XQuery

Style

Conventions

January 17, 2006

Table of Contents

1 Introduction

1.1 Why Have Style Conventions

Style conventions are important for a number of reasons:

- Common style conventions will decrease the long-term maintenance costs associated with XQuery modules.
- Hardly any XQuery module will be maintained for its whole life by the original author.
- Style conventions improve the readability of the XQuery modules, allowing everyone to understand the XQuery more quickly and thoroughly.

The style conventions proposed in this document will not ensure the XQuery authored for one XQuery engine will work on a different XQuery engine. However, the migration to a different XQuery engine can be simplified by adhering to a consistent style when authoring XQuery modules.

Since the XQuery language is fully composable, any of the expressions can be nested within another, it is an insurmountable task to completely define style guidelines that will cover all cases. Instead, we have opted to offer general guidelines that should cover the more common scenarios.

1.2 Acknowledgments

This document reflects XQuery Style Conventions for the November 2005 W3C XQuery Language Candidate Recommendation[1]. When XQuery becomes an official recommendation and as we receive feedback from the XQuery development commnity, this document will be updated as appropriate.

Many of the concepts and guidelines contained in this document have been borrowed with permission from the *Code Conventions for the Java Programming Language*, Copyright 1995-1999 Sun Microsystems, Inc. All rights reserved. Rather than re-inventing the wheel, we felt it would be best to leverage a familiar tested and proven style that has been widely embraced.

This document is maintained by Darin McBeath.

[1] A majority of the guidelines are also applicable to prior versions of the W3C XQuery language specifications, although the XQuery language syntax and module structure may be slightly different in the older versions of the specification.

2 Module Organization

There are two basic types of module organizations for XQuery, library and main modules. The guidelines for each module type will be presented in the following sections. Appendixes A and B offer representative examples for both a main module and library module that follow the style guidelines described in this document. For complete details on XQuery main modules and library modules, we suggest reviewing the BNF for XQuery available from the W3C web site.

2.1 Library Modules

As a general rule, an XQuery library module will normally contain one or more XQuery functions and global variable declarations. All of these functions and variables should be related together in some meaningful fashion (related processing, support one another, etc.)

The XQuery library module should contain the sections identified below in the listed order.

1. XQuery version declaration
2. Beginning comments
3. Module declaration
4. Prolog

2.1.1 XQuery version declaration

All library modules should explicitly indicate the version of the XQuery specification that they adhere to as well as any optional special encoding information.

```
xquery version “1.0”;
or
xquery version “1.0” encoding “UTF-8”;
```

2.1.2 Beginning comments

All library modules should begin with an XQuery-style comment that lists the module name, version information, date, copyright, any vendor specific proprietary extension dependencies, XQuery specification and a module overview. See section 4 for more information on XQuery commenting style.

```
(:
 : Module Name:  Sample Library Module
 : Module Version:  1.0
 : Date:  January 6, 2006
 : Copyright:  xqDoc.org …
 : Proprietary XQuery Extensions Used:  None
 : XQuery Specification:  November 2005
 : Module Overview:  This is a sample library module …
 :)
```

2.1.3 *Module declaration*

All libraries must provide a module namespace declaration.

```
module namespace pfx=”some-uri”;
```

2.1.4 *Prolog*

The following table defines the recommended ordering of items in the prolog if they exist.

Item	**Notes**
Imports	Any schema imports followed by module imports.
Namespace Declarations	The default function and element namespaces should precede any other namespace declarations.
Static Context Declarations	Static context declarations should appear in the following order: • BaseURI • Copy Namespaces • Boundary Space • Default Collation • Ordering Mode • Empty Order • Construction
Option Declarations	Implementation specific options.
Variable Declarations	Global variables.
Function Declarations	User defined functions.

2.2 Main Modules

As a general rule, an XQuery main module will normally contain just a query body. While the W3C XQuery specification allows for the definition of global variables and functions within a main module, this practice is generally not recommended since it prevents the reusability of these functions and variables by other library and main modules[2]. However, since not all XQuery engines support the ability to import modules, it may become necessary in those situations to define functions and global variables within a main module. One exception to this general rule is that global variables marked as 'external' should only be defined in a main module.

