Working With XML and DTDs

tutorial #tutorial06

James L. Parry
B.C. Institute of Technology

Tutorial Goals

This tutorial is meant to give you some practice constraining and building an XML document, as described in lessons 7 and 8.

I have prepared a starter project for you to build on. It is not a webapp, but provides a standard folder to put stuff in. Next week, we can make it into a webapp, when you have to process the XML data you build this week.

Revisions & Notes

-

Background

Barker Bob, with his Burger Bar, wants to hold a bigger, better, burger bonanza, in honour of his uncle Billy Bob.

He has persuaded a local food company, PlentyOfFries, to sponsor a hackathon which will result in up to a dozen different web or mobile apps for people to order burgers for this event.

Barker Bob's ordering app needs to collect order data from all these disparate sources, and Grandpa George has decided that XML data is the way to achieve consistency and completeness. Our job is to derive a DTD to constrain orders, and several XML documents to help prove the correctness of our DTD.

Preparation

I have prepared a starter project. The lab 6 starter will just hold data for now, and functionality will be added next week.

For your reference, the menu we will be using is found in `/assets/images/menu1.gif`.

Fork the github project, and clone it locally to work with, the same as you have done with the previous tutorials.

The End Result

Your data folder will hold the DTD for Barker Bob, `bonanza.dtd`, and three XML orders:

- `order1.xml` will hold the XML data for a specialty burger #8
- `order2.xml` will hold the XML data for an order with three cheesburgers
- `order3.xml` will hold the XML data for several convoluted burgers.

What Needs Doing?

1. PLAN YOUR ORDER DATA STRUCTURE

Each order will have some common characteristics:

- An order is for one or more burgers
- Each burger may be customized differently
- An order can be for eat-in or for takeout
- Each order will have a designated customer name
- An order might have special instructions, eg for delivery

Looking at the menu, each burger has the following characteristics:

- A burger has a specific patty
- A burger may have one or two cheeses added
- A burger may have any number of toppings
- A burger may have any number of sauces
- A customer might have special instructions for a burger

What About Payment Details?

So far, all we are planning is the data needed to place an order.

Next week, we will enhance the data structure to provide for a unique order #, date/time stamp, total amount, and so on.

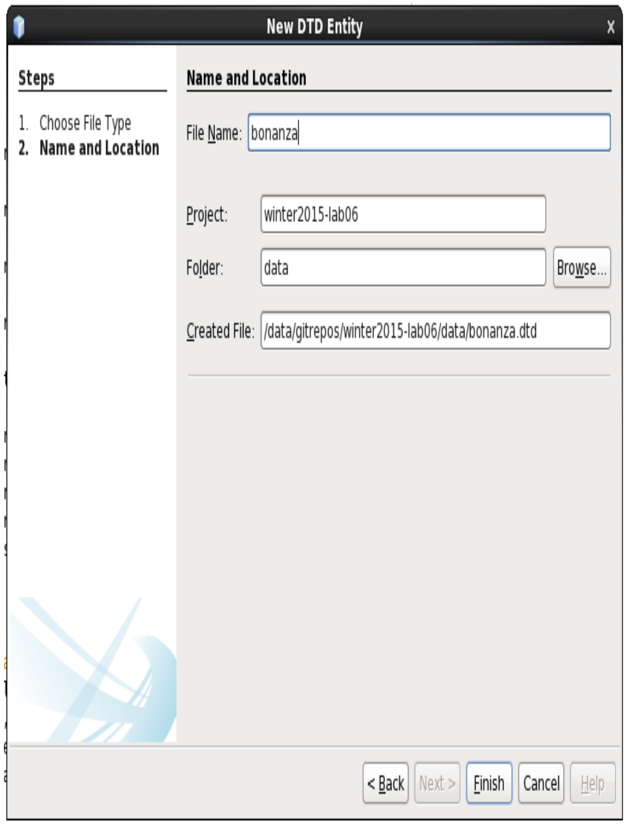
Pricing data, even though on the menu, can be added then too.

How Do We Start?

We need to add two fields to our quotes table:

Right-click on the data folder in your project tree, and choose new>DTD Entity...

Let's use the name "bonanza" for it.



File Name: bonanza

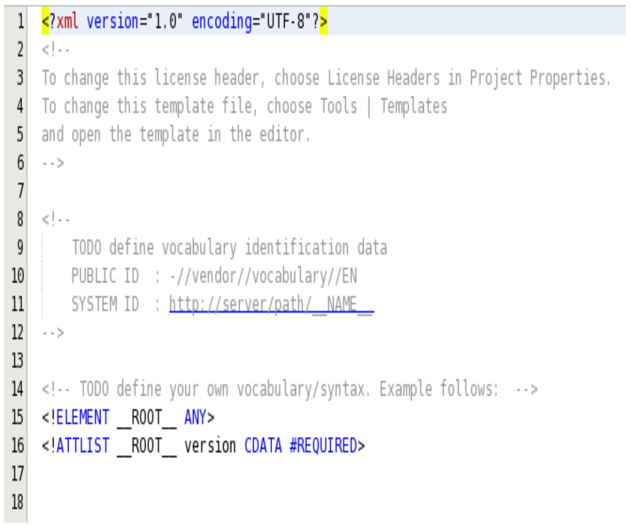
Project: winter2015-lab06

Folder: data

Created File: /data/gitrepos/winter2015-lab06/data/bonanza.dtd

Our Starter DTD

The starter DTD isn't too impressive... we will have to replace a bunch of pieces, sigh.

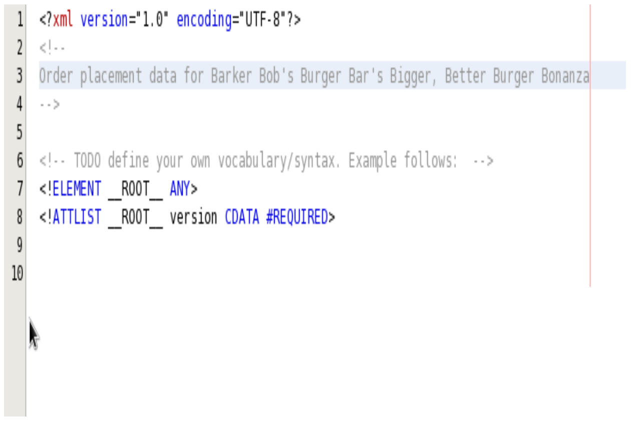


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->

<!--
    TODO define vocabulary identification data
    PUBLIC ID  : -//vendor//vocabulary//EN
    SYSTEM ID  : http://server/path/__NAME__
-->

<!-- TODO define your own vocabulary/syntax. Example follows:  -->
<!ELEMENT __ROOT__ ANY>
<!ATTLIST __ROOT__ version CDATA #REQUIRED>
```

2. BUILD A BASIC DTD WITH METADATA

We will make a number of changes to this, starting with the opening comment block

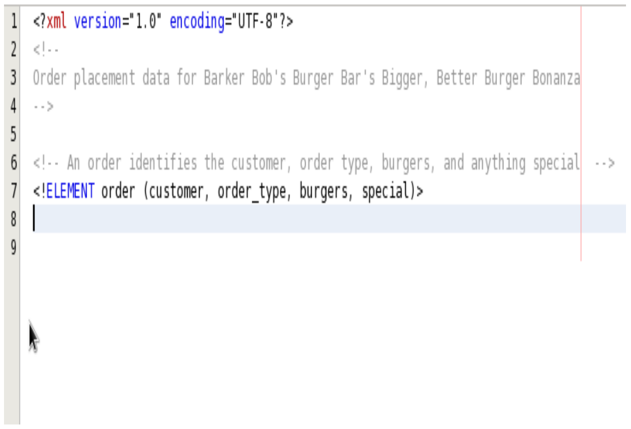
Yours does not have to be identical - feel free to add any notes that will help you make sense of it later.



