

Develop Python/XML with 4Suite, Part 5

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. Tutorial introduction and preparation	2
2. Building the sample application	5
3. The Web interface.....	13
4. Indices and updates	21
5. Wrap-up and packaging.....	29
6. Resources	36

Section 1. Tutorial introduction and preparation

What this tutorial covers

This intermediate-level tutorial walks developers through the process of using 4Suite, an open-source XML processing toolkit for Python developers, to create a practical Web application: a manager of software information records. With this Web application, users can manage a database of software descriptions and related vendors using a standard XML format. The development of the sample application demonstrates how to do these tasks:

- Add software records and update the RDF database according to rules that you set
- Experiment with the command-line and Web-based query interfaces and other operations
- View documents from the Web, including triggering XSLT transforms from Web queries
- Query the RDF database
- Write server-side XSLT scripts to update the repository based on Web form input
- Package the application

Who should take this tutorial?

Web and Web-based application developers, who are turning to XML for improved information management can use XML repositories as the anchor for data processing and storage. The 4Suite XML toolkit for Python developers includes repository features for managing XML and RDF data in a form that is easily integrated into the Web.

If you are interested in using XML, XSLT, RDF, and Python for developing Web applications, then you'll benefit from the practical examples of these technologies used to develop a Web-based demonstration application. Following the examples here will help you rapidly develop your own applications using similar techniques.

Prerequisites

This intermediate tutorial assumes that you are already familiar with XML, XSLT, and RDF technologies, and their implementations in 4Suite. You can get all the necessary background in the previous four parts of this tutorial series.

You'll also need to know how to initialize the repository and how to use its basic features. The 4Suite repository [Quick Start document](http://4suite.org/docs/QuickStart.xml) (<http://4suite.org/docs/QuickStart.xml>) covers this information.

The examples use the [Versa](http://uche.ogbuji.net/tech/rdf/versa/) query language for RDF (<http://uche.ogbuji.net/tech/rdf/versa/>). You should be familiar with the basics of Versa. There is an good introduction, "[RDF Query using Versa](http://www.ibm.com/developerworks/xml/library/x-think10/index.html)", right here on IBM developerWorks (<http://www.ibm.com/developerworks/xml/library/x-think10/index.html>). After this, review the [Versa tutorial](http://uche.ogbuji.net/tech/rdf/versa/versa-by-example.txt) (<http://uche.ogbuji.net/tech/rdf/versa/versa-by-example.txt>). Lessons 1 through 10 are sufficient for this tutorial, though you might want to read all the lessons for maximum comfort with Versa.

The [Resources](#) on page 36 section of this tutorial gives links to developerWorks tutorials and other useful resources that cover the prerequisites.

Preparing the environment

You will need to install Python and 4Suite.

- [Python 2.2](http://www.python.org/download) (<http://www.python.org/download>). Version 2.1 of Python will also work, but we recommend version 2.2.
- [4Suite 0.12.0a3 or more recent](http://4suite.org) (<http://4suite.org>). The older version, 0.11.1 will not work for this tutorial, and 0.12.0 releases prior to alpha 3 have bugs that may affect the operation of the examples.

Detailed instructions for installing 4Suite are available online for [UNIX](http://4suite.org/docs/howto/UNIX.xml) (<http://4suite.org/docs/howto/UNIX.xml>) or [Windows](http://4suite.org/docs/howto/Windows.xml) (<http://4suite.org/docs/howto/Windows.xml>).

To follow the examples after you have Python and 4Suite installed, you must initialize the repository and configure it. The installation documents come with sample configuration files for 4Suite. (You can also use the online [configuration helper](http://uche.ogbuji.net/tech/4Suite/?xslt=config-helper.xslt) (<http://uche.ogbuji.net/tech/4Suite/?xslt=config-helper.xslt>) to generate a customized configuration file based on a few simple questions.) Run **4ss agent**, as described in the 4Suite Repository [Quick Start guide](http://4suite.org/docs/QuickStart.xml) (<http://4suite.org/docs/QuickStart.xml>). Also follow the Quick Start guide in setting up a scratch server port 8080 and a container named **web** (a container is much like a folder or directory). Follow the guide to sign on as a regular user, rather than as a superuser.

Then take a few moments to learn the basic repository features from the Quick Start document. To follow the examples in this tutorial, you must initialize the repository and grasp the basics of how it works before you begin the tutorial.

All the code presented in this tutorial is available for download in the [xsa-app.zip](#) package.

Getting help and finding out more

For technical questions about the content of this tutorial, contact the authors, Uche Ogbuji (uche.ogbuji@fourthought.com) and Chimezie Ogbuji (chimezie@ogbuji.net).

Uche Ogbuji is a computer engineer, co-founder, and Principal Consultant at [Fourthought, Inc.](http://Fourthought.com) (<http://Fourthought.com>). He has worked with XML for several years, co-developing [4Suite](http://4Suite.org) (<http://4Suite.org>), an open source platform for XML and RDF development in [Python](http://python.org) (<http://python.org>). He writes articles on XML for IBM developerWorks, LinuxWorld, SunWorld, and XML.com. Mr. Ogbuji is a Nigerian immigrant living in Boulder, Colorado, USA.

Chimezie Thomas-Ogbuji is a software consultant for Fourthought, and he is a co-developer of 4Suite. He enjoys writing and developing computer games in his spare time. He also researches artificial intelligence techniques.

Section 2. Building the sample application

What is the 4Suite repository?

You might think of the 4Suite repository as an XML and RDF database management system (DBMS). The repository supports the loading and management of XML files and RDF statements, including query support and interfaces to the Web, FTP, WebDAV, Web services, and the like.

You can access the repository from the command line, from Python or XSLT extension APIs, or from a Web-based management console in 4Suite called the **Dashboard**.

The 4Suite repository runs on the same platforms as 4Suite, and comes bundled with 4Suite starting with version 0.12.0.

The sample application

With the sample Web-based application, users can manage a database of software descriptions and related vendors using XML Software Autoupdate, a standard XML format. XSA is just one example of the many XML formats that can be stored and processed in 4Suite. In the process of creating, testing, and packaging the sample application -- if you follow along with the examples -- you'll find out how to use the 4Suite repository features and some basic RDF querying techniques.

XSA: XML Software Autoupdate

XSA is an XML data format designed by Lars Marius Garshol for listing and describing information about software packages. The following is a simple XSA record:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xsa
  PUBLIC "-//LM Garshol//DTD XML Software Autoupdate 1.0//EN//XML"
  "http://www.garshol.priv.no/download/xsa/xsa.dtd">
<xsa>
  <vendor>
    <name>Fourthought, Inc.</name>
    <email>info@fourthought.com</email>
    <url>http://fourthought.com</url>
  </vendor>
  <product id="4suite">
```

```
<name>4Suite</name>
<version>0.12.0</version>
<last-release>20020630</last-release>
<info-url>http://4Suite.org</info-url>
<changes>
- Merge the core library and repository/server
- Add bundled demos
</changes>
</product>
</xsa>
```

This file is available as **4suite.xml** in the package for this tutorial (see [Resources](#) on page 36).

XSA field descriptions

Element name	Description
xsa	The document element contains one <code>vendor</code> and one or more <code>product</code> elements.
vendor	Information about the vendor of the products listed in the file: name, e-mail address, and Web address.
product	Information about a particular product. The <code>id</code> attribute is a local identifier. Child elements give the name, version, date of current release, Web address, and list of changes since the previous release.

4Suite document definitions

The first step in creating the application is to specify a document definition for XSA records. Document definitions create a set of rules for processing certain documents in the 4Suite repository. These include the schema type and location for validation, whether to maintain a full-text index of associated documents, and mappings for updating the repository's RDF database with information from the XML file. The RDF mappings feature is the only one demonstrated in this tutorial.

One form of RDF mapping uses a set of triples of XPath expressions to create corresponding RDF triples in the repository database. Document definitions that use this approach are called *XPath document definitions*.

With 4Suite, you can also use XSLT for document definitions, but this tutorial doesn't show how to use them.

A document definition for XSA

This listing shows an XPath document definition for XSA for use with 4Suite.