The XQuery main module should contain the sections identified below in the listed order.

1. XQuery version declaration
2. Beginning comments
3. Prolog
4. Query body

2.2.1 XQuery version declaration

All main modules should explicitly indicate the version of the XQuery specification that they adhere to as well as any optional special encoding information.

```
xquery version “1.0”;
or
xquery version “1.0” encoding “UTF-8”;
```

2.2.2 Beginning comments

All main modules should begin with an XQuery-style comment that lists the module name, version information, date, copyright, any vendor specific proprietary extension dependencies, XQuery specification and module overview.

```
(:
 : Module Name:  Sample Main Module
 : Module Version:  1.0
 : Date:  January 6, 2006
 : Copyright:  xqDoc.org …
 : Proprietary XQuery Extensions Used:  None
 : XQuery Specification:  November 2005
 : Module Overview:  This is a sample main module …
 :)
```

[2] For an exception to this general rule, please consult the best practices section (9.9 Hidden Functions) of this document for when it is advisable to declare functions within a main module.

2.2.3 Prolog

The following table defines the recommended ordering of items in the prolog if they exist.

Item	Notes
Imports	Any schema imports followed by module imports.
Namespace Declarations	The default function and element namespaces should precede any other namespace declarations.
Static Context Declarations	Static context declarations should appear in the following order: • BaseURI • Copy Namespaces • Boundary Space • Default Collation • Ordering Mode • Empty Order • Construction
Option Declarations	Implementation specific options.
Variable Declarations	As a general rule, global variables should not be defined in a main module. However, if the XQuery engine does not support the ability to import modules, it may be necessary to define global variables in a main module. One exception to this rule is that global variables marked as 'external', should only be defined in a main module.
Function Declarations	As a general rule, functions should not be defined in a main module. However, if the XQuery engine does not support the ability to import modules, it may be necessary to define functions in a main module.

2.2.4 Query body

The Query body contains the XQuery expression(s) to evaluate.

3 Indentation

A consistent value should be used for the default unit of indentation to provide a clear indication of nesting and readability. The exact construction of the indentation (spaces vs. tabs) is unspecified.

3.1 Line Length

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

Note: Examples for use in documentation should have a shorter line length—generally no more than 70 characters.

3.2 Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing XQuery or to XQuery that's squished up against the right margin, just use the default indentation value.

If no breaks are necessary, the function call should look as follows:

```
some-function(expression1, expression2)
```

Here are some examples of breaking function calls:

```
some-function(longExpression1,
              longExpression2)

some-function(longExpression1, longExpression2,
    longExpression3, longExpression4, longExpression5)

some-function(longExpression1,
    another-function(longExpression2,
        longExpression3))
```

Following are examples of breaking an arithmetic expression. The first is preferred, since the break occurs outside the parenthesized expression, which is at a higher level.

```
let $longName1 := longName2 * (longName3 + longName4 - longName5)
                  + 4 * longname6

let $longName1 := longName2 * (longName3 + longName4
                  - longName5) + 4 * longname6
```

Following are two examples of indenting function declarations. The first is the conventional case. The second would shift the second and third lines to the far right if it used conventional indentation, so instead it indents by the default indentation value. The body of the function should also be indented by the default indentation value.

```
(: CONVENTIONAL INDENTATION :)


declare function some-function(arg1 as item(), arg2 as item(),
                               arg3 as item(), arg4 as item())
as item()
{
...
};

(: INDENT TO AVOID VERY DEEP INDENTS :)


declare function some-unction(arg1 as item(),
    arg2 as item(),
        arg3 as item())
as item()
{
    let $x := “test”
    ….
};
```

Line wrapping for `if` statements should generally use double the default indentation value, since the default indentation value might make seeing the body difficult. For example:

```
(: DON’T USE THIS INDENTATION :)

if ((condition1 and condition2)
    or (condition3 and condition4)
    or (condition5 and condition6)) then
    do-something-about-it()
else
    ()

(: USE THIS INDENTATION INSTEAD :)

if ((condition1 and condition2)
        or (condition3 and condition4)
        or (condition5 and condition6)) then
    do-something-about-it()
else
    ()

(: OR USE THIS :)

if ((condition1 and condition2) or (condition3 and condition4)
        or (condition5 and condition6)) then
    do-something-about-it()
else
    ()
```