```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order placement data for Barker Bob's Burger Bar's Bigger, Better Burger Bonanza
-->

<!-- TODO define your own vocabulary/syntax. Example follows:  -->
<!ELEMENT __ROOT__ ANY>
<!ATTLIST __ROOT__ version CDATA #REQUIRED>
```

Order Structure

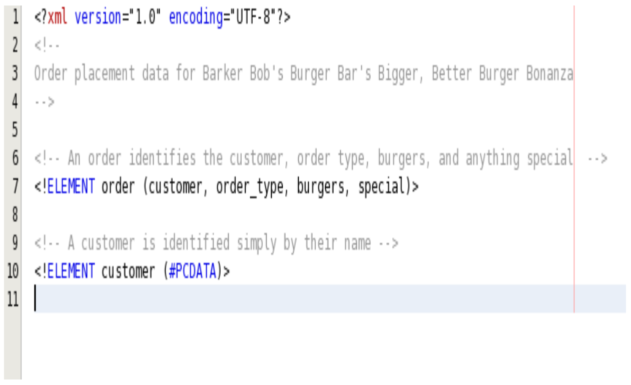
Here is the basic "order" plan, as it might make sense at this point.



```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order placement data for Barker Bob's Burger Bar's Bigger, Better Burger Bonanza
-->

<!-- An order identifies the customer, order type, burgers, and anything special  -->
<!ELEMENT order (customer, order_type, burgers, special)>
```

Customer Info

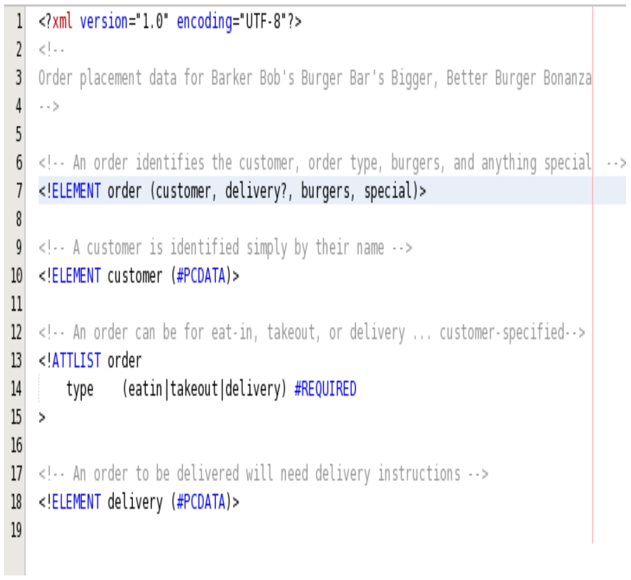
We need to flesh out the pieces that make up an order, starting with the customer,



```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order placement data for Barker Bob's Burger Bar's Bigger, Better Burger Bonanza
-->

<!-- An order identifies the customer, order type, burgers, and anything special -->
<!ELEMENT order (customer, order_type, burgers, special)>

<!-- A customer is identified simply by their name -->
<!ELEMENT customer (#PCDATA)>
```

Order Type

Order type makes more sense as an attribute, so we can add that and remove the "order_type" element we had planned. I have added an optional "delivery" element, planning ahead.



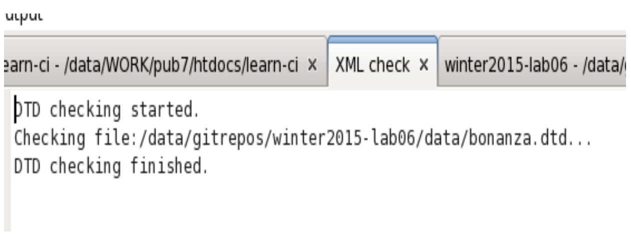
```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order placement data for Barker Bob's Burger Bar's Bigger, Better Burger Bonanza
-->

<!-- An order identifies the customer, order type, burgers, and anything special -->
<!ELEMENT order (customer, delivery?, burgers, special)>

<!-- A customer is identified simply by their name -->
<!ELEMENT customer (#PCDATA)>

<!-- An order can be for eat-in, takeout, or delivery ... customer-specified-->
<!ATTLIST order
    type    (eatin|takeout|delivery) #REQUIRED
>

<!-- An order to be delivered will need delivery instructions -->
<!ELEMENT delivery (#PCDATA)>
```

DTD Checking

Check your DTD at any time by right-clicking inside its panel and choosing "Check DTD".

The checking will stop at the first (obvious) error.

A "successful" check will show no complaints.

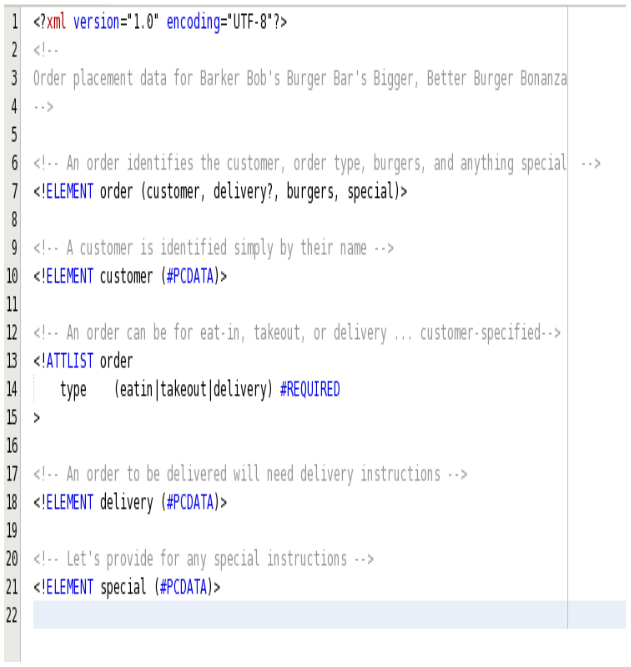

earn-ci - /data/WORK/pub7/htdocs/learn-ci ×
XML check ×
winter2015-lab06 - /data/

