

# Internationalization Features in XPath, XQuery and XSLT

Frank Yung-Fong Tang (Google)  
Liam Quin, Felix Sasaki (W3C)

1

IUC 29, San Francisco, March 2006

## Overview

- The Big Picture:
  - General overview of XQuery and future developments
  - How XQuery might be deployed
  - i18n and XQuery
- Into the Deep:
  - Introduction to XPath
  - Introduction to XQuery / FLWOR
  - Collations
  - String functions
  - Regular expressions

2

IUC 29, San Francisco, March 2006

This presentation will provide an overview on internationalization features in the two languages XQuery and XSLT, with a focus on XQuery. It is split into three parts:

- an overview on the current status of XQuery / XSLT, deployment scenarios and i18n issues
- a deep discussion of general features of XQuery, with an emphasis on features relevant for internationalization like string functions and regular expressions
- some specific issues concerning string versus typed values, time zone handling and IRI processing.

## Overview

- **Some specific Issues:**
  - String values versus typed values
  - Time Zone handling
  - IRI processing

## The Big Picture

- General overview of XQuery and future developments
- How XQuery might be deployed
- i18n and XQuery

# General Overview of XQuery and Future Developments

- What is XQuery?
- Current status of XQuery and related specs
- Future Work
- Internationalization and XQuery
- Choosing between XQuery and XSLT

## Overview 1: What is XQuery?

- A database query language built on XML and on W3C XML Schema-compatible XML data types (integer, double, string, URI/IRI, etc)
- Two syntaxes: XML and text
- Expressions can combine multiple data sources (e.g. RDBMS tables, files, XML databases)

6

IUC 29, San Francisco, March 2006

The XML Query Working Group is currently busy processing public feedback, and also producing both a full text search facility and an update facility.

Public drafts of both of these are available from the public XML QUery Web page at <http://www.w3.org/XML/Query/> and also at the Technical Reports page, <http://www.w3.org/TR/>.

(\*) A future version of XML Query is expected, but the exact feature set is not yet known. To some extent it will depend on the results of the current Candidate Recommendation phase.

## Overview 2: Sample XQuery

```
declare variable $rb {  
  doc("http://www.example.org/recipes.xml")  
};  
<results> {  
  for $r in $rb/recipes/recipe  
  where ingredients/item = 'eggs' and  
        ingredients/item = 'marmite'  
  return $r/title  
} </results>
```

7

IUC 29, San Francisco, March 2006

The details will be explained later in the presentation.

The only thing that might not be obvious is that when one or both operand of = is a sequence, = becomes the “any of those is” operator, so that ingredients/item = “eggs” is true if there is any item element that’s a child of the ingredients element and that has the text content value “eggs”. There might be other ingredients (and almost always will be, of course) but that’s OK, only one has to match.

So the query finds recipes which use both eggs and Marmite.

## Current Status of XQuery

- XQuery is currently in Candidate Recommendation status

This is a general call for implementations. We (W3C) are looking to learn to what extent the implementations interoperate: does every query return the same results on every system? If not, are the variations reasonable and permitted?

## Implementations

- We have listed over forty implementations on the XQuery Public Web page at <http://www.w3.org/XML/Query/>.
- We also have a test suite, designed to help us determine the level of interoperability, with over 9,500 test cases.

## Next Steps

- If Candidate Recommendation is successful, the next step will be Proposed Recommendation, and thence to W3C Recommendation, an internationally recognized standard.

## Next Steps / 2

- The Java Community Process has already proposed an API for calling XQuery
- ISO SQL has a mapping for SQL to XQuery data types as part of SQL/X.

## Next Steps / 3

- XML Query Working Group is processing public feedback
- Upcoming are internationalized full text search support and also an XML Query update facility.
- A future version of XQuery itself is also expected.

## Next Steps / 4: Full Text

- Full Text draft extends XPath (and XQuery) to add *ftcontains* and also *score expressions* for sorting/ranking;
- You can specify the language of individual words/phrases in searches, so that e.g. language-sensitive stemming rules can be applied, and the right stop-list used;
- XQuery engines are expected to use `xml:lang` for indexing.

## Sample XQuery with Full text

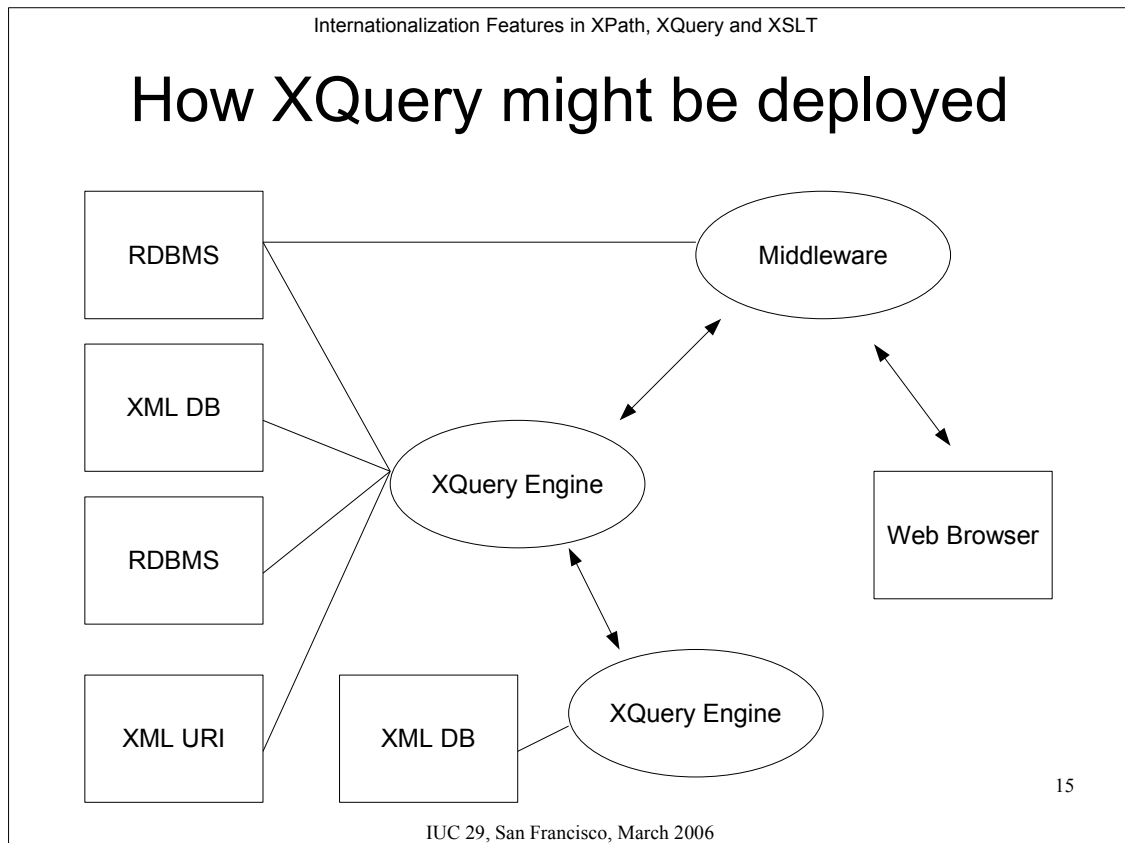
```
declare variable $rb {  
  doc("http://www.example.org/recipes.xml")  
};  
<results> {  
  for $r in $rb/recipe[  
    ingredients ftcontains "barrels of brandy"  
  ]  
  return $r/title  
} </results>
```

