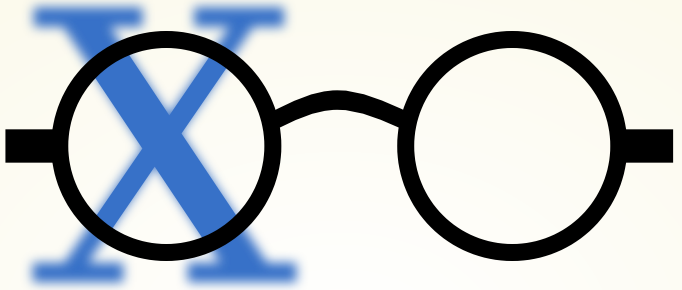
Introduction to XSpec



Amanda Galtman

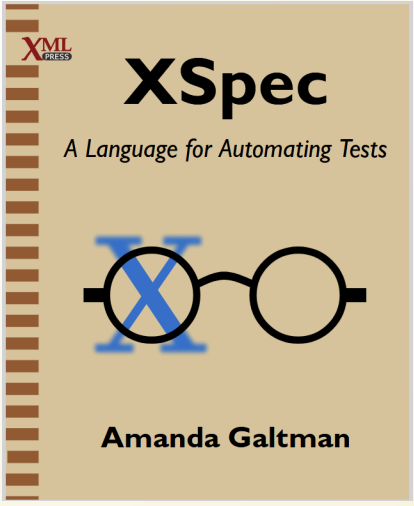
November 2026

Agenda

- Background
- High-level testing procedure
- How to run XSpec tests
- Explore some XSpec tests
  - XSLT, XQuery, XProc, Schematron
- Along the way...
  - Verification, failure reporting, whitespace

My XSpec Background

- Developing/testing XSLT, XQuery, Schematron
  - Technical pubs
- Contributing to XSpec project since 2018
  - Maintaining since 2022
- medium.com/@xspectacles
- *Soon:* Book from XML Press!



Your Background

- You use one or more of
  - XSLT, XQuery, Schematron, or XProc 3
- You might be new to
  - XSpec
  - Automated testing
- Or, you use XSpec and want to learn it more systematically

XSPEC BACKGROUND

What Is XSpec?

- Open-source software for testing
  - XSLT
  - XQuery
  - Schematron
  - XProc (XSpec v4.0, 2026)
- Created by Jeni Tennison, 2009
- Latest XSpec version is 4.1

Parts of XSpec

1. **XML vocabulary** to express
    - How to call the code you're testing, including inputs
    - What you expect to happen
2. **Software** to automate running tests, i.e.,
    - Calling
    - Verifying: Do results meet expectations?
    - Reporting pass/fail outcomes

This tutorial covers both parts.

XSpec Can Help You...

- Detect regressions
  - Ensure quality in a scalable way
- Use behavior-driven development process
  - BDD not required, though
- Document software behavior
- Investigate defects

BDD Process

Scenarios guide software design/development.

1. **Describe** a desired future behavior, concretely
2. **Execute** the description (test fails)
3. **Code** only enough to make it pass
4. **Refactor** to improve code design, reduce duplication, etc.
    - Maintain passing status
5. **Repeat** for another behavior

HIGH-LEVEL TESTING PROCEDURE

Terminology: Test Target

Section of code you're testing, e.g.,

- XSLT template or function
- XQuery function
- XProc step
- Schematron assertion, report, or rule

Terminology: Test Case

Specific instructions for doing two things:

1. Exercise test target to produce some data
    - "Call" → actual result
2. Verify that actual result meets expectations
    - Pass/fail/pending

(Custom setup/cleanup aren't part of XSpec.)

Testing Procedure

You write instructions. XSpec follows them.

1. Exercise test target to produce some data
   - "Call" → actual result
2. Verify that actual result meets expectations
   - Pass/fail/pending

Repeat as needed.

HOW TO RUN XSPEC TESTS

Two Ways, for this Tutorial

- Choose: Command script or Oxygen
- Both require up-front installation/configuration
  - https://codeberg.org/galtm/xspec-tutorial/

(Other ways to run tests are not covered here.)