3.3 Sequences

When specifying a long sequence of XQuery expressions, place the first expression following the opening parenthesis and list each subsequent expression on a separate line. Put the closing parenthesis on the same line and following the last expression.

```
(expression1,
 expression2,
 …
 expressionn)
```

When specifying a short sequence of XQuery expressions, place the first expression following the opening parenthesis and list each subsequent expression on the same line delimited by a comma and space. Put the closing parenthesis on the same line and following the last expression.

```
(expression1, expression2)
```

3.4 Embedded XQuery

When embedding XQuery statements within XML or XHTML markup, the following conventions should be followed. The opening and closing curly brackets should follow and precede the corresponding markup tags and the initial XQuery statement be indented by the default indentation value from the markup tag. After that, the rules described above should be followed for the XQuery expression.

```
<tag>{
    if (condition) then
        do-something-about-it()
    else
        ()
}</tag>
```

If the XQuery statement is simple, it is also acceptable to include the statement on the same line as the markup tags. In this situation, the opening and closing curly brackets should follow and precede the corresponding markup tags and the XQuery statement should be delimited by a single space from the curly brackets.

```
<tag>{ $tag }</tag>
```

4 Comments

Note: See "XQuery Source File Examples" contained in Appendix A and B for examples of the comment formats described here.

XQuery modules can have two kinds of comments: implementation comments and documentation comments. Implementation comments are those found in the W3C XQuery Specification, are delimited by `(: ... :)`. Documentation comments (known as "doc comments") have been defined by *xqdoc.org* and are delimited by `(:~ ... :)`. Doc comments can be extracted to HTML files using the xqDoc tool.

Implementation comments are means for commenting out sections of a module or for comments about the particular implementation. Doc comments are meant to describe the specification of the module, from an implementation-free perspective to be read by others who might not necessarily have the module at hand.

Comments should be used to give overviews of the module and provide additional information that is not readily available in the module itself. Comments should contain only information that is relevant to reading and understanding the module.

4.1 Implementation Comment Formats

Modules can have three styles of implementation comments: block, single-line, trailing and end-of-line.

4.1.1 Block Comments

Block comments are used to provide descriptions of modules, functions, variables and algorithms. Block comments may be used at the beginning of each module and before each function. They can also be used in other places, such as within functions. Block comments inside a function should be indented to the same level as the XQuery fragment they describe.

A block comment should be preceded by a blank line to set it apart from the rest of the code. If multiple lines are needed, begin each subsequent line with a ":" and vertically align the colons.

```
(:
 : Here is a block comment.  It is a rather long comment to provide an
 : example for a multi-line block comment.
 :)
```

4.1.2 Single-Line Comments

Short comments can appear on a single line indented to the level of the XQuery that follows. If a comment can't be written in a single line, it should follow the block comment format (see section 4.1.1). A single-line comment should be preceded by a blank line. Here's an example of a single-line comment XQuery:

```
if (condition) then

    (: Handle the condition. :)
    ...
else
    ()
```

4.1.3 Trailing Comments

Very short comments can appear on the same line as the XQuery they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of XQuery, they should all be vertically aligned.

Here’s an example of a trailing comment in XQuery:

```
if (condition) then
    some-function()            (: a comment :)
else
    another-function()         (: a comment :)
```

4.2 Documentation Comments

xqDoc comments should be used to document XQuery library and main modules in a manner similar to how Javadoc comments are used to document Java classes. With the documentation close to the source, it increases the chances that the documentation will be kept current and with tools provided by xqDoc, useful documentation can be quickly and easily generated.

xqDoc style comments begin with a '(:~' and end with a ':)' Like Javadoc, the following tags have special meaning within an xqDoc comment. In addition, the values provided for each of these tags can contain embedded XHTML markup to enhance or emphasize the xqDoc XHTML presentation. However, make sure the content is well formed and that entities are used (i.e. &amp; instead of &). The beginning text up to the first xqDoc tag (i.e. @author) in a xqDoc comment block is assumed to be description text for the component being documented.