```
<!DOCTYPE DocDef [  
  <!ENTITY xsa "http://www.garshol.priv.no/download/xsa/">  
  <!ENTITY dc "http://purl.org/dc/elements/1.1/">  
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">  
<DocDef xmlns="http://xmlns.4suite.org/reserved" xmlns:rdf="&rdf;">  
<RdfMappings>  
  <!-- Map the URI of the document being added or documented  
    to the vendor: the URL identifies the vendor -->  
  <RdfMapping>  
    <Subject>$uri</Subject>  
    <Predicate>'&xsa;vendor'</Predicate>  
    <Object type="rdf:Resource">/xsa/vendor/url</Object>  
  </RdfMapping>  
  <!-- Set the RDF type of the vendor -->  
  <RdfMapping>  
    <Subject>/xsa/vendor/url</Subject>  
    <Predicate>'&rdf:type'</Predicate>  
    <Object type="rdf:Resource">'&xsa;Vendor'</Object>  
  </RdfMapping>  
  <!-- Map the vendor to its name -->  
  <RdfMapping>  
    <Subject>/xsa/vendor/url</Subject>  
    <Predicate>'&xsa;name'</Predicate>  
    <Object type="rdf:Literal">/xsa/vendor/name</Object>  
  </RdfMapping>  
  <!-- Map the vendor to the products in the file:  
    the URL identifies the product -->  
  <RdfMapping>  
    <Subject>/xsa/vendor/url</Subject>  
    <Predicate>'&xsa;distributed'</Predicate>  
    <Object type="rdf:Resource">/xsa/product/info-url</Object>  
  </RdfMapping>  
  <!-- Map each product to its name -->  
  <RdfMapping>  
    <Subject>/xsa/product/info-url</Subject>  
    <Predicate>'&xsa;name'</Predicate>  
    <Object type="rdf:Literal">/xsa/product/name</Object>  
  </RdfMapping>  
  <!-- Set the RDF type of the product -->  
  <RdfMapping>  
    <Subject>/xsa/product/info-url</Subject>  
    <Predicate>'&rdf:type'</Predicate>  
    <Object type="rdf:Resource">'&xsa;Product'</Object>  
  </RdfMapping>  
  <!-- Map each product to its version -->  
  <RdfMapping>  
    <Subject>/xsa/product/info-url</Subject>  
    <Predicate>'&xsa;version'</Predicate>
```

```
<Object type="rdf:Literal"/>/xsa/product/version</Object>
</RdfMapping>
<!-- Map each product to its change info -->
<RdfMapping>
  <Subject>/xsa/product/info-url</Subject>
  <Predicate>'&xsa;changes'</Predicate>
  <Object type="rdf:Literal"/>/xsa/product/changes</Object>
</RdfMapping>
<!-- Map each product to its last release date -->
<RdfMapping>
  <Subject>/xsa/product/info-url</Subject>
  <Predicate>'&xsa;release'</Predicate>
  <Object type="rdf:Literal"/>/xsa/product/last-release</Object>
</RdfMapping>
</RdfMappings>
</DocDef>
```

This file is available as **xsa.dd** in the package for this tutorial (see [Resources](#) on page 36).

Explaining the document definition

The `RdfMappings` element wraps a series of mappings, each of which has three child elements that represent XPath expressions. The expressions are evaluated to become the corresponding part of an RDF statement. For example, open up the file in [XSA: XML Software Autoupdate](#) on page 5 in a different browser window and examine the following excerpt from the document definition:

```
<RdfMapping>
  <Subject>/xsa/vendor/url</Subject>
  <Predicate>'&xsa;name'</Predicate>
  <Object type="rdf:Literal"/>/xsa/vendor/name</Object>
</RdfMapping>
```

The `Subject` expression is a path expression that evaluates to `http://fourthought.com`. The `Predicate` expression is a simple string: The only matter of note is the entity reference, which indicates that this string actually expands to `http://www.garshol.priv.no/download/xsa/name`. We made up all the RDF predicates to start with `&xsa;` based on Garshol's home page for XSA. The object, which is specified as an RDF literal type rather than a resource, comes from another expression that evaluates to `Fourthought, Inc.`

Creating multiple RDF statements with one mapping

If an XPath expression in the RDF mappings in a document definition returns a node

set with multiple entries, then the processor creates multiple RDF statements. The created statements make up every possible combination of values from evaluating the subject, predicate, and object expressions. So, for instance, if the file contains two products, the processing yields two statements.

If the subject XPath expression results in a node set with string values [s1, s2], the predicate expression results in [p1, p2] and the object expression in [o1, o2]. In this case, the following RDF triples result:

- {s1, p1, o1}
- {s2, p1, o1}
- {s1, p2, o1}
- {s2, p2, o1}
- {s1, p1, o2}
- {s2, p1, o2}
- {s1, p2, o2}
- {s2, p2, o2}

This set of triples is known as the *Cartesian product* of each node set generated.

Loading up the repository

From this point in the tutorial, you'll need an initialized repository. If you have not already completed the setup, then go back to [Preparing the environment](#) on page 3 to find out how to:

- Initialize it according to the 4Suite Repository [Quick Start guide](#)
- Set up a scratch server port 8080 and a container named **web**
- Sign on as a regular user (not as a superuser).

Add the document definition to the repository using the command line tools. The first command creates a container for the XSA material. The second command adds the document definition to that container.

```
4ss create container /web/xsa
4ss create document --type=xpathdocdef xsa.dd /web/xsa/xsa.dd
```

The `--type` option tells 4Suite that this is a special document that's being added -- specifically, an XPath document definition. Then comes the local file with the document to be added, and then the path of the added document within the repository -- in this case, within the top-level `xsa` folder.

The following command adds the XSA instance document for 4Suite:

```
4ss create document --docdef=/web/xsa/xsa.dd 4suite.xml /web/xsa/4suite.xml
```

The `--docdef` parameter tells 4Suite to assign the given document definition to the document that you're adding. You don't need a `--type` parameter this time; the default is plain XML.

You can also add these documents using the 4Suite Dashboard.

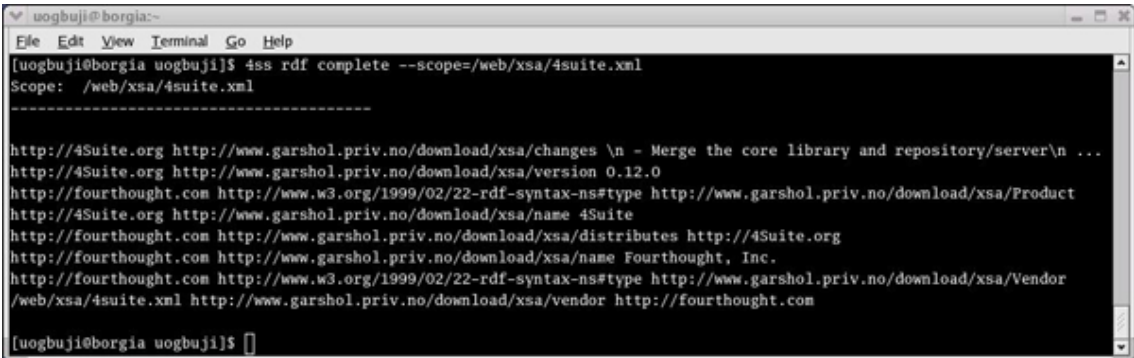
The generated RDF on the command line

So far you have added a document assigned to a document definition with RDF mappings. Remember, this means that RDF statements are automatically added to the repository RDF database according to the rules in the document definition. You can view the generated RDF triples by using the following command:

```
4ss rdf complete --scope=/web/xsa/4suite.xml
```

The `4ss rdf complete` command without any arguments reports all statements in the repository database as triples. You may remember the term **complete** from the 4RDF part of this tutorial series (see Part 3 of the series in [Resources](#) on page 36). By adding the `--scope`, you narrow it down to reporting just the triples associated with a particular path in the repository. All statements added or modified automatically because of RDF mappings are placed in a scope whose ID is the path of the particular instance document. If that instance document is deleted from the repository, all the statements within its associated scope are also cleared.

The output that results from the above command is:



```
uogbuji@borgia:~$ 4ss rdf complete --scope=/web/xsa/4suite.xml
Scope: /web/xsa/4suite.xml
-----
http://4Suite.org http://www.garshol.priv.no/download/xsa/changes \n - Merge the core library and repository/server\n ...
http://4Suite.org http://www.garshol.priv.no/download/xsa/version 0.12.0
http://fourthought.com http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.garshol.priv.no/download/xsa/Product
http://4Suite.org http://www.garshol.priv.no/download/xsa/name 4Suite
http://fourthought.com http://www.garshol.priv.no/download/xsa/distributes http://4Suite.org
http://fourthought.com http://www.garshol.priv.no/download/xsa/name Fourthought, Inc.
http://fourthought.com http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.garshol.priv.no/download/xsa/Vendor
/web/xsa/4suite.xml http://www.garshol.priv.no/download/xsa/vendor http://fourthought.com
uogbuji@borgia uogbuji$
```

If you run this command yourself, you might see the statements in a different order

than that displayed above. There is no ordering of RDF statements in models, and so there is no guarantee that the statements will be retrieved and sorted in any particular way.

The part before the first space on each line is the subject. The part between the first and second space is the predicate, and the remainder is the object. New lines are rendered as `\n` and long objects are truncated and marked with ellipsis.

The generated RDF in graph form

Note: The steps in this panel are optional. You can complete this tutorial without following them, but they do illustrate a useful tool.

You can also view the RDF generated from the XSA document graphically by using **Triclops**, a special system for interactively querying the repository RDF database. The simplest way to access Triclops is through the 4Suite Dashboard. To use Triclops, access the URL `http://localhost:8800` and then click **Metadata Search**. The Triclops form appears.

In the top text area of Triclops, enter the following text: `@"/web/xsa/4suite.xml"`. This is a simple query in Versa, an RDF query language that comes with 4Suite. This command means "give me the resource identified by `/web/xsa/4suite.xml`."

To generate the graph, from the Results drop-down menu, select **RDF Graph** and then click **Execute Metadata Query**.

One note: You may see some properties in the RDF graph beyond those that came strictly from the document definition. That is because the Triclops graph does not limit itself by scope, as you learned how to do in [The generated RDF on the command line](#) on page 10 . Therefore, you'll see in the graph some of the system metadata that 4Suite keeps about each resource (last modified date and the like). One of the metadata items you'll see is a record of the assigned document definition. This can be a useful RDF property to query for, say, finding all documents that have been added using a particular definition.

Command-line experimentation

The following table illustrates some of the things you can do with the document from the command line once it's in the repository. Feel free to experiment. You may want to delete and recreate the document after you've finished fooling around.

You can use the `-h` option with any command to access help for the command. To see all the commands available, use `4ss --show-commands`. (Note: Each command in the left-hand column of the table should be entered on one line; it is split here for viewing.)

Full command	Explanation
<code>4ss fetch document /web/xsa/4suite.xml</code>	Prints the contents of the document.
<code>4ss fetch resource /web/xsa/4suite.xml</code>	Prints just the system information, or metadata, about the document.
<code>4ss set acl /web/xsa/4suite.xml</code>	Sets ownership and access control information on the resource from an interactive prompt.
<code>4ss render /web/xsa/4suite.xml /ftss/data/decorated-xml.xslt</code>	Applies an XSLT transform from the repository to a source document in the repository. In this case it uses a utility style sheet that comes with 4Suite that renders XML documents in a syntax-highlighted view. Pipe the output to HTML and view in a Web browser.
<code>4ss update document 4suite.xml /web/xsa/4suite.xml</code>	Updates the contents of the repository resource. Use this, for instance, if you have changed the source file and want these changes reflected in the repository. Note: You do not have to repeat options such as <code>--type</code> and <code>--docDef</code> on an update, unless you want to change these properties from what you used on creation.
<code>4ss delete 4suite.xml</code>	Deletes the resource from the repository.
<code>4ss fetch container /xsa</code>	Displays all the resources in the top-level xsa container.

Section 3. The Web interface

Basics

The next step is to work on the Web with the files you have created.

Make sure you have a scratch server running (usually on port 8080), as mentioned in [Preparing the environment](#) on page 3 , and make sure the repository is started. If you have set up the scratch server on a different port, adjust your code from the examples to whatever port you've chosen.

You can then view the XSA document added in [Building the sample application](#) on page 5 by browsing `http://localhost:8080/xsa/4suite.xml`.

You can also have the repository apply a style sheet and then display the results. To do this, you need to use a special URL parameter, which is `xslt` by default. Try the following URL:

```
http://localhost:8080/xsa/4suite.xml?xslt=/ftss/data/decorated-xml.xslt
```

It displays the document after applying the style sheet `/ftss/data/decorated-xml.xslt`, which, as you may recall, is a utility style sheet that comes with 4Suite and renders XML documents in a syntax-highlighted view.

A style sheet for displaying XSA

As a general utility style sheet, `decorated-xml` can work for any XML file. This panel presents **`xsa.xslt`**, available in the package for this tutorial (see [Resources](#) on page 36), which is specialized to give an HTML view of XSA records.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:param name='show-products' select='"yes"' />

  <xsl:template match="xsa">
    <HTML>
      <HEAD>
        <TITLE>XSA Record for
          <xsl:value-of select='vendor/name' />
        </TITLE>
      </HEAD>
      <BODY>
        <H1>XSA Record for <xsl:value-of select='vendor/name' /></H1>
```

```

        <xsl:text>Contact </xsl:text>
        <A HREF='{vendor/url}'><xsl:value-of select='vendor/name' /></A>
        at <A HREF='mailto:{vendor/email}'>
        <xsl:value-of select='vendor/email' /></A>
        <xsl:if test='$show-products = "yes"'>
            <xsl:apply-templates select='product' />
        </xsl:if>
    </BODY>
</HTML>
</xsl:template>

<xsl:template match='product'>
    <DIV>
        <P><BIG>Product:
        <A HREF='{info-url}'><xsl:value-of select='name' /></A> -
        <xsl:value-of select='version' /></BIG></P>
        <P>Released:
            <xsl:call-template name='format-date'>
                <xsl:with-param name='date' select='last-release' />
            </xsl:call-template>
        </P>
        <P>Changes in release:
            <PRE>
                <xsl:value-of select='changes' />
            </PRE>
        </P>
    </DIV>
</xsl:template>

<xsl:template name='format-date'>
    <xsl:value-of select='
        concat(substring(last-release, 1, 4), "-",
            substring(last-release, 5, 2), "-",
            substring(last-release, 7, 2))' />
</xsl:template>

</xsl:stylesheet>

```

Using the XSA style sheet

Now that you've created the XSA-specific style sheet from [A style sheet for displaying XSA](#) on page 13 , add it to the repository:

```
4ss create document --type=xslt xsa.xslt /web/xsa/xsa.xslt
```

The `--type` option is actually optional in this case because 4Suite does some inferencing for you. If 4Suite sees the proper `xsl:stylesheet` or `xsl:transform` document element, it automatically sets the type to XSLT. XSLT documents are a distinct document type in 4Suite because they are preprocessed for maximum performance when you use them as style sheets (remember that you can also use them as plain XML documents).

