# Magic URLs in an XML Universe

# Contents

# Magic URLs in an XML Universe

## Introduction

George Bina

oXygen XML Editor

george@oxygenxml.com

Syncro Soft

@georgebina

Hi! Good afternoon! My name is George Bina, I am one of the founders of Syncro Soft, the company that develops oXygen XML Editor. I am happy to be present again this year at XML London and to have the chance to introduce you to some hopefully interesting and useful ideas.

My talk is about being able to see content in a way that is different from accessing that in the usual way, and achive that by pointing to that resource with a "magic" URL.

## Overview

- The problem

  We will begin with a description of the problem that we are trying to solve.
- Proposed solution

  Then we will describe the solution we want to propose to solve the problem.
- Technical details

  We will then go into some technical details
- Examples

  followed by some examples
- Conclusions

  and we will end with some conclusions and ideas for the future.

## The problem

### The XML Universe!

We have a lot of tools for processing XML content and we may already have pre-set processing workflows. It will be great if we can bring information from the outside of our XML world very easily into these existing workflow and take advantage of existing processing as well as other functionality that we have already built in.

For example, if we look at DocBook or DITA, we have publishing processes that allow us to get different output formats, including PDF, HTML, WebHelp, EPUB and so on. Also, they provide some additional functionality based on XML, or on top of XML, like reuse and linking mechanisms.

It will be great if everyone can agree on a common XML based format, and use that for everything, but that is not really possible, and for sure not in the immediate future.

So the problem that we are trying to solve, is how can we get resources outside the XML Universe to work seemlessly in our XML processing workflow, to be part of our XML Universe!

### API Documentation

One example of a non-XML format is API documentation that stays within source files - for example, if we look at Java, that will be Java comments and JavaDoc annotations that are added to describe different Java components like classes, methods and fields. We cannot expect these to be replaced with XML simply because these formats are already supported by many tools and there is no clear benefit or reason why someone will want that. On the other hand, if we want to write a tutorial we would like to reuse some of the existing content without copy/paste type of reuse.

### Spreadsheet Data

Another example is access to speadsheet data. If you setup a nice speadsheet that uses formulas to compute interesting information, you may just want to use that in a deliverable that is generated from an XML source - a more concrete example, if you want to respond to a work request and you want to include a cost estimate - if you just fill in your speadsheet document with some data and you get everything computed, it will be great to be able to include easily that information in a proposal document that you write in DITA or DocBook or some other XML language, again, without copy/pasting values from that speadsheet into your XML content.

### How can we bring non-XML resources (Java, speadsheets, etc.) into our XML Universe?

So, to summarize, the problem that we are trying to solve is how can we bring non-XML resources like what I mentioned above, Java, speadsheets but also Markdown, CSV files, and so on, into our XML Universe so we can include them into our XML-based processing workflows.

## The idea

### Magic URLs

Use URLs to dynamically convert content, point to a resouce and see that as something else:

**See a Java class as a DITA topic**

```
java2dita:/[URL pointing to a Java class]
```

**See an Excel sheet as a DITA topic**

```
excel2dita:/[URL pointing to an Excel file]
```

### URLs

```
https://www.example.com/path/cgi?param=value
```

```
zip:URL!/path/to/file.ext
```

There is a lot of processing involved also for these URLs!

### URLs in Java

**Pluggable** support for **custom URLs** - just register an URL handler

## The "convert" URLs

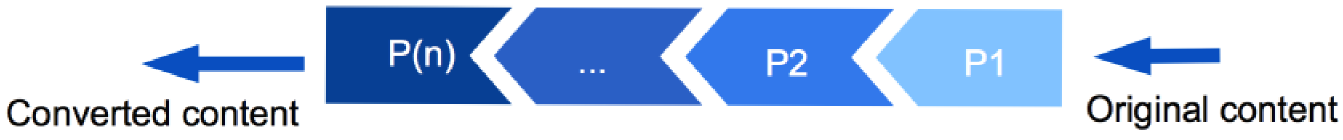
### Too many Magic URLs

Develop a new Java URL handler for each magic URL is a little too much...

Thus we propose:

- one *flexible* URL, the "convert" URL scheme
- use XML Catalog mappings to revrite each type of "convert" url in a short form

### Format



```
convert:/pipelineStepN/.../pipelineStep1!/targetContentURL
```

### Pipeline Processors

**XSLT**

```
processor=xslt;ss=convert.xsl;param=value
```

**XQuery**

```
processor=xquery;ss=convert.xquery;p1=v1
```

**Java**

```
processor=java;jars=urn:jars;ccn=j.to.xml.JavaToXML
```

**JavaScript**

```
processor=js;js=urn:processors:md.js;fn=convert
```

**Excel to XML**

```
processor=excel;sn=test
```

**JSON to XML**

```
processor=json
```

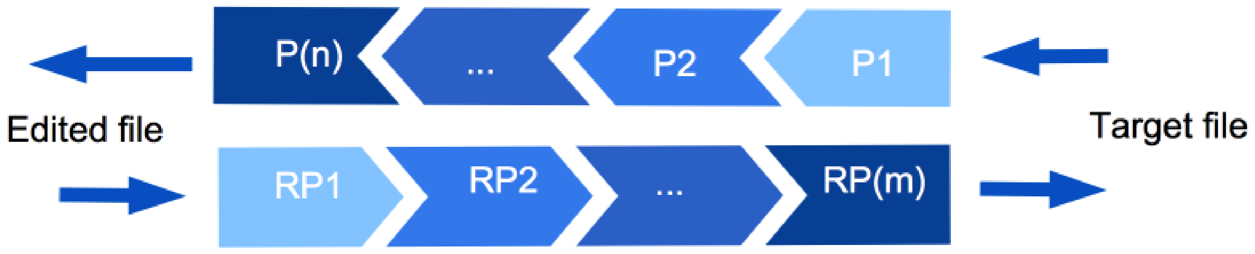
**HTML to XHTML**

```
processor=xhtml
```