```
DTD checking started.
Checking file:/data/gitrepos/winter2015-lab06/data/bonanza.dtd...
DTD checking finished.
```

Order Instructions

This shows all of the metadata in place.

The "burgers" element is not there yet, but the DTD curiously "checks" without an error message :-/

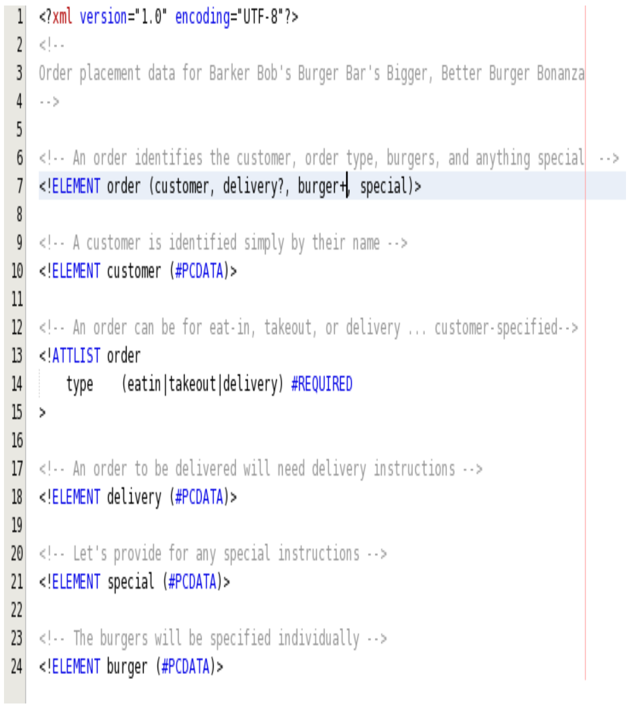


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order placement data for Barker Bob's Burger Bar's Bigger, Better Burger Bonanza
-->

<!-- An order identifies the customer, order type, burgers, and anything special  -->
<!ELEMENT order (customer, delivery?, burgers, special)>

<!-- A customer is identified simply by their name -->
<!ELEMENT customer (#PCDATA)>

<!-- An order can be for eat-in, takeout, or delivery ... customer-specified-->
<!ATTLIST order
    type    (eatin|takeout|delivery) #REQUIRED
>

<!-- An order to be delivered will need delivery instructions -->
<!ELEMENT delivery (#PCDATA)>

<!-- Let's provide for any special instructions -->
<!ELEMENT special (#PCDATA)>

```

3. ADD A BURGER DESCRIPTION

Let's start working on our burger descriptions.

We could have a "burgers" element, with multiple nested "burger" elements, but it will be easier to provide for one or more "burger" elements at the order level.



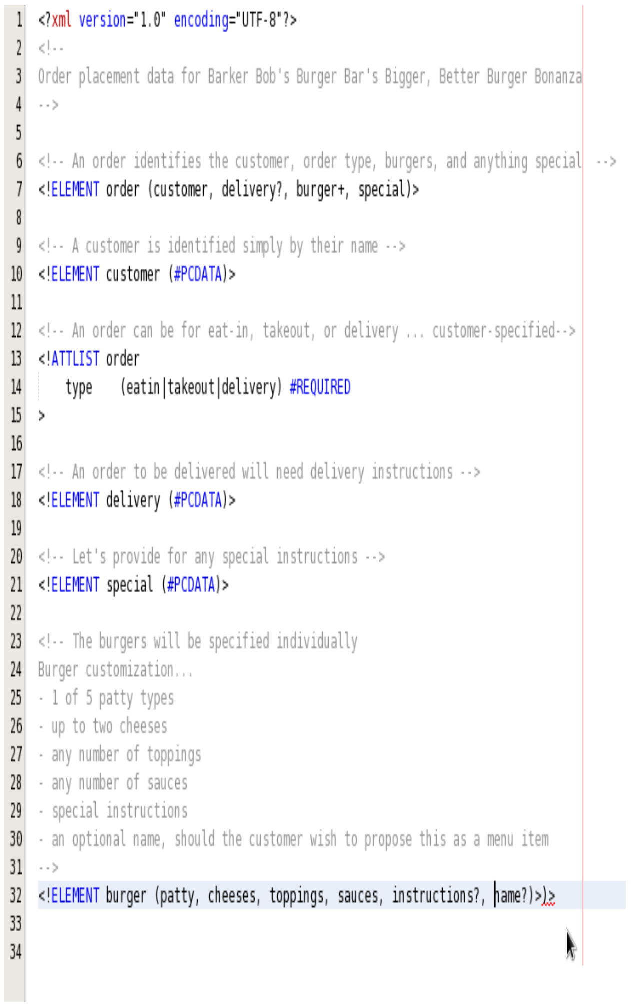
```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order placement data for Barker Bob's Burger Bar's Bigger, Better Burger Bonanza
-->

<!-- An order identifies the customer, order type, burgers, and anything special  -->
<!ELEMENT order (customer, delivery?, burger+, special)>

<!-- A customer is identified simply by their name -->
<!ELEMENT customer (#PCDATA)>

<!-- An order can be for eat-in, takeout, or delivery ... customer-specified-->
<!ATTLIST order
    type    (eatin|takeout|delivery) #REQUIRED
>

<!-- An order to be delivered will need delivery instructions -->
<!ELEMENT delivery (#PCDATA)>

<!-- Let's provide for any special instructions -->
<!ELEMENT special (#PCDATA)>

<!-- The burgers will be specified individually -->
<!ELEMENT burger (#PCDATA)>
```

Burger Structure

How do you make one of Barker Bob's burgers?

This makes sense at the moment.

Oops - I see an extra parenthesis and greater than sign at the end of the "burger" element directive. That will need to be removed!

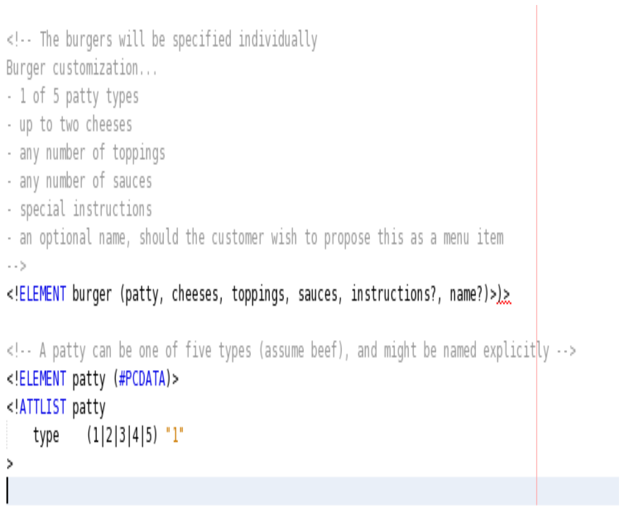


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order placement data for Barker Bob's Burger Bar's Bigger, Better Burger Bonanza
-->

<!-- An order identifies the customer, order type, burgers, and anything special  -->
<!ELEMENT order (customer, delivery?, burger+, special)>

<!-- A customer is identified simply by their name -->
<!ELEMENT customer (#PCDATA)>

<!-- An order can be for eat-in, takeout, or delivery ... customer-specified-->
<!ATTLIST order
    type    (eatin|takeout|delivery) #REQUIRED
>

<!-- An order to be delivered will need delivery instructions -->
<!ELEMENT delivery (#PCDATA)>