Now you can view the transformed document by browsing

`http://localhost:8080/xsa/4suite.xml?xslt=xsa.xslt`. The following figure shows what the output from the XSA style sheet looks like. Your results ought to resemble this screen shot:



If you try this yourself, remember to use the correct port if you have configured your scratch server differently. This time the full path to `xsa.xslt` is not needed because the style sheet is in the same container as the source file. This is very similar to file system locations. So, for instance, to access a style sheet in the parent container in which the source document is stored, you could use a URL such as `http://localhost:8080/a/b/c.xml?xslt=../d.xslt`, which would apply the style sheet `d.xslt` in the `/a` container against the `c.xml` source document in the `/a/b` container.

You might have noticed that the XSA style sheet supports a top-level parameter to determine whether or not to display the individual product information. You can provide values for such top-level parameters by adding URL query parameters. To apply the style sheet but show only the vendor information, use the following URL:

So how do you query across the repository?

Presenting XSA documents according to the approach described so far in this section isn't terribly special. You would do things quite similarly if the documents were hosted directly on the file system and you had a basic Web server. 4Suite's main contribution here is the integrated XSLT engine. But what if you have a variety of XSA documents in the system and want to display information according to a query of the whole repository, rather than knowledge of a particular XSA document? This is where the RDF database comes in, and where 4Suite repository features shine. After all, this database is automatically updated when XSA instance documents are added or modified. The flexibility of RDF makes for an extraordinarily rich query of the system. As we mentioned at the beginning of the tutorial, we use Versa for RDF query.

Setting up namespace mappings

Namespace mappings are very handy in Versa for brevity in queries. You start by adding a namespace mapping for the `xsa` prefix to the user preferences. The package for this tutorial comes with a pair of scripts for just this purpose. First, add a namespace mapping for your user, where you replace `/home/uo/ns-mappings.xml` with any similar path in your own home container:

```
python create-nsmapping-file.py /home/uo/ns-mappings.xml
```

Next add the mapping for `xsa`:

```
python add-nsmapping.py xsa http://www.garshol.priv.no/download/xsa/
```

The first parameter is the namespace prefix and the second is the namespace URI.

Querying XSA information

To run Versa queries from the 4Suite command line, you can use the `4ss rdf versa` command. (You can also use the Triclops Web form, but we'll stick to the command line in this tutorial, for simplicity.) The following query retrieves all resources of type `xsa:Vendor`:

```
4ss rdf versa "type(xsa:Vendor) "
```


You should get the following result:

```
<Set>
  <Resource>http://fourthought.com</Resource>
</Set>
```

The following query retrieves the name and distributed products of the resource with ID `http://fourthought.com`:

```
4ss rdf versa "distribute(@'http://fourthought.com', '.-xsa:name->*',
    '.-xsa:distributes->*')"
```

(The preceding code is split into two lines for ease of viewing. You actually enter the query as a single line.)

You should get the following result:

```
<List>
  <List>
    <List>
      <String>Fourthought, Inc.</String>
    </List>
    <List>
      </List>
    </List>
    <List>
      <Resource>http://4Suite.org</Resource>
    </List>
  </List>
</List>
```

Note: The `xsa:name` sub-query should return just one entry, but `xsa:distributes` could, in theory, return a list of items if the vendor distributes multiple products.

A version of the XSA style sheet that uses embedded RDF query

Now you can put all this together for a variation of the provided `xsa.xslt` style sheet (see [A style sheet for displaying XSA](#) on page 13) that presents similar output, using RDF query rather than direct XPath operations on the source document. This file is available in the tutorial package as **`xsa-rdf.xslt`** (see [Resources](#) on page 36).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- A -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:exslt="http://exslt.org/common"
```

```

        xmlns:f="http://xmlns.4suite.org/ext"
        xmlns:frdf="http://xmlns.4suite.org/4ss/rdf"
        xmlns:xsa="http://www.garshol.priv.no/download/xsa/"
        extension-element-prefixes="f frdf"
        version="1.0">

<!-- B -->
<xsl:param name='vendor' />

<!-- C -->
<xsl:variable name="vendor-info-rtf">
  <frdf:versa-query query="
    distribute(@'{$vendor}', '.-xsa:name->*', '.-xsa:distributes->*')
  "/>
</xsl:variable>
<xsl:variable name="vendor-info"
  select="exslt:node-set($vendor-info-rtf)"/>

<!-- D -->
<xsl:template match="/">
  <HTML>
    <HEAD>
<!-- E -->
      <TITLE>Products from XSA record for
        <xsl:value-of select='$vendor-info/List/List/List[1]'/>
      </TITLE>
    </HEAD>
    <BODY>
      <H1>Products from XSA record for
        <xsl:value-of select='$vendor-info/List/List/List[1]'/></H1>
      <xsl:apply-templates select='$vendor-info/List/List/List[2]/*'
        mode='product' />
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match='Resource' mode='product'>
<!-- F -->
  <xsl:variable name="product-info-rtf">
    <frdf:versa-query query="
      distribute(@'{'', '.-xsa:name->*', '.-xsa:version->*',
        '.-xsa:changes->*')
    "/>
  </xsl:variable>
  <xsl:variable name="product-info"
    select="exslt:node-set($product-info-rtf)"/>

  <DIV>
    <P><BIG>Product:
      <A HREF='{.'}'>
        <xsl:value-of select='$product-info/List/List/List[1]'/>
      </A> -
      <xsl:value-of select='$product-info/List/List/List[2]'/>
    </BIG></P>
    <P>Changes in release:
      <PRE>
        <xsl:value-of select='$product-info/List/List/List[3]'/>
      </PRE>
    </P>
  </DIV>
</xsl:template>

```