14

IUC 29, San Francisco, March 2006

Not a very convincing example perhaps, but I wanted to use a phrase, and although "to be or not to be" is my favorite, most search engines have problems with it, either because of stop-words or because of interpreting "or not" as a boolean operator.

I do have a very old recipe book that in places calls for "a barrel of your best brandy".



### Some background to how XQuery might be deployed

An XQuery engine can address multiple sources of data; in many cases this includes remote relational databases (e.g. via JDBC), local relational stores, native XML stores and also the file system. Databases will typically make indexes to both XML and relational data, to enable efficient querying. This is usually done when documents are loaded, or when relational data is updated, not at query time. This means that a query might run against multiple indexes, and that these indexes might be on servers in various different parts of the world.

The recipient of the query result might be an automated process (for example, an XSLT transformation or a database load) or it might be a human user, possibly in yet another part of the world via the World Wide Web.

## Deployment Issues

- Any component could be in any location, or even spread out over multiple locations
- Document collection can be *very* heterogeneous.
- XML 1.1 support is optional, so some “newer” characters may be legal in XML names only with some XQuery engines and not others.

## Deployment Issues / 2

- Both relational databases and XML Stores make indexes when documents are loaded, not at query time; these indexes may use *local* defaults for timezones, locales, collations, etc.
- If your data has e.g. an XML attribute whose value sometimes has timezones and sometimes not, you might get surprising results when the data is normalized!

## Deployment Issues / 3

- The recipient of query results could be a user agent or human, or, more likely, Java, JavaScript/ECMAScript, XSLT, etc.
- You may need to do runtime string generation, or to access Java resource bundles.

## Deployment Issues / 4

- XQuery supports *modules*, named collections of functions, which can be collections of XQuery functions or can be native.
- Hence you can implement efficient lookup in message catalogues, e.g. locale/po files or bundles.
- Errors have URIs so that the corresponding (translated) natural-language string can be presented to users.

## Some Internationalization Issues

- Two syntaxes, text and XML (XqueryX).
- XQueryX uses standard XML encoding mechanism.
- XQuery text syntax has optional version and encoding header, and application/xquery Internet Media Type to prevent transcoding.

20

IUC 29, San Francisco, March 2006

There are currently two syntaxes for XML Query: a plain text syntax and an XML representation of the parse tree.

XML Query uses Unicode as the internal representation in both cases (as does XML itself). The default encoding for the text version is UTF-8, but there is provision for a query to be in an alternate encoding and to label its encoding.

To prevent transcoding by Web proxies, the text syntax has an application/\* Internet Media Type.

The XML format uses the XML encoding mechanism, as you would expect.

There is explicit support for system-specific collations for sorting and comparisons. More about these from Frank.

Error conditions have standardized codes (actually URIs) so that an implementation can provide both the code and a native-language message. English messages are given in the Specifications.

Date and Time formats are in an ISO format, with an extension in XML Query to allow a partial value. There are some internationalization issues with timezones, and Felix will expand on these.

## When to use XQuery and XSLT

- XQuery implementations more likely to do efficient joins and cross-database queries;
- XQuery implementations exist that can use a full text inverted index on XML files, for fast searches;
- No plans for XSLT update facility

21

IUC 29, San Francisco, March 2006

If you are putting together a large middleware system using XML Query, you may find you need to deal with resource bundles and generated text.

Luckily, XML Query provides "modules", which are collections of functions in their own namespace. Most XML Query implementations (including open source ones such as Saxon, Galax and Qizx/Open) support the addition of Java or C native methods as modules, so that access to locale ".po" facilities or bundles might be performed reasonably efficiently.

## When to use XQuery and XSLT / 2

- XML Query has a text syntax, so you can't use XML tools. The XML syntax is too detailed to read, but can be used for introspection.
- XSLT has a comfortable readable XML syntax.

## When to use XQuery and XSLT / 3

- XML Query produces XML output; XSLT can also produce text and HTML (but so can most XML Query implementations, as an extension)

I'm hoping that a future version of XML Query will provide standardized access to more of the serialization options. In the mean-time, Saxon and QuzX/open both support generation of XHTML, and probably so do others.

## When to use XQuery and XSLT / 4

- Use XSLT if you are transforming entire documents, use XQuery if you are extracting from many documents. Use whichever you prefer if you are doing something between these extremes.
- Over 40 XQuery implementations, many specialized, so do some research first!