@author

The @author tag identifies the author for the documented component. Zero or more @author tags can be specified (one per author)

@author Darin McBeath

@version

The @version tag identifies the version of the documented component. Zero or more @version tags can be specified (one per version) but in reality only a single @version tag would normally make sense. The value for the @version tag can be an arbitrary string.

@version 1.0

@since

The @since tag identifies the version when a documented component was supported. Zero or many @since tags can be specified, but in reality only a single @since tag would normally make sense. The value for the @since tag can be an arbitrary string but should likely match an appropriate version value.

@since 1.0

@see

The @see tag provides the ability to hypertext link to an external web site, a library or main module contained in xqDoc, a specific function (or variable) defined in a library or main module contained in xqDoc, or arbitrary text. To link to an external site, use a complete URL such as http://www.xquery.com. To link to a library or main module contained in xqDoc, simply provide the URI for the library or main module. To link to a specific function (or variable) defined in an xqDoc library or main module, simply provide the URI for the library or main module followed by a ';' and finally the function or variable name. To provide a name for a link, simply include a second ';' followed by the name. To provide text, simply include the 'text'. Multiple @see tags can be specified (one per link or string of text).

@see http://www.xquery.com
@see http://xqdoc.org/xqdoc/xqdoc-display
@see http://xqdoc.org/xqdoc/xqdoc-display;build-link
@see http://xqdoc.org/xqdoc/xqdoc-display;$months
@see http://xqdoc.org/xqdoc/xqdoc/xqdoc-display;$months;month variable
@see http://www.xquery.com;;xquery
@see some text

@param

The @param tag identifies the parameters associated with a function. For each parameter in a function, there should be a @param tag. The @param tag should be followed by the parameter name (as indicated in the function signature) and then the parameter description.

@param $name The username

@return

The @return tag describes what is returned from a function. Zero or one @return tags can be specified.

@return Sequence of names matching the search criteria

@deprecated

The @deprecated tag identifies the identifies the documented component as being deprecated. The string of text associated with the @deprecated tag should indicate when the item was deprecated and what to use as a replacement.

@deprecated As of 1.0 and replaced with add-user

@error

The @error tag identifies the types of errors that can be generated by the function. Zero or more @error tags can be specified. An arbitrary string of text can be provided for a value.

@error The requested URI does not exist

A representative library module xqDoc comment is included below. This comment would precede the module declaration statement for the library module.

```
(:~
 : This module provides the functions that control the Web presentation
 : of xqDoc. The logic contained in this module is not specific to any
 : XQuery implementation.
 : It should also be noted that these functions not only support the
 : real-time presentation of the xqDoc information but are also used
 : for the static offline presentation mode as well. The static offline
 : presentation mode has advantages because access to a native XML
 : database is not needed when viewing the xqDoc information ... it is
 : only needed when generating the offline materials.
 :
 : @author Darin McBeath
 : @version 1.0
 :)
 module namespace xqdoc="http://xqdoc.org/xqdoc/display"
```

A representative library module xqDoc function comment is included below. Note how embedded XHTML markup can be used within an xqDoc description to enhance the presentation when the xqDoc comments are viewed. This comment would precede the function declaration statement in the library module.

```
(:~
 : The controller for constructing the xqDoc HTML information for
 : the specified library module. The following information for
 : each library module will be generated.
 : <ul>
 : <li> Module introductory information</li>
 : <li> Global variables declared in this module</li>
 : <li> Modules imported by this module</li>
 : <li> Summary information for each function defined in the module</li>
 : <li> Detailed information for each function defined in the module</li>
 : </ul>
 :
 : @param $uri the URI for the library module
 : @param $local indicates whether to build static HTML link for offline
 : viewing or dynamic links for real-time viewing.
 : @return XHTML.
 :)
 declare function xqdoc:print-module($uri as xs:string, $local as xs:boolean)
 as element()*
 {
```

For further details about doc comments and xqDoc, see the xqDoc home page at: `http://xqdoc.org`

As a general rule, if you need to give information about a module, import, variable, or function that isn't appropriate for documentation, use an implementation block comment (see section 4.1.1) or single-line (see section 4.1.2) comment immediately *after* the declaration. For example, details about the implementation of a module should go in such an implementation block and not in the module doc comment.