<!-- Let's provide for any special instructions -->
<!ELEMENT special (#PCDATA)>

<!-- The burgers will be specified individually
Burger customization...
- 1 of 5 patty types
- up to two cheeses
- any number of toppings
- any number of sauces
- special instructions
- an optional name, should the customer wish to propose this as a menu item
-->
<!ELEMENT burger (patty, cheeses, toppings, sauces, instructions?, name?)>)>
```

Burger Type

Each burger has a unique patty type, and we can use an attribute with an enumeration to ensure validity.

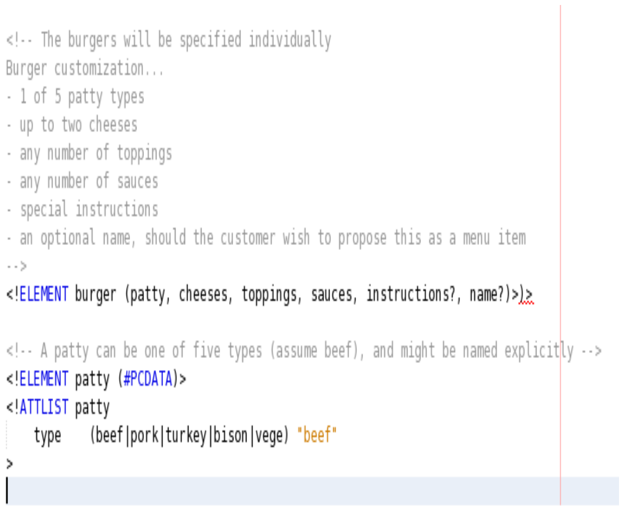
The default patty type is "beef", which would be the value "1" the way we have prescribed it.



```
<!-- The burgers will be specified individually
Burger customization...
- 1 of 5 patty types
- up to two cheeses
- any number of toppings
- any number of sauces
- special instructions
- an optional name, should the customer wish to propose this as a menu item
-->
<!ELEMENT burger (patty, cheeses, toppings, sauces, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
    type    (1|2|3|4|5) "1"
>
```

Burger Type Revisited

That was ugly. It would be better to spell out the patty types.



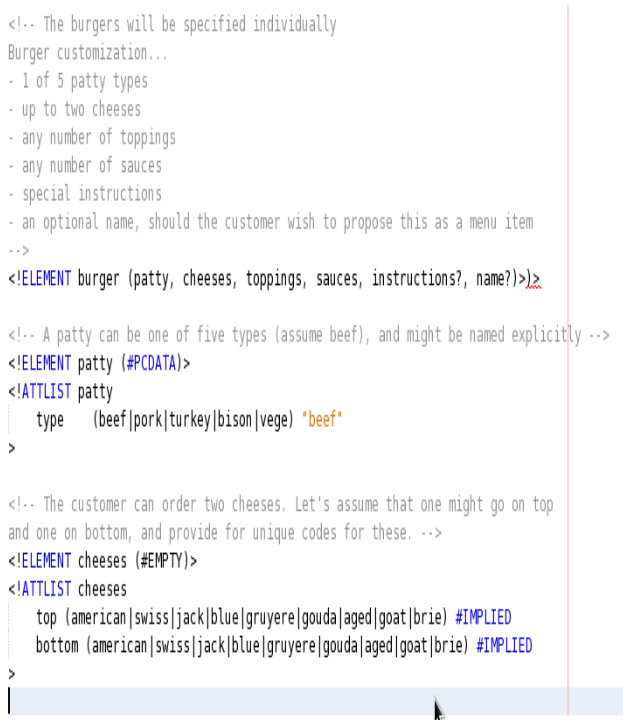
```
<!-- The burgers will be specified individually
Burger customization...
- 1 of 5 patty types
- up to two cheeses
- any number of toppings
- any number of sauces
- special instructions
- an optional name, should the customer wish to propose this as a menu item
-->
<!ELEMENT burger (patty, cheeses, toppings, sauces, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
    type    (beef|pork|turkey|bison|vege) "beef"
>
```

Cheese Toppings

Let's make a couple of assumptions to make our DTD more enforceable.

Too bad we have to repeat the cheese options, but this means that we just need the one "cheeses" element, with two attributes.

Oops - I see that the cheeses "content" is "(#EMPTY)" ... that will need to be changed to "EMPTY" at some point, before we can validate.



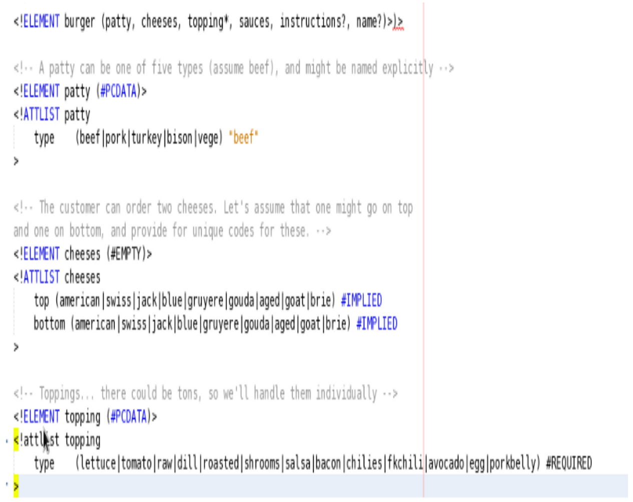
```
<!-- The burgers will be specified individually
Burger customization...
- 1 of 5 patty types
- up to two cheeses
- any number of toppings
- any number of sauces
- special instructions
- an optional name, should the customer wish to propose this as a menu item
-->
<!ELEMENT burger (patty, cheeses, toppings, sauces, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
    type    (beef|pork|turkey|bison|vege) "beef"
>

<!-- The customer can order two cheeses. Let's assume that one might go on top
and one on bottom, and provide for unique codes for these. -->
<!ELEMENT cheeses (#EMPTY)>
<!ATTLIST cheeses
    top (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
    bottom (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
>
```

Burger Toppings

Change the burger-level multiplicity.

Provide a long and perhaps silly list of toppings, er topping codes.

Note that we changed "toppings" to "topping*" in the "burger" element directive. A worthy simplification!

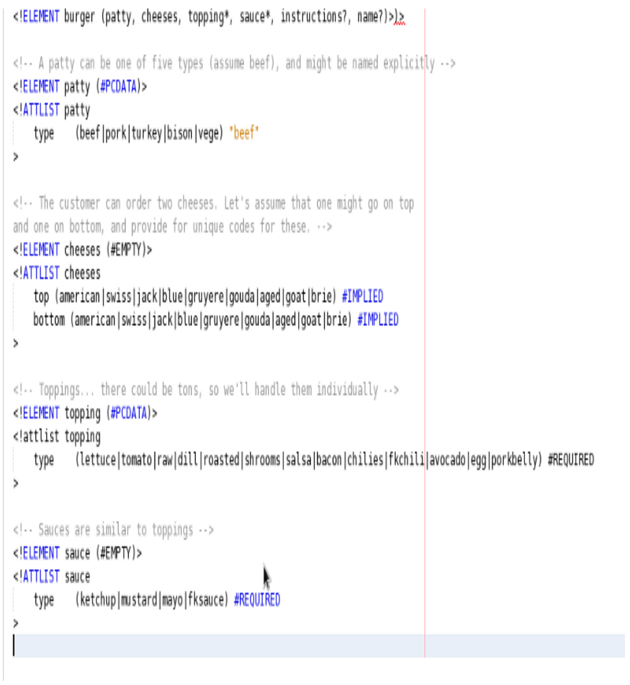