## Into the Deep

- Introduction to XPath
- Introduction to XQuery / FLWOR
- Collations
- String functions
- Regular expressions

## Introduction to XPath

- XPath is a language for addressing parts of an XML document
  - XML can be view as in Tree structure, like a directory in the file system
  - XPath start from a node and select (other) nodes in the tree
- Optional “/” followed by zero or more location steps separated by “/”
- Location step: Axis::NodeTest[Predicates]

26

IUC 29, San Francisco, March 2006

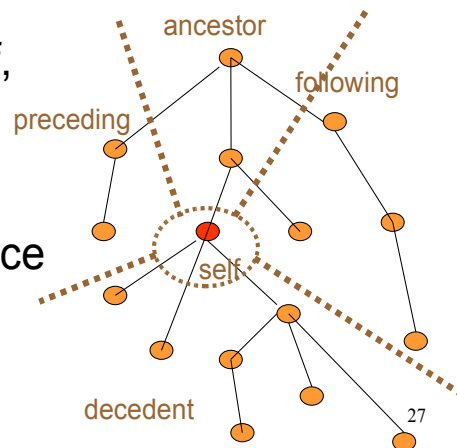
XPath is a language for addressing parts of an XML document. In general, we can view XML as a tree structure. XPath start for a node in that tree and select other nodes in the tree. It use a ‘path expression’ to make the syntax clear and clean.

XPath is an important specification and are used by several W3C technologies, including but not limited to XSLT, XML Schema, XLink and XQuery. In XSLT, XPath is used to select which element to apply for transformation. See <http://www.w3.org/TR/xslt> for details. A limited subset of XPath features is also used in XML Schema to select the elements for key, keyref and unique. Please see See "5 Advanced Concepts III: The Quarterly Report " in XML Schema Part 0 Primer <http://www.w3.org/TR/xmlschema-0/> for details. XQuery is build on top of XPath and used in the FLWOR expression to select the sequence of nodes for proccession. We will discuss some details of this process in this presentation. Please also reference to "XQuery Use Case" <http://www.w3.org/TR/xquery-use> for details.

XPath is a path expression consist of an optional “/” followed by zero or more location steps separated by “/”. Each location step is in the format of Axis::NodeTest[Predicates]. The “::” is used to separate Axis from NodeTest. And the ‘[’ and ‘]’ are used to wrap around the optional Predicates part.

## Axis in XPath

- ancestor, descendant, following, preceding, and self partition the tree
- ancestor-or-self, descendant-or-self, following-sibling, preceding-sibling, parent, child
- attribute, namespace



IUC 29, San Francisco, March 2006

Axis in XPath could be one of the following: ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, or self

These axis is case sensitive. Five axis- ancestor, descendant, following, preceding, and self partition the tree. ancestor-or-self and descendant-or-self are union of two basic axis. parent is the immediate ancestor of the node. child is the immediate descendant of the node. following-sibling and preceding-sibling are following and preceding nodes that share the same parent. The attribute and namespace nodes are special and are NOT be treat as child or descendant of the node. But these node will have a parent node.

### W3C Recommendation:

- *XML Path Language (XPath): Version 1.0. November, 1999.*  
<http://www.w3.org/TR/xpath>.

### W3C Working Drafts:

- *XML Path Language (XPath) 2.0* <http://www.w3.org/TR/xpath20/>
- *XQuery 1.0 and XPath 2.0 Functions and Operators*  
<http://www.w3.org/TR/xpath-functions/>

## Node Test and Predicate

- **NodeTest:** (separated from Axis by :: )
  - Name of the element/attribute or
    - Ex: child::BOOK
  - Kind of the node (e.g. node() or comment())
    - Ex: following::comment()
- **Predicate** (optional, wrap between [ and ] )
  - Use to add additional conditions
    - Ex: child::BOOK[position()=last()]
  - 'and' and 'or' can be used inside the predicate
    - Ex: //img[count(preceding-sibling::\*)=1 and count(ancestor::\*[attribute::onclick]=0)]

28

IUC 29, San Francisco, March 2006

NodeTest could be either the name of the element(or attribute) or a the type of the node. Below are some example of Local steps that have Axis, and NodeTest:

- self::\*        select self
- self::AAA    select self if the element name is AAA
- child::AAA   select all children if the element name is AAA
- ancestor:: AAA   select all ancestors if the element name is AAA
- self::node() select self if it's a node
- self::comment() select self if it's comment
- attribute::AAA        select attributes AAA

Predicate are optional parts in location step. It is wrapped between '[' and ']'. The predicate is used to add additional constraints to filter out nodes. Usually use with some build-in functions. We will discuss some of the function shared by XPath and XQuery later. The predicate can be joined by 'and' or 'or'. Below are some example of Local steps that have Axis, NodeTest and Predicate:

- child::AAA[position()=1]        select first AAA child
- child::AAA[position()=last()]    select the last AAA child
- self::\*[node()]        select self if it's a node (same as self::node())

## Abbreviation in XPath

- (nothing)                      child::  
   - Ex: AAA                      child::AAA
- /                                  document root  
   - Ex: /AAA                      start from the root: child::AAA
- @                                  attribute::  
   - Ex: @height                  attribute::height
- //                                  /descendant-or-self::node()/  
   - Ex: //AAAall AAA descendants of root
- .                                  self::node()
- ..                                  parent::node()  
   - Ex: ../AAA                      all AAA siblings that is named AAA
- \*                                  all element children of the context node
- para[1]                              selects the first para child of the context node

29

IUC 29, San Francisco, March 2006

XPath also provide some abbreviation to make the path syntax looks clean and short. The leading '/' before any location step is used to abbreviate the document root. "child::" can be abbreviate as nothing. For example "BOOK" is equal to "child::BOOK". "attribute::" can be abbreviate by using "@". For example "img/@height" is equal to "child::img/attribute::height". '/' is used to abbreviate "/descendant-or-self::node()". It will get all descendant or self. For example "/html//form" will get all the 'form' element inside the html, regardless how many levels deep. "." is used to abbreviate "self::node()" and ".." is used to abbreviate "parent::node()". \* selects all element children of the context node. para[1] selects the first para child of the context node which is equal to para[position()=1].