**Wrap text**

```
processor=wrap;rn=wrapperElement
```

### Direct and Reverse Conversion Pipelines



```
convert:/pipelineStepN/.../pipelineStep1!/
```

```
          reverseStep1/.../reverseStepM/targetContentURL
```

### Reverse Processing

Use the same processors as the direct conversion (XSLT, XQuery, Java, JavaScript, Excel, HTML, JSON, Wrap)

Use **`rprocessor`** instead of **`processor`** in the pipeline definition

### "convert" URLs as Magic URLs

Use XML Catalogs to compress a convertion pipeline to a URL scheme

```
<rewriteURI uriStartString="excel2dita:/"
rewritePrefix="
convert:/processor=xslt;ss=urn:e2d.xsl/processor=excel;sn=s!/
"/>
```

## Sample Magic URLs

### Excel to DITA

DITA topic ← XSLT(excel2dita.xsl) ← Excel to XML ← Excel file

**Excel to DITA URL**

```
excel2dita:/urn:files:sample.xls
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="excel2dita:/"
 rewritePrefix="convert:/
 processor=xslt;ss=urn:processors:excel2d.xsl/
 processor=excel;sn=sample!/"/>
```

### Google Sheets to DITA

DITA topic ← XSLT(XHTML to DITA) ← XSLT(filter) ← XHTML ← GoogleSheet

**Google Sheets to DITA URL**

```
gs2dita:/https://docs.google.com/spreadsheets/.../edit
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="gs2dita:/"
 rewritePrefix="convert:/
 processor=xslt;ss=urn:processors:h2d.xsl/
 processor=xslt;ss=urn:processors:googleSheets2dita.xsl/
 processor=xhtml!/"/>
```

## HTML to DITA

DITA topic ⟵ XSLT(XHTML to DITA) ⟵ XHTML ⟵ HTML

**HTML to DITA URL**

```
html2dita:/urn:files:care.html
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="html2dita:/"
 rewritePrefix="convert:/
 processor=xslt;ss=urn:processors:h2d.xsl/
 processor=xhtml!/"/>
```

## Markdown to DITA

DITA topic ⟵ XSLT(XHTML to DITA) ⟵ XHTML⟵ JavaScript (MD to HTML) ⟵ Markdown

**Markdown to DITA URL**

```
md2dita:/urn:files:sample.md
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="md2dita:/"
 rewritePrefix="convert:/
 processor=xslt;ss=urn:processors:h2d.xsl/
 processor=xhtml/
 processor=js;
   js=urn:processors:pagedown%2FMarkdown.Converter.js;
   fn=convertExternal!/"/>
```

## XML Schema to DITA

DITA topic ⟵ XSLT(serialization) ⟵ XSLT(XSD to DITA) ⟵ XML Schema

**XSD to DITA URL**

```
xsd2dita:/urn:files:xsd/personal.xsd
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="xsd2dita:/"
 rewritePrefix="convert:/
 processor=xslt;ss=urn:processors:copyAsTopic.xsl/
 processor=xslt;ss=urn:processors:xsdToTopic.xsl!/"/>
```

## Java to DITA

DITA topic ← XSLT (XML to DITA) ← Java (Java source to XML) ← Java

**Java to DITA URL**

```
java2dita:/urn:files:java/WSEditorBase.java
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="java2dita:/"
 rewritePrefix="convert:/
 processor=xslt;ss=urn:processors:javaToTopic.xsl/
 processor=java;jars=urn:processors:jars;ccn=j.to.xml.JavaToXML!/"/
>
```

## Javadoc to DITA

DITA topic ← XSLT(XHTML to DITA) ← XHTML ← Javadoc

**Javadoc to DITA URL**

```
javadoc2dita:/urn:files:javadoc/ro/sync/ecss/component/editor/
ButtonEditor.html
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="javadoc2dita:/"
 rewritePrefix="convert:/
 processor=xslt;ss=urn:processors:javaDocToTopic.xsl/
 processor=xhtml!/"/>
```

## Sales XML to SVG

SVG ← XSLT(Salex XML to SVG) ← Sales XML

**Sales XML to SVG URL**

```
sales2svg:/urn:files:sales.xml
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="sales2svg:/"
 rewritePrefix="
 convert:/processor=xslt;ss=urn:processors:sales.xsl!/"/>
```

## CSV to DITA and DITA to CSV (Round-tripping)

DITA topic ← XSLT(Wrapped CSV to DITA) ← Wrap ← CSV

DITA topic → XSLT(DITA to CSV) → CSV

**CSV to DITA URL**

```
csv2dita:/urn:files:sample.csv
```

**Catalog mapping**

```
<rewriteURI
 uriStartString="csv2dita:/"
 rewritePrefix="convert:/
 rprocessor=xslt;ss=urn:processors:dita2csv.xsl/
 processor=xslt;ss=urn:processors:csvtext2dita.xsl/
 processor=wrap!/"/>
```

## DITA Map to Schematron

## Conclusions

Simple idea but powerful

Works automatically with URL-aware applications

A generic approach, works to dynamically convert between any two formats

Allows implementing single sourcing across formats

Samples available at *https://github.com/oxygenxml/dita-glass*

## Thank you!

**Question?**

### Contact information

george@oxygenxml.com

@georgebina