```
<!ELEMENT burger (patty, cheeses, topping*, sauces, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
    type    (beef|pork|turkey|bison|vege) "beef"
>

<!-- The customer can order two cheeses. Let's assume that one might go on top
and one on bottom, and provide for unique codes for these. -->
<!ELEMENT cheeses (#EMPTY)>
<!ATTLIST cheeses
    top (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
    bottom (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
>

<!-- Toppings... there could be tons, so we'll handle them individually -->
<!ELEMENT topping (#PCDATA)>
<!attlist topping
    type    (lettuce|tomato|raw|dill|roasted|shrooms|salsa|bacon|chilies|fkchili|avocado|egg|porkbelly) #REQUIRED
>
```

Burger Sauces

Do the same for sauces, adding an appropriate directive for one.

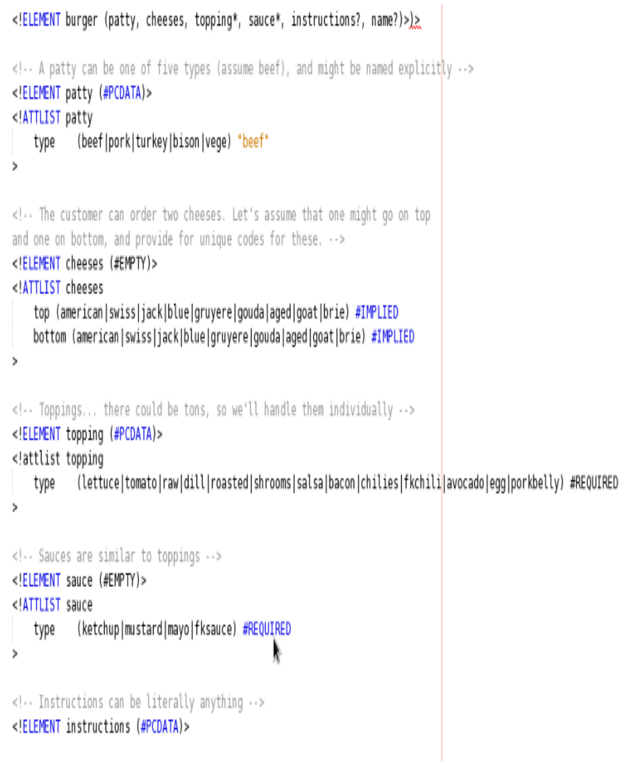
We changed "sauces" to "sauce*" in the "burger" directive.



```
<!ELEMENT burger (patty, cheeses, topping*, sauce*, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
    type    (beef|pork|turkey|bison|vege) "beef"
>

<!-- The customer can order two cheeses. Let's assume that one might go on top
and one on bottom, and provide for unique codes for these. -->
<!ELEMENT cheeses (#EMPTY)>
<!ATTLIST cheeses
    top (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
    bottom (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
>

<!-- Toppings... there could be tons, so we'll handle them individually -->
<!ELEMENT topping (#PCDATA)>
<!attlist topping
    type    (lettuce|tomato|raw|dill|roasted|shrooms|salsa|bacon|chilies|fkchili|avocado|egg|porkbelly) #REQUIRED
>

<!-- Sauces are similar to toppings -->
<!ELEMENT sauce (#EMPTY)>
<!ATTLIST sauce
    type    (ketchup|mustard|mayo|fksauce) #REQUIRED
>
```

Burger Instructions

Any special requests for this burger?

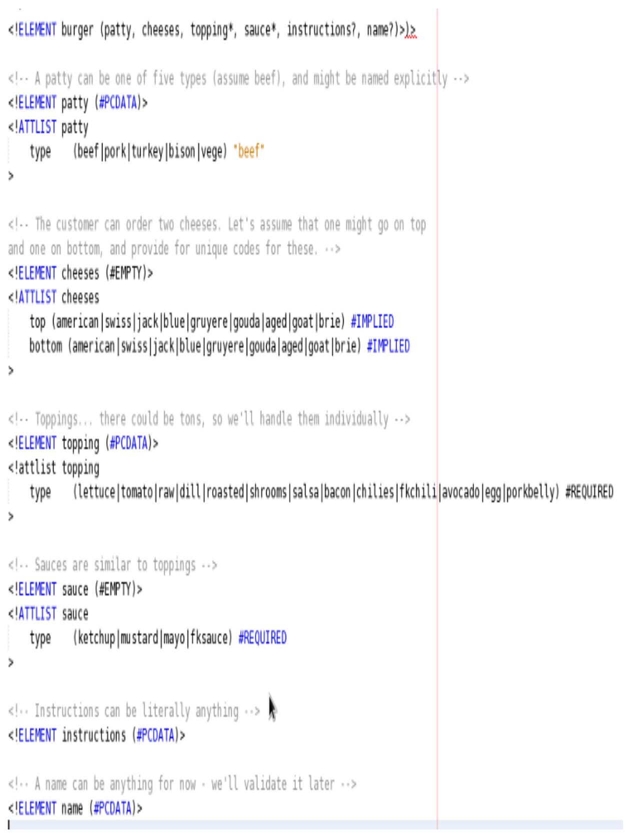


```
<!ELEMENT burger (patty, cheeses, topping*, sauce*, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
    type    (beef|pork|turkey|bison|vege) "beef"
>

<!-- The customer can order two cheeses. Let's assume that one might go on top
and one on bottom, and provide for unique codes for these. -->
<!ELEMENT cheeses (#EMPTY)>
<!ATTLIST cheeses
    top (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
    bottom (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
>

<!-- Toppings... there could be tons, so we'll handle them individually -->
<!ELEMENT topping (#PCDATA)>
<!attlist topping
    type    (lettuce|tomato|raw|dill|roasted|shrooms|salsa|bacon|chilies|fkchili|avocado|egg|porkbelly) #REQUIRED
>

<!-- Sauces are similar to toppings -->
<!ELEMENT sauce (#EMPTY)>
<!ATTLIST sauce
    type    (ketchup|mustard|mayo|fksauce) #REQUIRED
>