## Unicode in XML and the Impact on XPath

In XML, the following may contains any Unicode 'letters':

- Content of Elements (content): Ex: <author> 譚永鋒 </author>
- Comments (Comment): Ex: <!-- 作者名 -->
- Name of Element Type (Name): Ex: < 作者 >Frank Yung-Fong Tang</ 作者 >
- Attribute Value (AttValue): Ex:<author gender=" 男" />
- Attribute Name (Name): Ex:<author 性別 ="Male"/>
- Others: Ex: <?xml-stylesheet type="text/css" href="ming.css" title=" 明報" ?>

Some XPath to Query this type of XML:

- / 文件 / 作者
- // 作者 /@ 性別
- / 文件 /text()
- / 文件 /comment()
- 文件 / 作者 [@ 性別 =" 男 "]

30

IUC 29, San Francisco, March 2006

Since XML allow Unicode characters in many different places, we could have a XML as below:

```
<?xml version="1.0" ?>
  <?xml-stylesheet type="text/css" href="ming.css" title=" 明報" ?>
  <!DOCTYPE 文件 [
    <!ELEMENT 文件 ANY>
    <!ELEMENT 作者 (#PCDATA)>
    <!ATTLIST 作者 性別 CDATA #IMPLIED>
  ]>
  < 文件 >
    <!-- 作者名 -->
    < 作者 性別 =" 男" > 譚永鋒 </ 作者 > 於香港
  </ 文件 >
```

See more details in

Frank Yung-Fong Tang, *Use CSS and XML for International Content*, The Eighteenth International Unicode Conference, Hong Kong, 2001, [http://people.netscape.com/ftang/paper/unicode18/css\\_xml\\_for\\_intl.ppt](http://people.netscape.com/ftang/paper/unicode18/css_xml_for_intl.ppt)

We may use the following XPath to address parts of the XML above:

- / 文件 / 作者
- // 作者 /@ 性別
- / 文件 /text()
- / 文件 /comment()
- / 文件 / 作者 [@ 性別 =" 男 "]

## Introduction to XQuery

- A Query language for XML
- Use query language syntax- easier for programmer to read.
- Could be serialized into a XML syntax -XQueryX
- Use XPath to select part of the XML
- Support XML Schema but can operate without it.
- Input zero or more XML, output one XML
- Is a Query, not a filter.
- Everything in XQuery is a 'Sequence'
- The core of XQuery is in the FLWOR expression
- Document order by default.

31

IUC 29, San Francisco, March 2006

XQuery is a query language for information stored in XML. It uses query language syntax, not markup language syntax, for developer to retrieve element and/or attributes from a given XML. It is build on top of the facilities available in XPath. It uses XPath to 'walk through' the elements. It was designed to work with XML Schema but can be operate without the knowledge of XML Schema in a subset of usage. It is designed to pull information from several XML documents (or databases) and generate output information in the form of XML. It uses a Query model instead of a filter model. Optionally, it could be serialized into markup language syntax such as XQueryX.

Almost everything in XQuery is considered as a sequence of items instead of a single item. The core functionality of XQuery is in the FLWOR (For/ Let/ Where/ Order/ Return) expression. And by default, all the return items are in the original 'document order' if no explicit options specified.

XQuery is specified by several W3C documents. Author recommend the beginner read, in the listed order, the first three documents to gain more understanding about XQuery.

• *XML Query Use Cases* <http://www.w3.org/TR/xquery-use-cases/>

• *XQuery 1.0: An XML Query Language*

<http://www.w3.org/TR/xquery/>

• *XQuery 1.0 and XPath 2.0 Functions and Operators*

<http://www.w3.org/TR/xpath-functions/>

## Introduction to XQuery - FLWOR Expression

- `ExprSingle ::= FLWORExpr | QuantifiedExpr | TypeswitchExpr | IfExpr | OrExpr`
- `FLWORExpr ::= (ForClause | LetClause)+ WhereClause? OrderByClause? "return" ExprSingle`
- `for $d in fn:doc("depts.xml")/depts/deptno  
let $e := fn:doc("emps.xml")/emps/emp[deptno = $d]  
where fn:count($e) >= 10  
order by fn:avg($e/salary) descending  
return <big-dept> { $d,  
<headcount>{fn:count($e)}</headcount>,  
<avgsal>{fn:avg($e/salary)}</avgsal> } </big-dept>`

32

IUC 29, San Francisco, March 2006

**The core functionality of XQuery is in the FLWOR expression:** The ‘for’ and ‘let’ clauses in a FLWOR expression generate an ordered sequence of tuples of bound variables, called the ‘tuple stream’. The optional ‘where’ clause serves to filter the tuple stream, retaining some tuples and discarding others. The optional ‘order by’ clause can be used to reorder the tuple stream. The ‘return’ clause constructs the result of the FLWOR expression. The ‘return’ clause is evaluated once for every tuple in the tuple stream, after filtering by the where clause, using the variable bindings in the respective tuples. The result of the FLWOR expression is an ordered sequence containing the results of these evaluations, concatenated as if by the comma operator.

The above example of a FLWOR expression includes all of the possible clauses. The ‘for’ clause iterates over all the departments in an input document, binding the variable \$d to each department number in turn. For each binding of \$d, the ‘let’ clause binds variable \$e to all the employees in the given department, selected from another input document. The result of the ‘for’ and ‘let’ clauses is a ‘tuple stream’ in which each tuple contains a pair of bindings for \$d and \$e (\$d is bound to a department number and \$e is bound to a set of employees in that department). The ‘where’ clause filters the tuple stream by keeping only those binding-pairs that represent departments having at least ten employees. The ‘order by’ clause orders the surviving tuples in descending order by the average salary of the employees in the department. The ‘return’ clause constructs a new big-dept element for each surviving tuple, containing the department number, headcount, and average salary. [Example taking from <http://www.w3.org/TR/xquery/#doc-xquery-ForClause>]

## Collation-Terminologies

- Collation in XQuery is URI based control
  - Use URI to specify a specific collation order/algorithm
  - Can be controlled by both Implicit (default) and Explicit setting.
- Every implementation of XQuery/XPath must support at least the "Unicode codepoint collation"
  - <http://www.w3.org/2005/04/xpath-functions/collation/codepoint>
- Implementation are allow to include more to the Statically known collation
- Statically known collation
  - An implementation defined set of (URI, collation) pairs

33

IUC 29, San Francisco, March 2006

XQuery use URI to specify the collation rule. The URI identify a particular collation rule and must known to the XQuery processor. There are two ways to use the collation URI: First, we can declare a default collation URI in the prolog section and later use it implicitly for all ordering features. Or we can always pass in the URI explicitly. The second way can be seen as an overwrite mechanism too.