```
</DIV>
</xsl:template>

</xsl:stylesheet>
```

Dissecting the style sheet

This discussion follows the sections of the style sheet (in [A version of the XSA style sheet that uses embedded RDF query](#) on page 17) that are marked by lettered comments.

A: The namespace mapped to `f` is a special one for 4Suite XSLT extension functions and elements that are usually handy to have. The namespace mapped to `frdf` is for special XSLT extensions for accessing the repository RDF database. The namespace mapped to `exslt` defines a popular library of extension functions and elements that are supported by a variety of processors, including 4Suite's. (See [Resources](#) on page 36 for more information on EXSLT.) You need to define the `xsa` prefix for use in embedded Versa query. Some of these extension namespaces define special elements; you set the standard `extension-element-prefixes` attribute accordingly.

B: The URL of the vendor is passed in as a parameter, `vendor`. Notice that we did not bother with the `show-products` parameter, which we used earlier only for illustration.

C: This part shows how to embed Versa queries in XSLT. The `frdf:versa-query` extension element performs the given query and returns the result as an XML result identical to what you'd see if you were working with Versa on the command line. The `query` attribute is an attribute value template, with which you pass in the `vendor` parameter given as a Versa resource literal (`@'{$vendor}'`). The result then becomes the value of the variable `$vendor-info-rtf` as a result-tree fragment. The `distribute` Versa function returns results as embedded `List` elements. The code needs to use XPath to examine this XML Versa result to extract the useful data from the `List` elements. To do this according to XML 1.0 rules means using the EXSLT standard extension function `exslt:node-set` to convert the result-tree fragment to a node set.

D: This entire style sheet operates on Versa queries so the source document no longer matters. Instead of matching on an element from the source document, you start out matching on `/`, which is always there.

E: Here you dissect the result from the `distribute` result to get the information you need: the name of the vendor from the first sub-query in the `distribute`. (You might want to compare the output of the same `distribute` you tried on the command line.)

F: Continue to use a similar query on the product information for each product.

You'll observe that this style sheet omitted the vendor e-mail and product "last released" information from the display. That is because we left them out of the RDF mapping for brevity. These properties are not central to the discussion, so you can ignore them for now.

Trying out the modified XSA style sheet

To try out the new style sheet that incorporates the query, start by adding the new XSA style sheet to the repository:

```
4ss create document --type=xslt xsa-rdf.xslt /web/xsa/xsa-rdf.xslt
```

Next, open

`http://localhost:8080/xsa/?xslt=xsa-rdf.xslt&vendor=http://fourthought.com` in a browser to apply the transform. The source document for the transform is a container object, no longer the XSA document (`4suite.xml`) itself. Remember that the source document no longer matters; the premise of this new style sheet is that the data comes from metadata across the repository rather than the actual XSA source document. Also, we now set the `vendor` parameter, as required by the style sheet, in the URL parameter `vendor=http://fourthought.com`.

By the way, if you are experimenting with this and similar XSLT scripts and have trouble getting the results you expect, you may want to consult our collection of debugging tips (see [Resources](#) on page 36).

Section 4. Indices and updates

Indexing and RDF

This section shows how:

- To make an index of software vendor records: a simple Web page with one line for each entry, giving the title and a link to the detailed vendor page. This task demonstrates RDF query across the repository.
- To develop a simple form for updating software records. This task demonstrates how to update XML documents and add RDF data in the repository.

An index of records

Here is a style sheet that renders an index listing of the vendor records:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:exslt="http://exslt.org/common"
  xmlns:f="http://xmlns.4suite.org/ext"
  xmlns:frdf="http://xmlns.4suite.org/4ss/rdf"
  xmlns:xsa="http://www.garshol.priv.no/download/xsa/"
  extension-element-prefixes="f frdf"
  version="1.0">

  <!-- A -->
  <xsl:variable name="vendors-info-rtf">
    <frdf:versa-query query="
      distribute(type(xsa:Vendor), '.', '-xsa:name->*',
        '(-xsa:distributes->*)-xsa:name->*')
    " />
  </xsl:variable>
  <xsl:variable name="vendors-info"
    select="exslt:node-set($vendors-info-rtf)" />

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>Vendors with XSA records</TITLE>
      </HEAD>
      <BODY>
        <H1>Vendors with XSA records</H1>
        <TABLE>
          <!-- B -->
          <xsl:apply-templates select='$vendors-info/List/List' />
        </TABLE>
      </BODY>
    </HTML>
  </template>
</xsl:stylesheet>
```

```

mode='vendor' />
    </TABLE>
</BODY>
</HTML>
</xsl:template>

<xsl:template match='List' mode='vendor'>
    <!-- C -->
    <TR>
        <TD>
            <A HREF='./?xslt=xsa-rdf.xslt&vendor={*[1]}'>
                <xsl:value-of select='*[2]'/>
            </A>
        </TD>
        <TD>
            <!-- D -->
            <xsl:for-each select='*[3]/*'>
                <xsl:value-of select='.'/>
                <xsl:if test='not(position() = last())'>
                    <xsl:text>, </xsl:text>
                </xsl:if>
            </xsl:for-each>
        </TD>
    </TR>
</xsl:template>

</xsl:stylesheet>

```

Understanding the index.xslt style sheet

This explanation of the style sheet in [An index of records](#) on page 21 follows the lettered parts marked with comments.

A: This style sheet uses a different Versa query, which basically says, "Get me all the resources of type xsa:Vendor; then for each resource, get me its URI, name, and the names of all the products it distributes." The

(.-xsa:distributes->*)-xsa:name->* is an example of chained traversal in Versa, and it results in a list of product names.

B: This applies a template to the section of \$vendors-info that represents each vendor, in turn.

C: This template handles the display for each vendor. By examining the distribute statement in the query, you can tell that the first context element is the URI, the second is the name, and the third is a list of the products.

D: This part iterates over the list of product names for the vendor and prints them out, with each one separated by a comma.

Trying out the index

First, load the index style sheet from the command line:

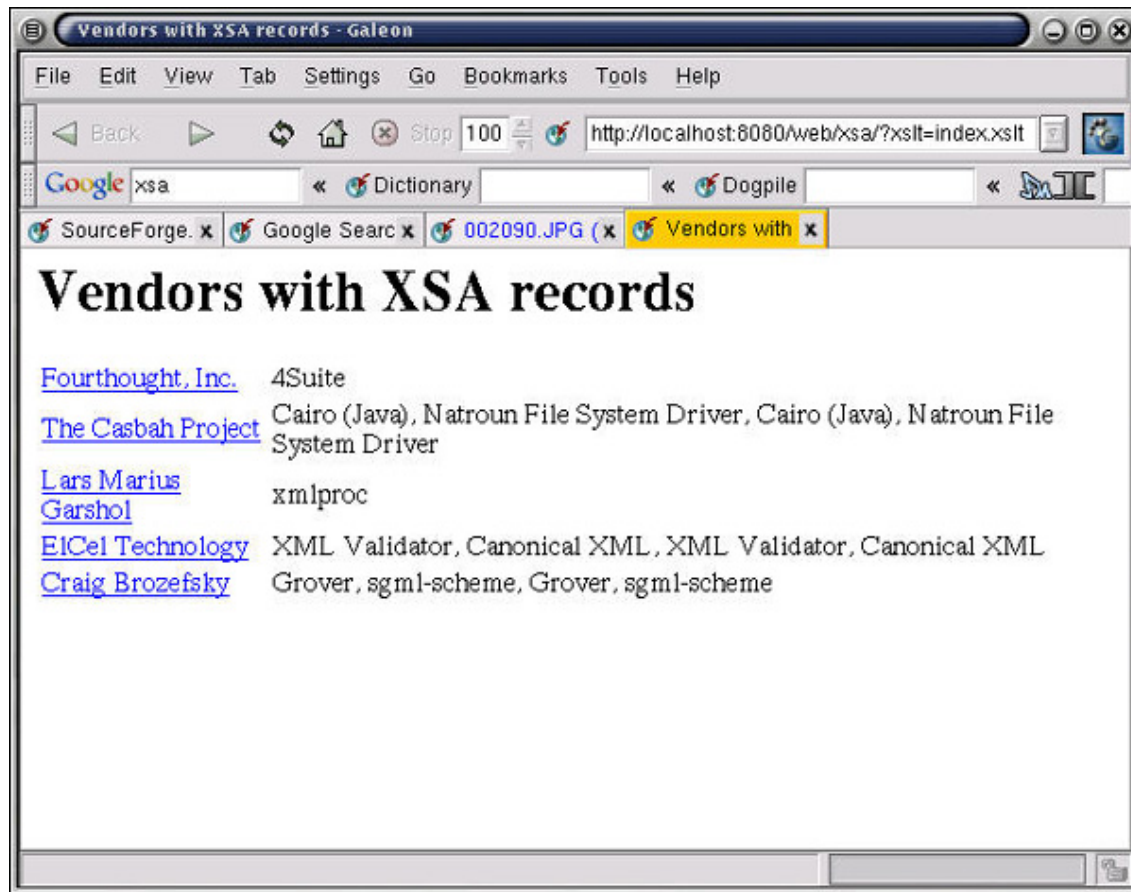
```
4ss create document --type=xslt index.xslt /web/xsa/index.xslt
```

So far the index has only one entry, which is not very interesting. In the tutorial package (in [Resources](#) on page 36), you will find a Python script that adds a number of XSA records. Execute it as follows:

```
python add_xsa_batch.py
```

You can then view the record index by visiting
<http://localhost:8080/xsa/?xslt=index.xslt>.