Demo

EXPLORE XSPEC TESTS

Basic Structures

Identical for XSLT and Schematron,...

```
1 <x:description xmlns:x="http://www.jenitennison.com/xslt/xspec"
2     stylesheet="identity.xsl">
3     <x:scenario label="When the input is 1">
4         <!-- scenario content -->
5     </x:scenario>
6 </x:description>
```

```
1 <x:description xmlns:x="http://www.jenitennison.com/xslt/xspec"
2   schematron="got-here.sch">
3   <x:scenario label="The input reaches a 'Got here' message">
4     <!-- scenario content -->
5   </x:scenario>
6 </x:description>
```

...as well as XQuery and XProc.

Scenarios Can Nest

<x:scenario> is for organizing code, too.

```
 1 <x:scenario label="When section has a title">
 2     <x:scenario label="and a subtitle">
 3         <!-- scenario content -->
 4     </x:scenario>
 5     <x:scenario label="and no subtitle">
 6         <!-- scenario content -->
 7     </x:scenario>
 8 </x:scenario>
 9 <x:scenario label="When section has no title">
10     <!-- scenario content -->
11 </x:scenario>
```

Inner Scenarios Can Inherit

Ex: Define variable, use in inner scenarios

```
1 <x:scenario label="When section has a title">
2     <x:variable name="my:title" select="//title"/>
3     <x:scenario label="and a subtitle">
4         <!-- scenario content using $my:title -->
5     </x:scenario>
6     <x:scenario label="and no subtitle">
7         <!-- scenario content using $my:title -->
8     </x:scenario>
9 </x:scenario>
```

EXPLORE XSPEC TESTS: XSLT/XQUERY FUNCTIONS

An Example Function

`eg:attr-ids`

- Input: One XML node
- Output: String values of `@id` and `@xml:id`
  - Self + descendants
  - Zero or more
  - Sequence of values

Recall the Two-Step Instructions

1. Exercise test target to produce some data
    - "Call" → actual result
2. Verify that actual result meets expectations
    - Pass/fail/pending

How does this look when you test a function?

Elements

1. `x:call` with `function` attribute
    - "Call" → actual result
2. `x:expect`
    - Choices of attributes/content
    - Verify that actual result meets expectations

Same elements for XSLT function vs. XQuery

Demo

Exercise 1: Function

1. Save a copy of `example1/find-ids_xslt_before_exercise.xspec`
    - In same directory, so file paths still work
2. Remove `pending="EXERCISE"`
3. Fill in the scenario labeled `"with multiple id attributes"`
4. Run the test

Pause

EXPLORE XSPEC TESTS: XSLT TEMPLATES

Testing Template vs. Function

1. Exercise test target... is *different*
2. Verify... is *the same*

Look at how to invoke templates in XSpec...

Named Template

```
1 <x:call template="attr-ids">
2   <x:param name="tree">
3     <root id="a1"/>
4   </x:param>
5 </x:call>
```

- `x:call` element with `template` attribute
- `x:param` requires `name` attribute
  - Order isn't well-defined

Template Rule

```
1 <x:context mode="attr-ids">
2   <x:param name="p1">xyz</x:param>
3   <root id="a1"/>
4 </x:context>
```

- `x:context` element instead of `x:call`
  - "Call" → actual result
  - Get `xsl:apply-templates` behavior
- Choices of attributes/content, as in `x:param`
- Optional `mode` attribute
- Optional `x:param` with `name` attribute

Named Template with Context

```
1 <x:context>
2   <root id="a1"/>
3 </x:context>
4 <x:call template="context-attr-ids">
5   <x:param name="p1">xyz</x:param>
6 </x:call>
```

- Optional `x:context` followed by `x:call`
- Template parameters go in `x:call`

Exercise 2: Template

1. From `example1/` folder, open `find-ids.xsl` to see templates:
    - Named template with parameter
    - Template rule
    - Named template with context item
2. Save a copy of `find-ids_xslt_after_exercise.xspec`
3. Replace `x:call` with XSpec code for testing one of the templates

(Solutions in `example1/exercise2` folder)

Pause

MORE ABOUT VERIFICATION AND REPORTING

Three x:expect Patterns

- Match against actual result (deep-equal)
  - You provide expected result
  - Content or `href`; `select`
- Check that boolean condition is true
  - No expected result
  - `test` = XPath for true/false
- Match against data derived from actual result
  - `test` = XPath for derivation
  - Filter, normalize, pick subtree, etc.
  - You provide expected result of derivation

Example 2

```
1 <xsl:template match="/" as="element(html)">
2   <html>
3     <head>
4       <title>HTML document</title>
5     </head>
6     <xsl:apply-templates select="." mode="body"/>
7   </html>
8 </xsl:template>
```

- Produces HTML element tree
- Imagine that result of `xsl:apply-templates` is verified elsewhere
- Can you verify outer `html` element and `head`?