5 Declarations

5.1 Local variable declarations

One local variable declaration (and assignment) per let clause is recommended since it helps with clarity. For example,

```
let $givenName := “Darin”
let $surname   := “McBeath”
```

is preferred over

```
let $givenName := “Darin”,
    $surname   := “McBeath”
```

5.2 Function declarations

When authoring a function, the following rules should be followed:

- No space between the function name and the parenthesis “(“ starting its parameter list.
- Follow conventions listed in section 3.2 for dealing with large parameter lists.
- The return type should appear on a separate line and vertically align with the declare function statement.
- The opening brace “{“ should appear on the line following the return type and vertically align with the declare function statement.
- XQuery expression(s) within the function should follow the indentation conventions described in section 3.
- The closing brace and semi-colon “};” should appear on a separate line and vertically align with the declare function statement.

```
declare function some-function(arg1 as item(),
                               arg2 as item())

as item()
{
    let $givenName := “Darin”
    …
};
```

6 Expressions

There are a number of expressions defined in the XQuery specification. Since XQuery is a fully composable language (specifically since expressions can be nested within one another), it is a difficult task to completely cover all scenarios. Instead, the following sections highlight the recommended format for a handful of expressions that are frequently used when authoring XQuery and offer some general guidelines.

6.1 if-then-else expressions

A general rule to follow is that the '`if`', '`else`', '`else if`' associated with the same '`if`' clause should all be on a separate line and vertically aligned. The expression associated with the true or false condition should be indented (as described in section 3). The `if-then-else` class of expressions should have one of the following forms:

```
if (condition) then
   …
else
   …


if (condition) then
   …
else if (condition) then
   …
else
   …
```

For simple and short `if-then-else` expressions, it is also acceptable to include the entire expression on a single line. In this scenario, the `if-then-else` expression should have the following form:

```
if (condition) then … else …
```

6.2 FLWOR expressions

The `FLWOR` expression is more complex than the basic if-then-else expression. A general rule to follow is that the '`let`', '`for`', '`where`', '`order by`', and '`return`' clauses associated with the same `FLWOR` expression should all be on a separate line and vertically aligned. The expressions associated with each of these clauses should be on the same line as the clause or on the following line indented by the default indentation value (if the expression is lengthy and fairly complex). A `FLWOR` expression embedded within another `FLWOR` expression's '`return`' or '`where`' clause should be indented by the default indentation value (on the following line). The `FLWOR` class of expressions should have one of the following forms:

```
let $set := (3,2,1)
for $i in $set
return $i


let $set := (3,2,1)
for $i in $set
where $i > 1
order by $i
return $i


let $set1 := (3,2,1)
let $set2 := (1,2,3)
for $x in $set2
for $y in $set1
return $x + $y



let $value := “some value”
return $value


let $set1 := (3,2,1)
let $set2 := (1,2,3)
for $x in $set2
for $y in $set1
return
    if ($x = $y) then true() else false()


let $set := (3,2,1)
for $x in $set
return
    for $j in $set
    return
        if ($j = $x) then true() else false()
```

6.3 Typeswitch expressions

The `typeswitch` expression is another more complex expression. A general rule to follow is that the ‘`typeswitch`’ (and expression) should be on a separate line. Each ‘`case`’ (and type) or ‘`default`’ should be on a separate line indented by the default indentation value. Lastly, each ‘`return`’ should be on a separate line indented by an additional default indentation value. The expression associated with the ‘`return`’ should be on the same line or on the following line indented by the default indentation value (if the expression is lengthy and fairly complex). The `typeswitch` class of expressions should have the following form:

```
typeswitch($customer/billing-address)
    case $a as element(*,USAddress)
        return $a/state
    case $a as element(*,CanadaAddress)
        return $a/province
    case $a as element(*,JapanAddress)
        return $a/prefecture
    default
        return “unknown”
```

7 White Space

7.1 Blank Lines

Blank lines improve readability by setting off sections of XQuery that are logically related.