The set of collation URI and collation algorithm pairs known by the XQuery processor are called 'Statically Known Collation'.

In XQuery 1.0, every implementation is required to support at least one collation URI as the Statically Known Collation. It is required to support the "Unicode code point collation" that is identified by "<http://www.w3.org/2005/04/xpath-functions/collation/codepoint>".

Implementations of XQuery are allowed to support bigger set of "Statically Known Collation". For example, one of the famous open source XQuery implementation, SAXON, provide a way to extend it's 'statically known collation' to cover all the collation algorithms available in Java text package. See the following page for more details:

- <http://www.saxonica.com/documentation/conformance/collation-uri.html>
- <http://www.saxonica.com/documentation/extensions/instructions/collation.html>

## Collation-Default Collation Declaration

- Default collation
  - One of statically known collation
  - Used by comparing and ordering when no explicit collation is specified.
- Default Collation Declaration:
  - declare default collation URILiteral
  - Ex: declare default collation "http://example.org/languages/Icelandic";
  - The Unicode codepoint collation will be used if no declaration and the implementation does not specified one.

34

IUC 29, San Francisco, March 2006

The default collation in XQuery is declared in the prolog section. It use the following syntax:

```
declare default collation URI
```

For example:

```
declare default collation "http://example.org/lang/Icelandic";
```

The URI has to be one of the URI in the 'statically known collation'. If neither default collation has been declared nor the implementation does not specify a different one, then the Unicode code point collation will be used.

The declared default collation will impact the ordering in the FLWOR expression and several functions (see later section) if no explicit collation URI pass in those functions to alternate the behavior.

## Collation- Examples of FLWOR:

- In document order:
  - for \$b in \$books/book  
return \$b
- Sort by \$b/title in default collation order:
  - for \$b in \$books/book  
order by \$b/title  
return \$b
- Sort by \$b/title in the specified collation
  - for \$b in \$books/book  
order by \$b/title **collation "http://www.example.org/collations/fr-ca"**  
  
return \$b

35

IUC 29, San Francisco, March 2006

The grammar of 'order by' clause can be specified formally as below:

[38] `OrderByClause` ::= (`<"order" "by">` | `<"stable" "order" "by">`)  
`OrderSpecList`

[39] `OrderSpecList` ::= `OrderSpec` ("`,`" `OrderSpec`)\*

[40] `OrderSpec` ::= `ExprSingle` `OrderModifier`

[41] `OrderModifier` ::= (`"ascending"` | `"descending"`)? (`<"empty"`  
`"greatest">` | `<"empty" "least">`)? (`"collation"` `URILiteral`)?

We explain the 'order by' clause by using the following 5 different FLWOR expressions. In all these 5 FLWOR expressions, we assume prior to this FLWOR expression, the \$books is already bind to a fragment of XML.

- The first FLWOR expression iterate through all the 'book' elements in the \$books in the document order.
- The second FLWOR expression is very similar to the first one, except it return the 'book' element by following the order of the 'title' element in them. Since no collation is specified, it will use the default collation declared earlier. Ascending order will be used since no 'descending' is specified.
- The third FLWOR expression returns the book sorted by the collation "http://www.example.org/collations/fr-ca" in ascending order.

## Collation- Examples of FLWOR:

- Sort by \$b/title in the specified collation descending order
  - for \$b in \$books/book
  - order by \$b/title **descending** collation
  - "http://www.example.org/collations/fr-ca"
  - return \$b
- Sort by \$b/title in the specified collation descending order after ignore case
  - for \$b in \$books/book
  - order by **fn:lower-case(\$b/title)** descending collation
  - "http://www.example.org/collations/fr-ca"
  - return \$b

36

IUC 29, San Francisco, March 2006

- The forth FLWOR expression do the same thing as the third one, except it return the order in descending order.
- The fifth FLWOR expression shows one interesting way to use the 'order by' clause, you can call a function to perform preprocessing before the collation apply to an element. In this case, we perform a lower-case operation to the \$book/title. This is not an ideal way to perform case insensitive collation since it is not theoretically correct for several cases. However, since XQuery 1.0 does not specify any required case insensitive collation, this could be one of the practical ways to simulate such case insensitive collation. The real solution of case insensitive collation has to be addressed by a richer set of "statically known collation" clearly specified and required in future version of XQuery specifications.

## Collation-Functions

- `fn:default-collation()` as `xs:string`
  - Return the default collation URI
- Functions [optionally] take collation as argument:
  - `fn:compare`, `fn:contains`, `fn:starts-with`, `fn:ends-with`, `fn:substring-before`, `fn:substring-after`
  - `fn:index-of`, `fn:distinct-values`, `fn:deep-equal`, `fn:max`, `fn:min`
- Functions DO NOT take collation as argument:
  - `fn:codepoint-equal` [not impact by collation]
  - `op:QName-equal` [based on default collation]

37

IUC 29, San Francisco, March 2006

In addition to the FLWOR expression, several functions and operator in XPath/XQuery are also related by the collation.

First, the **`fn:default-collation()`** return the default collation URI as `xs:string`.