<!-- Instructions can be literally anything -->
<!ELEMENT instructions (#PCDATA)>
```

Burger Name

Marketing idea: let the customer name their creation

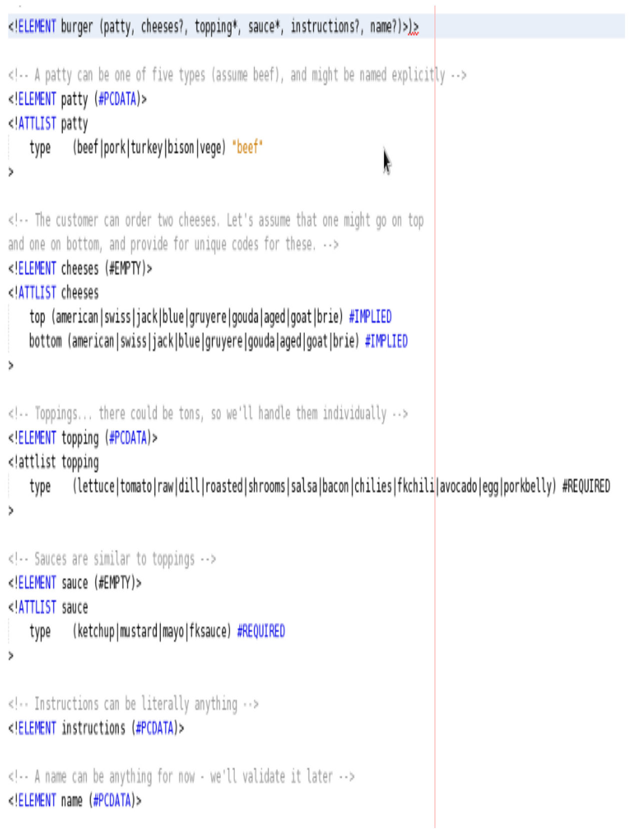
We're not doing anything with this right now, but we will next week :)



```
<!ELEMENT burger (patty, cheeses, topping*, sauce*, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
    type    (beef|pork|turkey|bison|vege) "beef"
>

<!-- The customer can order two cheeses. Let's assume that one might go on top
and one on bottom, and provide for unique codes for these. -->
<!ELEMENT cheeses (#EMPTY)>
<!ATTLIST cheeses
    top (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
    bottom (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
>

<!-- Toppings... there could be tons, so we'll handle them individually -->
<!ELEMENT topping (#PCDATA)>
<!attlist topping
    type    (lettuce|tomato|raw|dill|roasted|shrooms|salsa|bacon|chilies|fkchili|avocado|egg|porkbelly) #REQUIRED
>

<!-- Sauces are similar to toppings -->
<!ELEMENT sauce (#EMPTY)>
<!ATTLIST sauce
    type    (ketchup|mustard|mayo|fksauce) #REQUIRED
>

<!-- Instructions can be literally anything -->
<!ELEMENT instructions (#PCDATA)>

<!-- A name can be anything for now - we'll validate it later -->
<!ELEMENT name (#PCDATA)>
```

Are We There Yet?

Some minor adjustments, to get the DTD to validate...

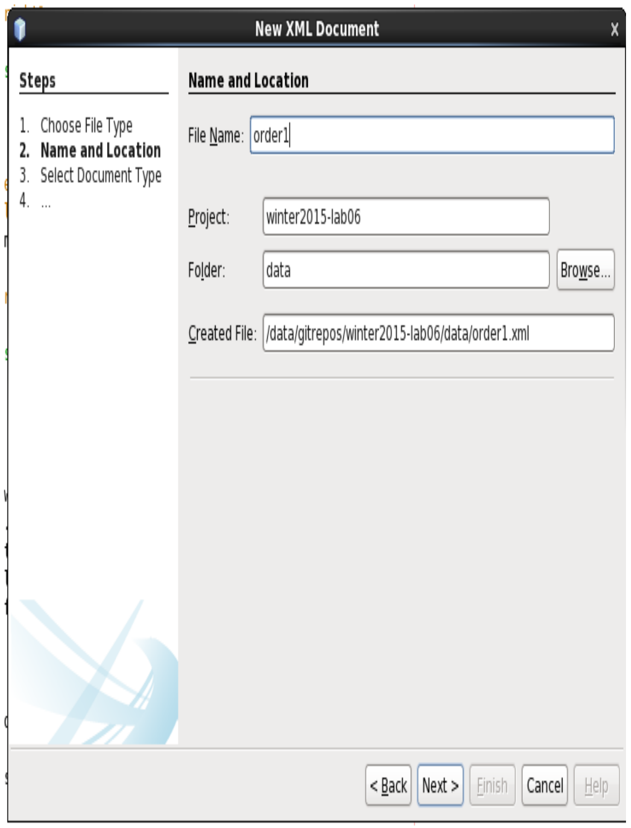


```
<!ELEMENT burger (patty, cheeses?, topping*, sauce*, instructions?, name?)>)>

<!-- A patty can be one of five types (assume beef), and might be named explicitly -->
<!ELEMENT patty (#PCDATA)>
<!ATTLIST patty
   type    (beef|pork|turkey|bison|vege) "beef"
>

<!-- The customer can order two cheeses. Let's assume that one might go on top
and one on bottom, and provide for unique codes for these. -->
<!ELEMENT cheeses (#EMPTY)>
<!ATTLIST cheeses
   top (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
   bottom (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
>

<!-- Toppings... there could be tons, so we'll handle them individually -->
<!ELEMENT topping (#PCDATA)>
<!attlist topping
   type    (lettuce|tomato|raw|dill|roasted|shrooms|salsa|bacon|chilies|fkchili|avocado|egg|porkbelly) #REQUIRED
>

<!-- Sauces are similar to toppings -->
<!ELEMENT sauce (#EMPTY)>
<!ATTLIST sauce
   type    (ketchup|mustard|mayo|fksauce) #REQUIRED
>

<!-- Instructions can be literally anything -->
<!ELEMENT instructions (#PCDATA)>