This screen shot shows output from the index script, which should resemble the index you see in your browser window.



A form for adding new vendor records

We shall create a form for adding vendor records. For the sake of simplicity, this form allows the addition of only one product per vendor record. The front end of this form is plain HTML, as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Add a new vendor record</title>
  </head>

  <body>
    <h1>Add a new vendor record</h1>

    <form method="post" action=" ../?xslt=add-vendor-handler.xslt">
      <table>
        <tr>
          <td>Vendor name</td><td><input type="text" name="vname"/></td>
```



```
        </tr>
      <tr>
        <td>Vendor URL</td><td><input type="text" name="vurl"/></td>
      </tr>
      <tr>
        <td>Vendor e-mail</td><td><input type="text" name="vemail"/></td>
      </tr>
      <tr>
        <td>Product name</td><td><input type="text" name="pname"/></td>
      </tr>
      <tr>
        <td>Current version</td><td><input type="text" name="pversion"/></td>
      </tr>
      <tr>
        <td>Product URL</td><td><input type="text" name="purl"/></td>
      </tr>
      <tr>
        <td>Product last release</td><td><input type="text" name="prelease"/></td>
      </tr>
      <tr>
        <td>Changes in current version</td>
        <td><textarea rows="4" cols="40" name="pchanges"></textarea></td>
      </tr>
      <tr>
        <td colspan="2"><input type="submit" name="submit"/></td>
      </tr>
    </table>
  </form>

</body>
</html>
```

Using XSLT for server-side handling

The code for the form in [A form for adding new vendor records](#) on page 24 is plain old HTML. Take a look at the form action URL, `./?add-vendor-handler.xslt` (boldfaced in the listing), which applies an XSLT script against the current path in the repository. This XSLT script actually does the server-side handling of the client request. Because 4Suite allows XSLT scripts to query and update the repository, you can use them as a server-side scripting implementation.

The most important items that a server-side script needs to get from a Web form are the form variables. These include the values the user enters into each of the controls in the form. 4Suite communicates this to server-side XSLT by passing them as XSLT parameters. The name of each parameter is the name given to the HTML form element (say `vname`). The value is what the user specifies (say `PythonLabs`). This transmission goes on behind the scenes, with no special work needed by the developer except to declare the parameters in the XSLT handler script and to use them as needed in the operation.

A handler for the form adding new vendor records

This code shows the handler style sheet, available in the tutorial packet as **add-vendor-handler.xslt** (see [Resources](#) on page 36).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- A -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:exslt="http://exslt.org/common"
  xmlns:f="http://xmlns.4suite.org/ext"
  xmlns:frdf="http://xmlns.4suite.org/4ss/rdf"
  xmlns:fcore="http://xmlns.4suite.org/4ss/score"
  xmlns:xsa="http://www.garshol.priv.no/download/xsa/"
  extension-element-prefixes="f frdf fcore"
  version="1.0">

  <!-- B -->
  <xsl:param name="vname"/>
  <xsl:param name="vurl"/>
  <xsl:param name="vemail"/>
  <xsl:param name="pname"/>
  <xsl:param name="pversion"/>
  <xsl:param name="purl"/>
  <xsl:param name="prelease"/>
  <xsl:param name="pchanges"/>

  <xsl:template match="/">
    <xsl:call-template name='add-record'/>
    <HTML>
      <HEAD>
        <TITLE>New record added</TITLE>
      </HEAD>
      <BODY>
        <H1>New record added</H1>
        <P>To view the record, click
          <A HREF="./?xslt=xsa-rdf.xslt&vendor={$vurl}">here</A>
        </P>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template name='add-record'>
    <!-- C -->
    <fcore:create-document
      path="/web/xsa/{f:generate-uuid()}" docdef="/web/xsa/xsa.dd"
      encoding="UTF-8"
      doctype-system="http://www.garshol.priv.no/download/xsa/xsa.dtd"
      doctype-public=
        "-//LM Garshol//DTD XML Software Autoupdate 1.0//EN//XML"
    >
    <xsa>
      <vendor>
```

```
<name><xsl:value-of select="$vname"/></name>
<email><xsl:value-of select="$vemail"/></email>
<url><xsl:value-of select="$vurl"/></url>
</vendor>
<product id="{ $pname}">
  <name><xsl:value-of select="$pname"/></name>
  <version><xsl:value-of select="$pversion"/></version>
  <last-release><xsl:value-of select="$prelease"/></last-release>
  <info-url><xsl:value-of select="$purl"/></info-url>
  <changes><xsl:value-of select="$pchanges"/></changes>
</product>
</xsa>
  </fcore:create-document>
</xsl:template>

</xsl:stylesheet>
```

Trying out the addition form

To try out the form, add the HTML form and XSLT handler to the repository. You actually enter each 4ss create command on a single line:

```
4ss create rawfile --imt=text/html add-vendor-form.html
    /web/xsa/add-vendor-form.html
4ss create document --type=xslt add-vendor-handler.xslt
    /web/xsa/add-vendor-handler.xslt
```

(In the preceding code, lines of code were split into two lines for ease of viewing.)

The first command adds a simple file to the repository. It is not specially treated, as XML files are. The `--imt` option tells the repository what media type to associate with the file. For instance, if the repository serves up this file on the Web, then that is the media type it uses.

This handler involves adding a document to the XSA container. By default, 4Suite won't let just any Web user execute such an operation. You have to explicitly allow this by setting the access control list (ACL) options for the container. Enter:

```
4ss set acl /web/xsa/
```

You'll see a listing of the currently allowed users and operations, and then a prompt. At this prompt, enter **s**. It will ask you for the type of access. Enter **write**. Finally it will ask you for the identity of the user or group to allow. Enter **world**. It will then ask you for confirmation. Enter **yes**.

After running these commands, you can access the addition form by simply accessing

the HTML file directly in the repository:

`http://localhost:8080/web/xsa/add-vendor-form.html`. When you use the form, once you click the **Add** button, the form you've entered is used to construct a new record, which you can view from the index page.