The following functions take a collation URI as optional parameter. If such parameter is not specified, then the declared default collation will be used. These functions directly or indirectly match or compare two strings internally:

**`fn:compare`, `fn:contains`, `fn:starts-with`, `fn:ends-with`, `fn:substring-before`, `fn:substring-after`, `fn:index-of`, `fn:distinct-values`, `fn:deep-equal`, `fn:max`, `fn:min`**

In the contrast to the functions above, the following two string comparison/matching functions DO NOT take optional collation URI parameter:

- The **`fn:codepoint-equal`** function do not alternate it's behavior regardless what default collation is
- The **`op:QName-equal`** operator always behave according to the default collation.

## Collation- Extension in SAXON

- Implementation (such as SAXON) are allow to support more collation
- SAXON define the following 'extension' in the format of `http://saxon.sf.net/collation?keyword=value;keyword=value;...`
  - **class**: [fully-qualified Java class name of a class that implements `java.util.Comparator`. ]
  - **lang**: [any value allowed for `xml:lang`, for example `en-US` for US English ]
  - **strength**: primary, secondary, tertiary, or identical
  - **decomposition**: none, standard, full

38

IUC 29, San Francisco, March 2006

XQuery implementations are allowed to extend the Collation support. For example, SAXON, a Java based open source implementation defined the following way to specify collation in XQuery to refer to implementation available in Java.

Summarized from

<http://www.saxonica.com/documentation/conformance/collation-uri.html>:

The SAXON define the collation URI takes the form `http://saxon.sf.net/collation?keyword=value;keyword=value;....`

The query parameters in the URI can be separated either by ampersands or semicolons, but semicolons are usually more convenient. The keywords can be '**class**', '**lang**', '**strength**', '**decomposition**'. The value of '**class**' is a fully qualified Java class name of a class that implements `java.util.Comparator`. It should not be combined with any other parameter. An instance of the requested class is created, and is used to perform the comparisons. The value of '**lang**' is any value allowed for `xml:lang`. For example: `en-US` for US English. It is used to find the collation appropriate to a Java locale. The collation may be further tailored using the parameters strength and decomposition. The value of '**strength**' is 'primary', 'secondary', 'tertiary', or 'identical'. It is used to indicate the differences that are considered significant when comparing two strings. The value of '**decomposition**' is either 'none', 'standard', or 'full'. It is used to indicate how the collator handles Unicode composed characters.

## Collation - Issues

- The minimum support in XQuery1.0 - "The Unicode Codepoint" collation is NOT good enough for user friendly operation
  - Need open specification to document collation algorithms
  - Need open specification to bind a URI to a well specified collation algorithm
  - Need a bigger set of above in the "minimum set" of future XQuery specification.
- The ability to allow implementation to extend the statically known collation provide possibility to allow user friendly operations
- However, it also may cause future interoperability issues.

39

IUC 29, San Francisco, March 2006

The XQuery 1.0 working draft currently only require every XQuery implementation support "The Unicode Code Point" collation. Such minimum support requirement is not enough to build user-friendly operation. In order to specify more standard collation rules in the future, we really need more open standard based specification that document different collation algorithm. We also need open standard specification to document how to bind a particular URI to a well-documented collation algorithm. Finally we need a bigger set of collation rules specified by the future version of XQuery specification.

Since currently the XQuery specification allows the vendor to extend and implement Collation rule, this allow us to extend the system by adding practical and useful collation algorithm and functionality. However, this will eventually become a portability and interoperability issue. An XQuery run in one XQuery processor may not be able to run in another machine if the declared collation URI is not in the "Statically Known Collation" set.

## String Functions

- **fn:concat**, **fn:string-join**, **fn:substring**, **fn:string-length**
- **fn:normalize-space**
  - Returns the value of \$arg with whitespace normalized
- **fn:normalize-unicode**
  - Returns the value of \$arg normalized according to the criteria
  - \$form is "NFC", "NFD", "NFKC", "NFKD", or "FULLY\_NORMALIZED"
  - MUST support "KFC" (also assume by default), MAY support other values.

XPath 2.0 and XQuery 1.0 include many functions. Some of them are string functions that support Unicode string handling but the behavior is designed to be independent to collation rules. **fn:concat** concatenates two or more `xdt:anyAtomicType` arguments cast to `xs:string`. **fn:string-join** returns the `xs:string` produced by concatenating a sequence of `xs:string`s using an optional separator. **fn:substring** returns the `xs:string` located at a specified place within an argument `xs:string`. **fn:string-length** returns the length of the argument. There are two string functions in XPath that contain 'normalize' in the function name. Don't get confused about what 'normalize' it refer to:

The **fn:normalize-space** function returns the whitespace-normalized value of the argument. It strips leading and trailing whitespace and replacing sequences of one or more than one whitespace character with a single space, `#x20`.

The **fn:normalize-unicode** returns the normalized value of the first argument in the normalization form specified by the second argument. This function performs the Unicode normalization. The algorithm is documented in "Character Model for the World Wide Web 1.0, Last Call Working Draft available at: <http://www.w3.org/TR/2002/WD-charmod-20020430/>". It takes an additional argument \$form to switch between different Unicode normalization forms. It is "NFC", "NFD", "NFKC", "NFKD", or "FULLY\_NORMALIZED". However, XPath/XQuery implementation is required to support NFC only.

## String Functions

- `fn:upper-case` and `fn:lower-case`
  - Unicode based, not language based conversion
- `fn:translate( $arg, $mapString, $transString )`
  - Map each characters in `$arg` to zero or one character
  - `fn:translate("abcdabc", "abc", "AB")` returns "ABdAB".
  - Can shirk string but not expand string

41

IUC 29, San Francisco, March 2006

**fn:upper-case** and **fn:lower-case** returns the upper-cased and the lower-cased value of the argument. It is a locale independent Unicode character case conversion.

**fn:translate** returns the first `xs:string` argument with occurrences of characters contained in the second argument replaced by the character at the corresponding position in the third argument. For example `fn:translate("abcdabc", "abc", "AB")` returns "ABdAB". In this example, character 'a' will map to 'A', 'b' will map to 'B', and 'c' will map to nothing, and 'd' will not be alternated. Therefore, this function can shrink the string but not expand the string.

