10.xqy)[**ex10.xqy**](https://gist.github.com/evanlenz/1468934#file-ex10-xqy) hosted with ❤ by **[GitHub](https://github.com/)**

If you didn't even know namespace wildcards existed in XPath, then you might find it odd that I'm both introducing them to you and recommending against using them in the same article. Consider this just another chance to become familiar with xdmp:plan(), which is much more generally useful. It will help you write fast queries and understand what makes them fast. *Do*use it.