Two blank lines should always be used in the following circumstances:

- Between the main sections defined in 2.1 and 2.2.
- Between the groups of like items defined in a prolog for a module.

One blank line should always be used in the following circumstances:

- Between items declared in a prolog (i.e. functions, namespace declarations, etc.)
- Between the local variables in a function and subsequent statements
- Before a block (see section 4.1.1) or single-line (see section 4.1.2) comment
- Between logical sections inside a function to improve readability

```
xquery version "1.0";


(: Beginning Comments :)


module namespace math="http://xqdoc.org/sample-math-lib";


declare default function namespace "http://www.w3.org/2005/xpath-functions";

declare default element namespace "http://www.w3.org/1999/xhtml";

declare function some-function($operand1 as xs:integer,
                               $operand2 as xs:integer)
as xs:integer
{
…
};
```

7.2 Blank Spaces

Blank spaces should be used in the following circumstances:

- A keyword followed by a parenthesis should be separated by a space. Example:

```
if (true()) then
    ...
else
    ()
```

Note that a blank space should not be used between a function name and its opening parenthesis. This helps to distinguish keywords from function invocations.

- A blank space should appear after commas in argument lists.

```
some-function(parm1, parm2, parm3)
```

- A blank space should appear after commas in sequence lists.

```
let $a := (item1, item2, item3)
```

- All arithmetic operators should be separated from their operands by spaces.

```
let $result := 1 + 2
```

8 Naming Conventions

Naming conventions make modules more understandable by making them easier to read. They can also give information about the purpose of the identifier—for example, whether it's a global variable, local variable, or a function name — which can be helpful in understanding the module.

Identifier Type	Rules for Naming	Example
Global Variable (not requiring run time evaluation)	Should be uppercase with hyphens separating each word	$FAILURE-CODE $LIST-OF-URIS
Global Variable (requiring run time evaluation)	Should be lowercase with hyphens separating each word	$country-list
Local Variable	Should be mixed camel case with a lower case first letter	$city $intermediateResult

Function Name	Should be verbs in lower case, with hyphens separating each word	produce-report()
		calculate-result()

Namespace Prefix	Should be lower case alpha characters and less than 8 characters in length.	display lib

9 Best Practices

Best practices help with the ongoing maintenance associated with library and main modules. Like naming conventions, if these practices are followed, then there will be consistency across the code base which can help team members in their understanding of a new module.

9.1 Predefined Namespaces

The following prefixes (and URIs) should not be used for any user-defined functions. The use (for your own purposes) of the URIs listed below is prohibited by the specification and will result in a static error. The use (for your own purposes) of the prefixes listed below is discouraged. The exception to this rule is 'local' … this should be used if you find it necessary to define functions within a main module.

xml = http://www.w3.org/XML/1998/namespace

xs = http://www.w3.org/2001/XMLSchema

xsi = http://www.w3.org/2001/XMLSchema-instance

fn = http://www.w3.org/2005/xpath-functions

xdt = http://www.w3.org/2005/xpath-datatypes

local = http://www.w3.org/2005/xquery-local-functions

9.2 Constants

Numeric and string literal value constants should be declared as global variables and not explicitly specified in an XQuery expression 'let' clause.

```
declare variable $COUNTRY as xs:string { "United States of America" };
```

Similarly, global variables requiring run time evaluation should not be defined in an XQuery expression 'let' clause.

```
declare variable $country-list as xs:string { doc('countries.xml')/country };
```

9.3 Default Function Namespace

Do not additionally declare a namespace prefix associated with the same URI for the default function namespace and interchangeably use the prefix (and no prefix) within the same module. This implies that it is not a good practice to create a default function namespace associated with the same URI of a given library module where the default function namespace is declared.