## Regular Expression - in XML Schema

- Based on Perl Regular Expression
- Used inside `<xs:pattern value=""/>` that is included in `<xs:restriction/>` to derive data type
- Quantifiers:
  - `?`, `*`, `+`, `{n,m}`, `{n}`, `{n,}`, `{0,m}`, `{0,0}`
- Classical Perl Character Classes:
  - `\s (\S)`: spaces (same as `[\x20\t\n\r]` )
  - `\d (\D)`: digits (same as `\p{Nd}`)
  - `\w (\W)`: 'word characters'
    - (same as `[\x0000-#x10FFFF]-[\p{P}\p{Z}\p{C}]` )
  - `\i (\I)`: **XML 1.0 initial name characters**
  - `\c (\C)`: **XML 1.0 name characters**

42

IUC 29, San Francisco, March 2006

The Regular Expression facility in XPath/XQuery is based on the Regular Expression defined in the XML Schema. Therefore, we first look at the RegEx in XML Schema:

The Regular Expression in XML Schema is based on the model form Perl language. In XML Schema, the `<restriction>` element is defined to derive data type. The `<restriction>` element contains a `<pattern>` element to describe the restriction. The `<pattern>` element has a value attribute that its value is a regular expression.

The Regular Expression may have quantifier similar to what available in Perl. `'?'` for zero or once, `'*'` for zero or more times, `'+'` for once or more, `{n,m}` for at least n times and at most m times, `{n}` for exactly n times, `{n,}` for at least n times, `{0,m}` for at most m times, and `{0,0}` mean empty string.

Also, a multi-character escape provides a simple way to identify a commonly used set of characters. `'.'` is equivalent to `[\^n\r]`, `'\s'` is equivalent to `[\x20\t\n\r]`, `'\i'` match XML 1.0 initial name characters, `'\c'` match XML 1.0 name characters, `\d` match all Unicode digits and is equivalent to `\p{Nd}`, `\w` match 'word characters' and is equivalent to `[\x0000-#x10FFFF]-[\p{P}\p{Z}\p{C}]` . The upper case multi-character escape `\S`, `\I`, `\C`, `\W`, `\D` match the complement set of the lower case escape.

## Regular Expression - in XML Schema

- Hex Notation: `[abc#x4e00]`
- Range: `[#x3400-#x4dbf]`
- Union: `([#x5d]|\p{Nd})`
- Subtraction: `(\p{L}-\p{Ll})`
- Negative Character Group: `[^abc]`

43

IUC 29, San Francisco, March 2006

The Hex Notation of Unicode character is in the form of `#xhhhh`, where `hhhh` is the hex value of the Unicode code point. The range of characters can be specified by using '-' inside character class expression wrapped between '[' and ']'. For example `[#x3400-#x4dbf]` mean all the characters in the "CJK Unified Ideographs Extension A" block. The '|' is used to disjoin two class expression to form a union. For example `([#x5d]|\p{Nd})` mean either character `#x5d` or characters belong to the `\p{Nd}` class. Subtraction are represented by '-' outside the class expression (not inside [ ]). For example, `(\p{L}-\p{Ll})` mean all Unicode letters which are not upper case letters. Finally negative character group can be expressed by '^' in the beginning of the class expression within [ and ].

### References:

- Eric van der Vlist, *XML Schema- The W3C's Object-Oriented Descriptions for XML*, O'Reilly, 2002
- *XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004*, Appendix F Regular Expressions, <http://www.w3.org/TR/xmlschema-2/#dt-pattern>
- Mark Davis, Unicode Regular Expressions, Unicode Technical Standard #18, <http://www.unicode.org/reports/tr18/>, 2005

## Regular Expression -in XML Schema: Unicode Character Classes

- `\p{Name}` and `\P{Name}`: where Name is Unicode Category or "Is" + Unicode Block:
  - Unicode Category:
    - C, Cc, ..., Co, L, Ll, Lm, Lo, Lt, ..., etc.
  - Unicode Block:
    - Latin, GeneralPunctuation, Arabic, etc
- Examples:
  - `(\p{Ll}|\p{IsHiragana})*` - match all code points that are either lower case character or in the Hiragana block.
  - `(\p{IsGreek}-\p{Ll})*` - match all code points in the Greek block that are not lower case letter.

44

IUC 29, San Francisco, March 2006

Unicode Database specifies a number of possible values for the "General Category" property and provides mappings from code points to specific character properties. The set containing all characters that have property X, can be identified with a category escape `\p{X}`. The complement of this set is specified with the category escape `\P{X}`. (`[\P{X}] = [^\p{X}]`).

The 'General Category' property:

L for All Letters, Lu for uppercase, Ll for lowercase, Lt for titlecase, Lm for modifier, Lo for other letters, M for all marks, Mn for nonspacing, Mc for spacing combining, Me for enclosing, N for all numbers, Nd for decimal digit, NI for letter number, No for other number, P for all punctuation, Pc for connector, Pd for dash, Ps for open, Pe for close, Pi for initial quote, Pf for final quote, Po for other punctuation, Z for all separators, Zs for space, Zl for line separators, Zp for paragraph separators, S for all symbols, Sm for math symbols, Sc for currency symbols, Sk for modifier symbols, So for other symbols, C for all others, Cc for control, Cf for format, Co for private use, Cn for not assigned.

Block class can also be used within `\p{}` and `\P{}`. The block class always begin with 'Is' followed by the block name. For example `\p{IsBasicLatin}` and `\p{IsGreek}`.

## Regular Expression in XPath/XQuery

- Extend from XML Schema
  - Add Flags: [s/m/i/x]
  - Addition: ^ (start), \$ (end), Reluctant quantifiers (match shortest), sub-expressions (groups), back-references, etc.
- Functions:
  - fn:matches(), fn:replace(), fn:tokenize()

45

IUC 29, San Francisco, March 2006

As mentioned earlier, the Regular Expression in XPath/XQuery is an extension/modification from the XML Schema Regular Expression. First, the regular expression functions takes 'flag' to control the behaviors. It is similar to Perl: The 's' flag signal the 'dot-all' mode to control whether '.' match newline (#x0A) or not; the 'm' flag signal the multi-line mode to control how "^" and "\$" match; the 'i' flag signal the case insensitive mode; the 'x' flag ignore white space within regular expression.