Demo

EXERCISES 3 AND 4

Exercise 3: Verification

1. Save a copy of `example2/make-html_xslt_before_exercise3.xspec`
2. Verify outermost element name in two ways
    - Boolean `test`
    - `test` deriving something from `$x:result`
        - Verify local name only, for simplicity

(Solutions in `example2/exercise3` folder)

Pause

Exercise 4: Report

1. Open `example2/exercise4/make-html_xslt_failures.xspec`
2. Run the test
    - Has deliberate failures
3. Examine the HTML test report to see how it describes the failures
    - How are they different?
    - Are some easier to understand?

Pause

EXPLORE XSPEC TESTS: XPROC STEPS

An Example Step

`eg:make-html`

- `source` input: JSON document
- `body-id` option: String to use as `id`
- `result` output: HTML document containing
  - `head` with `title`, text from the JSON file
  - `body` created by another step

Similar to the XSLT template rule in `example2/`

Markup for Testing XProc

- `x:call` element with `step` attribute
- `x:input` element with `port` attribute
  - Use like `x:param`
  - Or, embed `p:document`
- `x:option` element with `name` attribute
- `x:expect` with optional `port` attribute

Demo

Exercise 5: XProc Verification

1. Save a copy of `example2/make-html_xproc_before_exercise5.xspec`
2. Fill in the `x:expect` element labeled `"with a head element containing a title"`
3. Run the test

   (Solution in `example2/exercise5` folder)

Pause

A WHITESPACE TANGENT

Not XProc-specific

Demo

EXPLORE XSPEC TESTS: SCHEMATRON

Schematron Testing Concept

- "Call" = Validate a document
- Verify that the validation process
  - Triggers the assertions/reports you expect
  - Not those you don't expect
  - Fires rules you expect

Test with invalid documents *and* valid ones

Schematron Implementations

XSpec v4.0+ distribution includes:

- SchXslt2, XSLT-based
  - https://codeberg.org/schxslt/schxslt2
- XQS, XQuery-based
  - https://github.com/AndrewSales/XQS

Pick based on schema's `queryBinding` attribute (e.g., `"xslt3"` vs. `"xquery31"`).

An Example Schema

- If parent and child both have `id` attribute, the child's id must start with the parent's id.
  - e.g., `parent1` and `parent1-child1`
- Every `id` attribute value must be at least 5 characters

Markup for Testing Schematron

- `x:context` similar to testing XSLT
- `x:param` child of `x:description`

Markup for Testing Schematron (cont'd.)

- `x:expect-assert`, `...-report`
  - Text content
  - Attributes `id`, `role`, `location`, `count`
- `x:expect-not-assert`, `...-report`
  - Attributes `id`, `role`, `location`
- `x:expect-rule`
- `x:expect-valid`

HTML Report

- Automatic verification labels
  - Preceded by optional custom label
- Failures expose SVRL document

Demo

Exercise 6: Schematron

1. Save a copy of `example3/id-validation_before_exercise6.xspec`
2. Test the assertion that has `id="length"`
    - `"id is 5 characters"` should *not* trigger this assertion
    - `"id is 4 characters"` *should* trigger this assertion
3. Run the test

(Solution in `example3/exercise6` folder)

Pause

XSPEC HAS MORE

No time now to cover...

- Global XSLT parameters
- Three-dot syntax
- Testing for errors
- XSpec variables
- Reusing XSpec code
- Context item in `@test` (XSLT)
- Document properties (XProc)
- Wrapping in document nodes (XProc)
- Non-XML content outside XProc
- ...and more

Other Resources

- XSpec wiki: Official documentation
  - Presentations and Demos: Conference talks, etc.
- XSpec book: Paperback, PDF, ePub

Image Credits

Slide 1: XSpec logo is from XSpec repository

```
The MIT License

Copyright (c) 2008-2017 Jeni Tennison

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

Slide 2: Draft book cover is from XML Press

THANK YOU

Go forth and write high-quality code!