Don't do the following:

```
declare default function namespace "http://www.w3.org/2005/xpath-functions";

declare namespace func="http://www.w3.org/2005//xpath-functions";

….

concat($var1, $var2)
```

```
…

func:concat($var1, $var2)
```

It is generally a good rule to never specify another default function namespace other than the XPath Functions and Operators namespace. The use of a different namespace for the default function namespace only causes confusion amongst XQuery developers. Similarly, if this rule is followed, one should never use the 'fn' prefix when referencing functions defined in the XPath Functions and Operators namespace. It is more natural to use a prefix when invoking user defined functions and not use a prefix when invoking functions defined in the default function namespace.

9.4 Default Element Namespace

Do not additionally declare a namespace prefix associated with the same URI for the default element namespace and interchangeably use the prefix (and no prefix) within the same module. This will only be a source of confusion for others trying to understand an XQuery module.

Don't do the following:

```
declare default element namespace "http://www.w3.org/1999/xhtml";

declare namespace xhtml="http://www.w3.org/1999/xhtml";

<html>
    <xhtml:body>
    …
</html>
```

9.5 Function Types and Ordinality

Whenever possible, use the most constraining type possible when defining return types for functions. Since XQuery supports subtype substitution (but not supertype substitution) it's usually a good rule to choose a more primitive base type for parameters when appropriate. For example, if a function parameter was declared as xs:positiveInteger and the invoking module was using xs:integer, the xs:integer would need to be cast to xs:positiveInteger before invoking the function. Always specify the most constraining number of occurrences for each type whenever possible. A type should always be specified for every parameter and the return type.

Don't do the following:

```
declare function some-function($operand1,
                               $operand2)
{
…
};
```

But, instead do the following:

```
declare function some-function($operand1 as xs:string,
                               $operand2 as xs:string?)
as xs:string*
{
…
};
```

9.6 XPath and FLWOR

Whenever possible, use an XPath expression (with a predicate) instead of a FLWOR expression. Since this will result in a compact and clear expression, the XQuery module will not be cluttered with unnecessary XQuery statements.

Don’t do the following:

```
for $x in collection(“dogs”)/breed
where $x/name = “spot”
return
    $x
```

But, instead do the following:

```
collection(“dogs”)/breed[name=”spot”]
```

9.7 Library Module Namespace

A library module should not use the XPath F&O namespace for the module namespace (see 9.3).

```
module namespace mod="http://www.w3.org/2005/xpath-functions";
```

Instead**,** a library module namespace should be modelled after the namespace structure used in the source XML schema. Likewise, the collection of functions declared in the library module should be associated with the elements and types defined in the corresponding schema.

For example, the XML schema associated with xqDoc has the following namespace

```
http://xqdoc.org/xqDoc/xml
```

And the XQuery library module that manipulates this xqDoc XML has the following namespace

```
http://xqdoc.org/xqDoc/display
```

9.8 Variable name reuse

Reuse of local variable names is acceptable. This will eliminate the unnecessary specification of unique variable names for intermediate expressions.

Don’t do the following:

```
let $x := some-function()
let $y := another-function($x)
```

But, instead do the following:

```
let $x := some-function()
let $x := another-function($x)
```

However, do not reuse a variable name that is used within both a global and local scope as this will only cause confusion as to what variable will be in scope for the expression. If the rules for global variable and local variable names are followed, this should not be an issue.

```
declare variable $x as xs:string { “something” );
for $x in (1 to 5)
return $x
```

9.9 Hidden functions

Functions should be declared in a main module only when the intent is to hide the implementation of the particular function. By placing the function declaration in a main module, one is essentially making the function ‘private’ and preventing other library modules or main modules from reusing the function.