The meta character ^ matches the start of the entire string (or the any line in the multi-line mode). The meta character \$ matches the end of the entire string. Reluctant quantifiers are supported and are indicated by a "?" following a quantifier. For example x??., x\*?., x+?., etc. The effect of these quantifiers is that the regular expression matches the shortest possible substring consistent with the match as a whole succeeding. Without the "?", the regular expression matches the longest possible substring.

XPath/XQuery support three regular expression functions: **fn:matches** returns an xs:boolean value that indicates whether the value of the first argument is matched by the regular expression that is the value of the second argument. **fn:replace** returns the value of the first argument with every substring matched by the regular expression that is the value of the second argument replaced by the replacement string that is the value of the third argument. **fn:tokenize** returns a sequence of one or more xs:strings whose values are substrings of the value of the first argument separated by substrings that match the regular expression that is the value of the second argument.

## Some Specific Issues

- String values versus typed values
- Time Zone handling
- IRI processing

## String Values versus Typed Values

- XQuery is a typed language
- Values (e.g. values of attributes ) may depend on user-defined schemas (in the format of XML Schema)

47

IUC 29, San Francisco, March 2006

XQuery is a typed language. For example the values of attributes may depend on user-defined schemas (in the format of XML Schema). An example is given on the following slide.

## String Values versus Typed Values

```
string-length($myDoc/myEl/revision-date@)
```

```
string-length(xs:string($myDoc/myEl/revision-  
date@))
```

- With a schema: type of @revision-date =  
xs:date
- ~~Works not!~~ **works**

48

IUC 29, San Francisco, March 2006

Attribute nodes have a string value and a typed value. The difference between the two can be seen on this slide. There are two applications of the XPath 2.0 function `string-length`. The length of the `@revision-date` attribute should be calculated. It is now assumed that there is a schema which defines `@revision-date` with the XML Schema datatype `xs:date`. With such a schema, the first example would not work, because the `string-length` function expects a string value as an input. To be able to apply this function, `@revision-date` has to be casted to a string value. This is done via a call to the function `xs:string` in the second version of `string-length`.

## Time Zone Handling

- Time related data types have an optional time zone component:

2005-07-12+07:00

- Data types mostly based on XML Schema
- The component provides the offset from UTC

49

IUC 29, San Francisco, March 2006

XQuery provides a wide range of time zone related data types like “time”, “date” or “duration”. Most of them are based on XML Schema and do have an optional time zone component. This component does not provide the timezone (which would take region-specific changes for e. g. daylight time into account), but rather an offset from Universal Central Time (UTC).

## Time Zone Handling

- No total ordering of values with / without time zone information:

2005-07-12+07:00

2005-07-12

- XQuery defines an implicit time zone to allow total ordering:

2005-07-12+07:00

2005-07-12+*[implementation defined; often UTC]*

50

IUC 29, San Francisco, March 2006

Since the time zone component of the XML Schema based time related data types is optional, it is not possible to assure a complete value ordering.

This is the case for the two values “2005-07-12+07:00” and “2005-07-12”.

To assure total ordering, XQuery provides an implicit time zone for values without a time zone component. Many implementations set UTC as this implicit component, which is recommended for interoperability.

More information on the topic can be found at in a Note by the W3C i18n Core Working Group at <http://www.w3.org/TR/timezone/> .

## IRI Handling

- “Internationalized Resource Identifiers” (RFC 3987): Normative reference in XQuery / XSLT
- Data type “anyURI” relies on XML Schema “anyURI”, still defined in terms of URI

IRI (“Internationalized Resource Identifiers”) are a normative reference in XQuery and XSLT. That is: IRI are supported in wide areas of the two languages. An aspect one has to be careful about is the data type “anyURI”, which relies on XML Schema. It is defined in terms of URI for compatibility reasons.

## Functions for IRI Processing

- Casting to IRIs with `xs:anyURI` from untyped values
- Escaping of IRIs with `iri-to-uri`:  
`iri-to-uri`  
("http://example.dürst.com")  
result:  
`http://example.d%C3%BCrst.com`

For IRI processing, three functions are of importance: `xs:anyURI` provides a casting to the “anyURI” type. “iri-to-uri” converts an IRI to an URI, which encompasses escaping of a wide range of characters.

## Functions for IRI Processing

- **Reserved Characters in iri-to-uri:**

lower case letters a-z, the upper case letters A-Z, the digits 0-9, the NUMBER SIGN "#" and HYPHEN-MINUS ("-"), LOW LINE ("\_"), FULL STOP ".", EXCLAMATION MARK "!", TILDE "~", ASTERISK "\*", APOSTROPHE "'", LEFT PARENTHESIS "(", and RIGHT PARENTHESIS ")", SEMICOLON ";", SOLIDUS "/", QUESTION MARK "?", COLON ":", COMMERCIAL AT "@", AMPERSAND "&", EQUALS SIGN "=", PLUS SIGN "+", DOLLAR SIGN "\$", COMMA ",", LEFT SQUARE BRACKET "[", RIGHT SQUARE BRACKET "]", and the PERCENT SIGN "%"

- **escape-html-uri: Escaping for HTML user agents (all except US-ASCII printable characters)**

53

IUC 29, San Francisco, March 2006

There is a set of characters which must not be escaped during the processing of iri-to-uri. Finally, The function “escape-html-uri” is provided to escape a URI in the manner HTML user agents handle attribute values that expect URI.

# Internationalization Features in XPath, XQuery and XSLT

Frank Yung-Fong Tang (Google)  
Liam Quin, Felix Sasaki (W3C)

54

IUC 29, San Francisco, March 2006