<!-- A name can be anything for now - we'll validate it later -->
<!ELEMENT name (#PCDATA)>
```

4. TEST YOUR ORDER

Let's build an order for a #8 burger.

Right-click on the data folder and choose new>XML Document...



File Name: order1

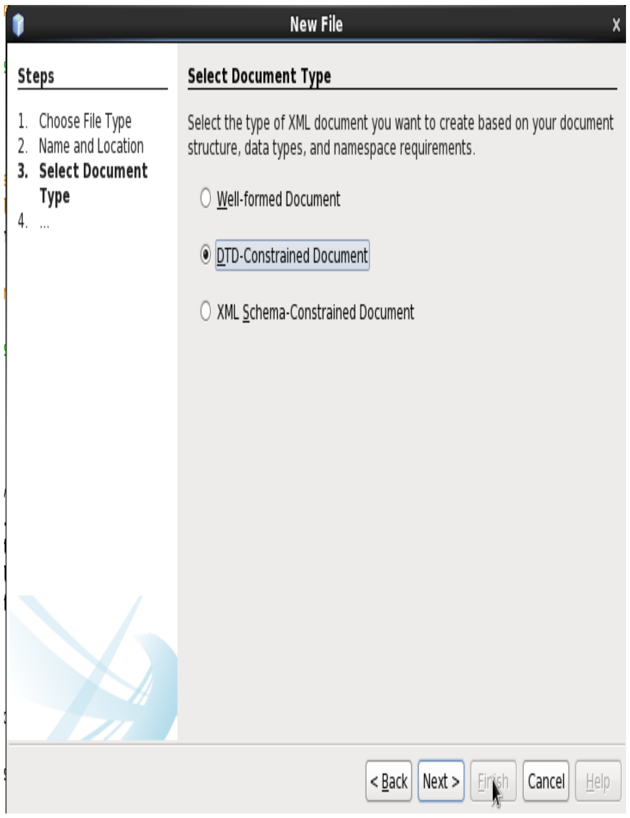
Project: winter2015-lab06

Folder: data

Created File: /data/gitrepos/winter2015-lab06/data/order1.xml

Order 1

We want it DTD-constrained



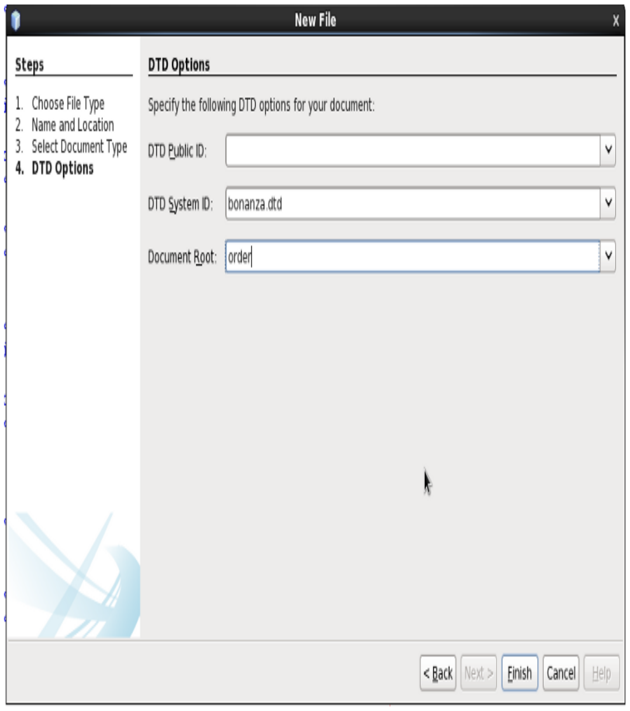
Select Document Type

Select the type of XML document you want to create based on your document structure, data types, and namespace requirements.

( ) Well-formed Document
(●) DTD-Constrained Document
( ) XML Schema-Constrained Document

Order 1

Bind it to our DTD



DTD Options

Specify the following DTD options for your document:

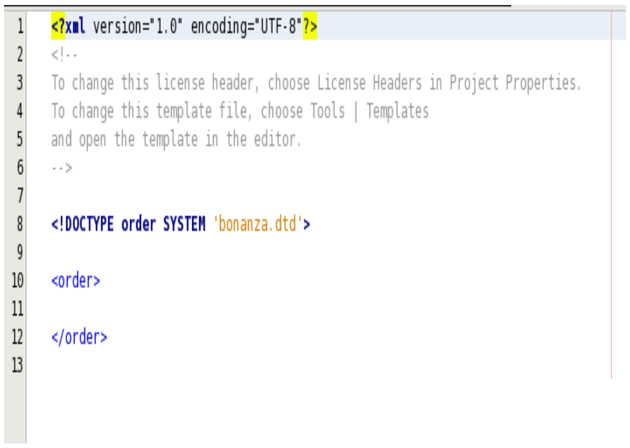
DTD Public ID:

DTD System ID: bonanza.dtd

Document Root: order

Order 1

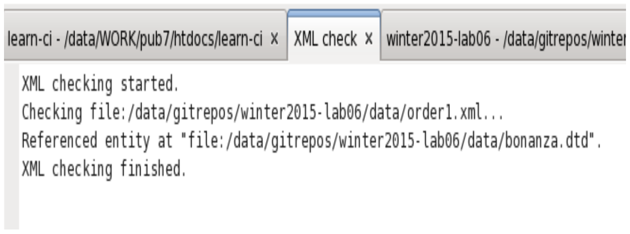
We have a starting point



```
<?xml version="1.0" encoding="UTF-8"?>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->

<!DOCTYPE order SYSTEM 'bonanza.dtd'>

<order>

</order>
```

Order 1

If we check it, it is well-formed

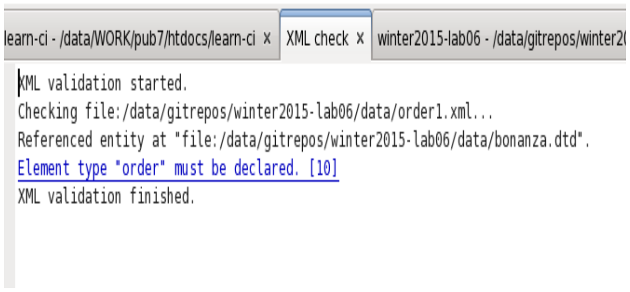


learn-ci - /data/WORK/pub7/htdocs/learn-ci ×
XML check ×
winter2015-lab06 - /data/gitrepos/winter

```
XML checking started.
Checking file:/data/gitrepos/winter2015-lab06/data/order1.xml...
Referenced entity at "file:/data/gitrepos/winter2015-lab06/data/bonanza.dtd".
XML checking finished.
```

Order 1

It doesn't validate yet ... the order needs a "tyoe" attribute, though this might not be apparent from the error message :-/

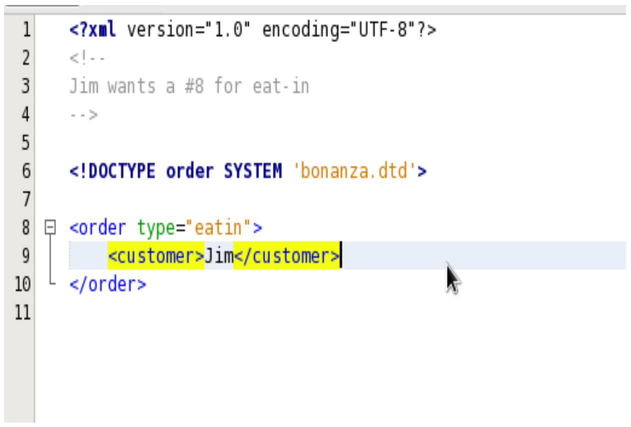


learn-ci - /data/WORK/pub7/htdocs/learn-ci ×
XML check ×
winter2015-lab06 - /data/gitrepos/winter2

```
XML validation started.
Checking file:/data/gitrepos/winter2015-lab06/data/order1.xml...
Referenced entity at "file:/data/gitrepos/winter2015-lab06/data/bonanza.dtd".
Element type "order" must be declared. [10]
XML validation finished.
```

Order 1

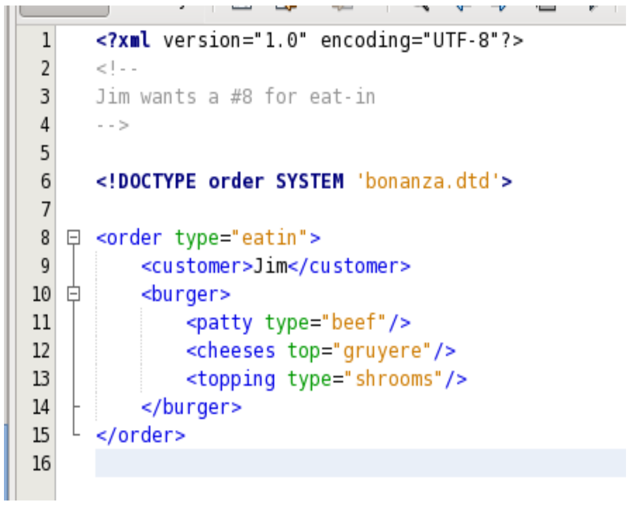
Let's start building our order