Appendix A – Sample Main Module

The following is a representative XQuery main module that adheres to the style conventions outlined in this document.

```
xquery version "1.0";


(:
 : Module Name:       Math Main Module
 :
 : Module Version:    1.0
 :
 : Date:              01/17/2006
 :
 : Copyright:         Sample Copyright here
 :
 : Proprietary
 : Extensions:        None
 :
 : XQuery
 : Specification     November 2005
 :
 : Module Overview:  This sample main module illustrates the
 :                   proper implementations of the XQuery
 :                   Style Guidelines.
 :)

(:~
 :  This module provides a sample main module authored according
 :  to the proposed XQuery Style Guidelines
 :
 :  @author Curt Kohler
 :  @author Darin McBeath
 :  @since January 17, 2006
 :  @version 1.0
 :)

import module namespace math="http://xqdoc.org/sample-math-lib" at "sample-math-lib.xqy";


declare default function namespace "http://www.w3.org/2005/xpath-functions";

declare default element namespace "http://www.w3.org/1999/xhtml";


(:~
 :  This variable contains the first variable to process.
 :)
declare variable $operand-one as xs:integer external;

(:~
 :  This variable contains the second integer to process.
 :)
declare variable $operand-two as xs:integer external;


<html>
  <head>
    <title>A Math Example</title>
  </head>
  <body>
    <p>The following is an XQuery math example:</p>
    <hr/>
    <p>Operand 1: { $operand-one }</p>
    <p>Operand 2: { $operand-two }</p>
    <p/>
    <hr/>
    <p>Sum is: { math:add-operands($operand-one, $operand-two) }</p>
    <p>Difference is: { math:subtract-operands($operand-one, $operand-two) }</p>
    <p>Product is: { math:multiply-operands($operand-one, $operand-two) }</p>
    <p>Quotient is: {
        if ($operand-two = 0) then
            <b>I’m sorry, you can’t divide by zero.</b>
        else
            math:divide-operands($operand-one, $operand-two)
    }</p>
  </body>
</html>
```

Appendix B – Sample Library Module

The following is a representative XQuery library module that adheres to the style conventions outlined in this document.

```
xquery version "1.0";


(:
 : Module Name:       Math Library Module
 :
 : Module Version:    1.0
 :
 : Date:              01/17/2006
 :
 : Copyright:         Sample Copyright here
 :
 : Proprietary
 : Extensions:        None
 :
 : XQuery
 : Specification      November 2005
 :
 : Module Overview: This sample main module illustrates the
 :                  proper implementations of the XQuery
 :                  Style Guidelines for a Library Module.
 :)


(:~
 :  This module provides a sample library module of simple math
 :  functions that are authored according to the proposed XQuery
 :  Style Guidelines
 :
 :  @author Curt Kohler
 :  @author Darin McBeath
 :  @since January 17, 2006
 :  @version 1.0
 :)
module namespace math="http://xqdoc.org/sample-math-lib";


declare default function namespace "http://www.w3.org/2005/xpath-functions";

(:~
 :  This variable contains the verion of the sample generic math
 :  library implemented in this module.
 :)
declare variable $math:MATH-VERSION as xs:string { "1.0" };

(:~
 :  Add two numbers.
 :
 :  @param $operand1   first operand.
 :  @param $operand2   second operand.
 :  @return result of the addition operation.
 :)
declare function add-operands($operand1 as xs:integer,
                              $operand2 as xs:integer)
as xs:integer
{
    let $sum := $operand1 + $operand2  (: add the values :)
    return
        $sum
};

(:~
 :  Subtract two numbers.
 :
 :  @param $operand1   first operand.
 :  @param $operand2   second operand.
 :  @return result of the subtraction operation.
 :)
declare function subtract-operands($operand1 as xs:integer,
                                   $operand2 as xs:integer)
as xs:integer
{
    let $difference := $operand1 - $operand2  (: subtract the values :)
    return
        $difference
};

(:~
 :  Multiply two numbers.
 :
 :  @param $operand1   first operand.
 :  @param $operand2   second operand.
 :  @return result of the multiplication operation.
 :)
declare function multiply-operands($operand1 as xs:integer,
                                   $operand2 as xs:integer)
as xs:integer
{
    let $product := $operand1 * $operand2  (: multiply the values :)
    return
       $product
};

(:~
 :  Divide two numbers.
 :
 :  @param $operand1   first operand.
 :  @param $operand2   second operand.
 :  @return result of the division operation.
 :)
declare function divide-operands($operand1 as xs:integer,
                                 $operand2 as xs:integer)
as xs:integer
{
    let $quotient := $operand1 div $operand2  (: divide the values :)
    return
       $quotient
};
```