Section 5. Wrap-up and packaging

Section outline

As a wrap-up of this tutorial, we'll present one neat feature for the sample application we've been developing. We'll also show you how to use 4Suite to package all the various parts you've been using so far. Specifically, this section:

- Implements a dynamic RSS feed for the software records
- Implements a packaging file that allows all the files and resources to be installed into the repository in one simple command

Introducing RDF Site Summary (RSS)

RDF Site Summary (RSS) is a very lightweight content-syndication format. Web developers use RSS to distribute blurbs about content broadly and easily. RSS is a popular RDF application that's used by Slashdot.org, Freshmeat.net, XML.com, the Motley Fool, O'Reilly.net (Meerkat), Wired News, Linux Today, Gnotices, and many other Web sites. Here's a sample RSS file:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
>
  <channel rdf:about="http://www.xml.com/xml/news.rss">
    <title>XML.com</title>
    <link>http://xml.com/pub</link>
    <description>
      XML.com features a rich mix of information and services
      for the XML community.
    </description>
    <items>
      <rdf:Seq>
        <rdf:li resource="http://www.xml.com/pub/a/2000/10/11/rdf/" />
      </rdf:Seq>
    </items>
  </channel>

  <item rdf:about="http://www.xml.com/pub/a/2000/10/11/rdf/">
    <title>4RDF: A Library for Web Metadata</title>
    <link>http://www.xml.com/pub/a/2000/10/11/rdf/</link>
    <description>
      One of the jewels in the crown of Python's XML support is the
      4Suite collection of libraries, the most recent addition to
```

```
        which is 4RDF, a library for the parsing, querying, and
        storage of RDF.
    </description>
</item>

</rdf:RDF>
```

A brief walk-through of RSS

The `channel` element describes a body of content, usually one source of notification on all the content coming from one site. A channel has a `title`, a main `link`, a `description`, and one or more `items`. It can also have other useful data such as links to images that portals can use.

Each `item` represents one particular piece of content and has a particular `link`, along with a `title` and `description`.

The link elements are duplicated in `rdf:about` attributes, which are provided because RSS is an RDF format.

Generating RSS in 4Suite

This script provides a dynamic RSS feed of the records of software that has been released since a given date:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- A -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:exslt="http://exslt.org/common"
    xmlns:f="http://xmlns.4suite.org/ext"
    xmlns:frdf="http://xmlns.4suite.org/4ss/rdf"
    xmlns:xsa="http://www.garshol.priv.no/download/xsa/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://purl.org/rss/1.0/"
    extension-element-prefixes="f frdf"
    version="1.0">

<!-- B -->
<xsl:variable name="product-info-rtf">
    <frdf:versa-query query="distributed(type(xsa:Product), '.',
        '._xsa:release->*)'"/>
</xsl:variable>
<xsl:variable name="product-info" select="exslt:node-set($product-info-rtf)"/>
```

```

<!-- C -->
<xsl:template match="/">
<rdf:RDF>
  <channel rdf:about="http://uche.ogbuji.org/eg/xsa/">
    <title>XML.com</title>
    <link>http://uche.ogbuji.org/eg/xsa/</link>
    <description>A collection of XSA records</description>
    <items>
      <rdf:Seq>
        <xsl:apply-templates select="$product-info/List/List" mode="item-list"/>
      </rdf:Seq>
    </items>
  </channel>

  <xsl:apply-templates select="$product-info/List/List" mode="item-detail"/>
</rdf:RDF>

</xsl:template>

<!-- D -->
<xsl:template match="List" mode="item-list">
  <rdf:li resource="{*[1]}" />
</xsl:template>

<xsl:template match="List" mode="item-detail">
  <xsl:variable name="product-info-rtf">
    <frdf:versa-query query="distribute(@'{*[1]}', '.-xsa:name->*',
      '.-xsa:version->*', '(&lt;-xsa:distributes-*)-xsa:name->*')"/>
  </xsl:variable>
  <xsl:variable name="product-info" select="exslt:node-set($product-info-rtf)"/>
  <item rdf:about="{*[1]}">
    <title><xsl:value-of select='$product-info/List/List/List[1]'/></title>
    <link><xsl:value-of select='*[1]'/></link>
    <description>
      <xsl:value-of select='$product-info/List/List/List[1]'/>version
      <xsl:value-of select='$product-info/List/List/List[2]'/>
      was released by <xsl:value-of select='$product-info/List/List/List[3]'/>
    </description>
  </item>
</xsl:template>

</xsl:stylesheet>

```

Dissecting the RSS script

The detailed look at the RSS script in [Generating RSS in 4Suite](#) on page 30 follows the parts marked in the code comments.

A: This Versa query grabs all products and their release dates.

B: This is the main template, which outlines the document and then processes the Versa query results twice: once in a mode for displaying the entries in the item list, and

again to display the full item records.

C: This template simply renders the product URL.

D: This template initiates another Versa query, which retrieves the product name, version, and the vendor name. The RSS item is then rendered from the results of this query.

Trying it out

To try out the RSS script, add the script as usual:

```
4ss create document --type=xslt rss.xslt /web/xsa/rss.xslt
```

You can check the RSS document that's created by this XSLT script by browsing <http://localhost:8080/xsa/?xslt=rss.xslt>. Remember that RSS is XML rather than HTML, so be sure to use a browser that can handle XML effectively.

The setup file for the XSA application

4Suite supports setup files, which use an XML vocabulary for packaging applications. In particular, setup files define a set of resources in 4Suite to be set up. They are in some ways similar to makefiles. You can write a single setup file to load all the files we have presented for the XSA application. The 4Suite tools for handling this are smart enough that they can leave up-to-date resources alone, adding only resources that are not yet present in the repository, and updating only those that have changed.

```
<!-- A -->
<ftssp:Product
  xmlns:ftssp="http://xmlns.4suite.org/reserved#product"
  xmlns:dc="http://purl.org/dc/elements/1.1#"
  name="XSA application" version="1.0"
>
  <dc:Title>XSA application</dc:Title>
  <dc:Description>XSA application</dc:Description>

  <!-- B -->
  <ftssp:Container path="/web/xsa">
    <ftssp:Acl>
      <ftssp:WorldAccess type='write' allowed='1' />
    </ftssp:Acl>
  </ftssp:Container>
```



```
<!-- C -->
<ftssp:XPathDocumentDefinition path='/web/xsa/xsa.dd' src='xsa.dd' />

<!-- D -->
<ftssp:XmlDocument path='/web/xsa/4suite.xml'
  src='4suite.xml' doc-def='/web/xsa/xsa.dd' />

<!-- E -->
<ftssp:XsltDocument path="/web/xsa/xsa.xslt" src="xsa.xslt" />
<ftssp:XsltDocument path="/web/xsa/xsa-rdf.xslt" src="xsa-rdf.xslt" />
<ftssp:XsltDocument path="/web/xsa/index.xslt" src="index.xslt" />
<ftssp:XsltDocument path="/web/xsa/rss.xslt" src="rss.xslt" />

<!-- F -->
<ftssp:RawFile path="/web/xsa/add-vendor-form.html"
  src="add-vendor-form.html" imt="text/html" />
<ftssp:XsltDocument path="/web/xsa/add-vendor-handler.xslt"
  src="add-vendor-handler.xslt" />

</ftssp:Product>
```

Understanding the setup file

As before, this discussion of the code follows the parts marked by comments in the code shown in [The setup file for the XSA application](#) on page 32 .

A: The top-level wrapper element has information such as the name of the overall package it represents, and the version of that package.

B: The `ftssp:Container` element specifies a container to be created. The `path` attribute specifies the path in the repository of the created container. In order for the Web user to be able to add XSA records, this container must be writable by any party, so we explicitly set the access control using an embedded `ftssp:Acl` element, setting access for "world" to write.

C: The `ftssp:XPathDocumentDefinition` element specifies an XPath document definition to be created or updated in the repository. The `src` attribute specifies the source from which the content of the document is to be read.

D: The `ftssp:XmlDocument` element specifies a regular XML document to be created. The `doc-def` attribute gives the repository path of the document definition to be used.

E: The `ftssp:XsltDocument` element specifies an XSLT transform to be created or updated in the repository.

F: The `ftssp:RawFile` element specifies a plain file to be stored or updated in the repository. The `imt` attribute gives the media type of the added resource.

Using the installation package

To use the installation package, you can simply change to the directory with the setup file and the other source files, and run the `4ss install` command, which creates or updates all specified resources based on whether their local sources have changed.

```
4ss install xsa-setup.xml
```

Wrapping up

This tutorial has explained the basics of how to develop Web applications using the 4Suite repository. It has also outlined two general XML technologies: XSA and RSS.

If you have followed along, you have learned how to:

- Add files to the 4Suite repository
- Set up the RDF database to be updated with metadata from those files
- View the resulting files in a Web browser
- Create an optional XSLT transform that can be applied dynamically
- Write server-side handling scripts in XSLT
- Package 4Suite applications conveniently

You can still explore a lot of detail in 4Suite. The demos bundled with 4Suite are a good place to look, if you like sample code. Their source code resides in the `Ft/Server/Share/Demos` directory of the 4Suite distribution, and you can view them online in your local 4Suite installation by browsing `http://localhost:8801` after starting the server.

4Suite also comes with documentation of the many extension elements available; for more prosaic examples, hints, and tips, see [Resources](#) on page 36 .

Where to go from here

This tutorial concludes our series on 4Suite. After completing all these tutorials, you should be able to put 4Suite to very effective use right away. Because software always changes, and there is always more to say, we have set up [a Web site](#)

(<http://uche.ogbuji.net/tech/4Suite/ibmdw-tutorial/>) for updates, additions, and expansions on these tutorials.

Keep an eye on this site, and let us know (see [Getting help and finding out more](#) on page 4 for contact information) if there is anything you as a tutorial user would find useful on that site.

Section 6. Resources

Resources

Other XML and 4Suite tutorials

- If you're completely new to XML, start with Doug Tidwell's "[Introduction to XML](http://www.ibm.com/developerworks/education/r-xmlintro.html)" tutorial (<http://www.ibm.com/developerworks/education/r-xmlintro.html>) (August 2002).
- The first tutorial in this series, "[Python and XML development using 4Suite, Part 1: PyXml](http://www.ibm.com/developerworks/education/r-x4suite.html)" (<http://www.ibm.com/developerworks/education/r-x4suite.html>), details the all-important DOM manipulation techniques (October 2001).
- [Part 2 \(XPath and XSLT\)](http://www.ibm.com/developerworks/education/r-x4suite2.html) (<http://www.ibm.com/developerworks/education/r-x4suite2.html>), gives a thorough introduction to key XML technologies XPath and XSLT for identifying nodes in an XML document's tree and for transforming documents, and it shows how to use them in the 4Suite tool set (October 2001).
- [Part 3 \(4RDF\)](http://www.ibm.com/developerworks/education/r-x4suite3.html) (<http://www.ibm.com/developerworks/education/r-x4suite3.html>) introduces RDF and details how to work with it using the 4Suite tool set (July 2002).
- [Part 4 \(Composition and updates\)](http://www.ibm.com/developerworks/education/r-x4suite4.html) (<http://www.ibm.com/developerworks/education/r-x4suite4.html>) introduces the W3C XML specifications XPointer, XInclude, and XML Base and the independent specification XUpdate, which offers an alternative to XSLT and DOM parsing for updating parts of XML documents (October 2002).

Other resources

- All of the code listings in this tutorial are included in this [xsa-app.zip](#) file.
- Download [4Suite](http://4Suite.org) 0.12.0 (<http://4Suite.org>) to follow along with this tutorial. 4Suite requires [Python](#) version 2.1 or newer (version 2.2 or newer is highly recommended). There are also many resources for 4Suite on [Uche Ogbuji's 4Suite page](http://uche.ogbuji.net/tech/4Suite/) (<http://uche.ogbuji.net/tech/4Suite/>), which also hosts [The 4Suite configuration helper](http://uche.ogbuji.net/tech/4Suite/?xslt=config-helper.xslt) (<http://uche.ogbuji.net/tech/4Suite/?xslt=config-helper.xslt>), an interactive form for creating 4Suite configuration files.
- Read more about [XSA](#) on Lars Marius Garshol's home page (<http://www.garshol.priv.no/download/xsa/>).
- Learn about the Versa query language for RDF in the column "[RDF Query using Versa](http://www.ibm.com/developerworks/library/x-think10/index.html)" (<http://www.ibm.com/developerworks/library/x-think10/index.html>), right here on *developerWorks* (April 2002). After that, review the [Versa tutorial](http://uche.ogbuji.net/tech/rdf/versa/versa-by-example.txt) (<http://uche.ogbuji.net/tech/rdf/versa/versa-by-example.txt>). Lessons 1 through 10 are sufficient for this tutorial, though you might want to read all the lessons for maximum comfort with Versa.
- Look to [EXSLT](http://exslt.org) (<http://exslt.org>) for useful and widely supported extension functions

for XSLT. The [common](#) module has the node-set and other functions (<http://exslt.org/exsl/index.html>).

- Uche Ogbuji hosts a great deal of useful tips and examples for 4Suite development in his [4Suite Akara site](#) (<http://uche.ogbuji.net/tech/akara/4suite/>), which includes [this page of XSLT debugging tips](#) for 4Suite repository users (<http://uche.ogbuji.net/tech/akara/4suite/debug-xslt/>).
- There are several resources on RSS in IBM *developerWorks*. "[An introduction to RSS news feeds](#)" (<http://www.ibm.com/developerworks/library/w-rss.html>), by James Lewin, is older, but a good place to start. It covers RSS 0.91 and 1.0, and Perl interfaces (November 2000). "[Grab headlines from a remote RDF file](#)" (<http://www.ibm.com/developerworks/xml/library/x-tiphtml.html>), by Nicholas Chase, shows some XSLT and JSP code for processing RSS 0.91 and 1.0 (April 2002).
- XML.com also has several articles on RSS. Read "[RSS: Lightweight Web Syndication](#)" (<http://www.xml.com/pub/a/2000/07/17/syndication/rss.html>), by Rael Dornfest, for a good general introduction. In "[Building a Semantic Web Site](#)" (<http://www.xml.com/pub/a/2001/05/02/semanticwebsite.html>), Eric van der Vlist provides an great technical introduction based on very practical examples. "[RSS Modularization](#)" (<http://www.xml.com/pub/a/2000/07/05/deviant/rss.html>), by Leigh Dodds, follows some very interesting conversation at a crucial juncture in RSS development.
- Find more XML resources on the [developerWorks XML zone](#) (<http://www.ibm.com/developerworks/xml/>).
- [IBM WebSphere Studio](#) (<http://www.ibm.com/software/info1/websphere/index.jsp?tab=landings/studiosplashv5>) provides a suite of tools that automate XML development, both in Java and in other languages. It is closely integrated with the [WebSphere Application Server](#) (<http://www.ibm.com/software/webervers/appserv/>), but can also be used with other J2EE servers.

Your feedback

Please send us your feedback on this tutorial. We look forward to hearing from you! Additionally, you are welcome to contact the authors, Uche Ogbuji (uche.ogbuji@fourthought.com) and Chimezie Ogbuji (chimezie@ogbuji.net).

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats

from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at

www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial [Building tutorials with the Toot-O-Matic](#) demonstrates how to use the Toot-O-Matic to create your own tutorials.

developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at

www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11.

We'd love to know what you think about the tool.