```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
Jim wants a #8 for eat-in
-->

<!DOCTYPE order SYSTEM 'bonanza.dtd'>

<order type="eatin">
    <customer>Jim</customer>
</order>
```

Order 1

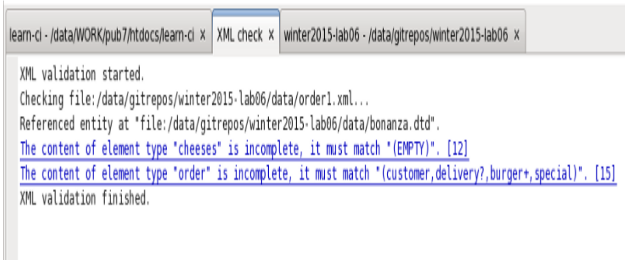
This looks good



```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
Jim wants a #8 for eat-in
-->

<!DOCTYPE order SYSTEM 'bonanza.dtd'>

<order type="eatin">
    <customer>Jim</customer>
    <burger>
        <patty type="beef"/>
        <cheeses top="gruyere"/>
        <topping type="shrooms"/>
    </burger>
</order>
```

Order 1

But it's not valid

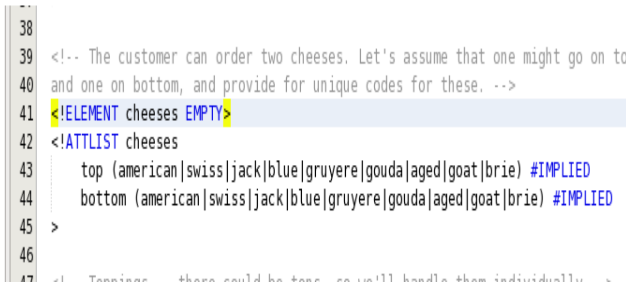


learn-ci - /data/WORK/pub7/htdocs/learn-ci x
XML check x
winter2015-lab06 - /data/gitrepos/winter2015-lab06 x

```
XML validation started.
Checking file:/data/gitrepos/winter2015-lab06/data/order1.xml...
Referenced entity at "file:/data/gitrepos/winter2015-lab06/data/bonanza.dtd".
The content of element type "cheeses" is incomplete, it must match "(EMPTY)". [12]
The content of element type "order" is incomplete, it must match "(customer,delivery?,burger+,special)". [15]
XML validation finished.
```

Order 1

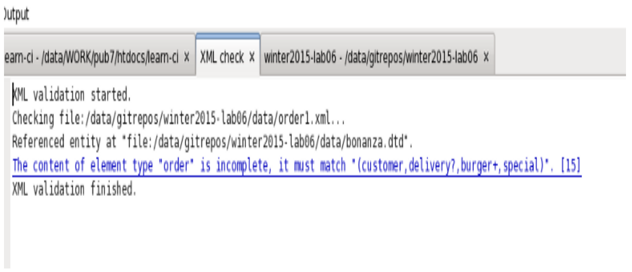
Oops - didn't specify the EMPTY cheeses element properly.



```
<!-- The customer can order two cheeses. Let's assume that one might go on to
and one on bottom, and provide for unique codes for these. -->
<!ELEMENT cheeses EMPTY>
<!ATTLIST cheeses
    top (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
    bottom (american|swiss|jack|blue|gruyere|gouda|aged|goat|brie) #IMPLIED
>

<!-- Toppings - there could be tons, so we'll handle them individually -->
```

Order 1

Better, but still not right

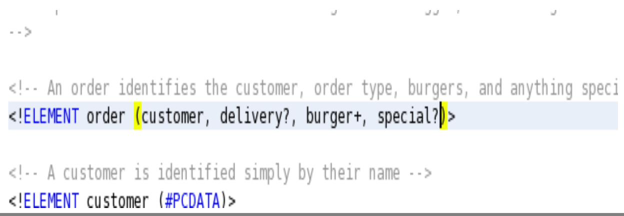


Output
earn-ci - /data/WORK/pub7/htdocs/learn-ci ×
XML check ×
winter2015-lab06 - /data/gitrepos/winter2015-lab06 ×

```
XML validation started.
Checking file:/data/gitrepos/winter2015-lab06/data/order1.xml...
Referenced entity at "file:/data/gitrepos/winter2015-lab06/data/bonanza.dtd".
The content of element type "order" is incomplete, it must match "(customer,delivery?,burger+,special)". [15]
XML validation finished.
```

Order 1

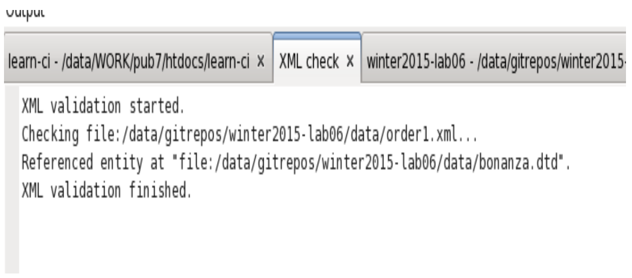
We needed to make the order special instructions optional



```
-->

<!-- An order identifies the customer, order type, burgers, and anything speci
<!ELEMENT order (customer, delivery?, burger+, special?)>

<!-- A customer is identified simply by their name -->
<!ELEMENT customer (#PCDATA)>
```

Order 1

This looks better



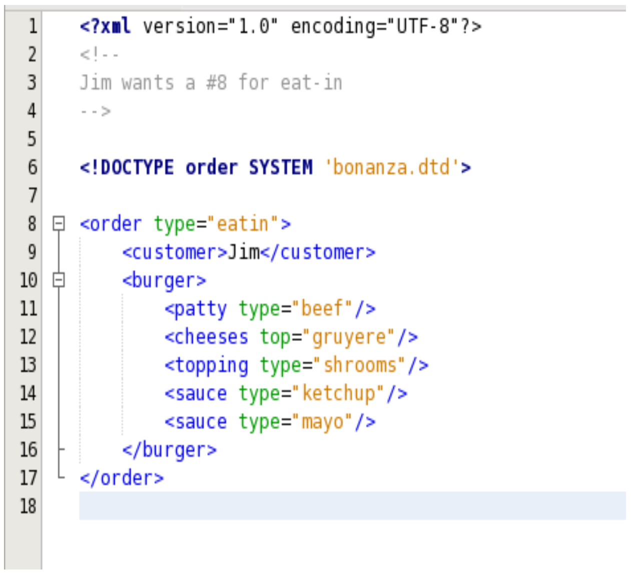
Output
learn-ci - /data/WORK/pub7/htdocs/learn-ci ×
XML check ×
winter2015-lab06 - /data/gitrepos/winter2015-

```
XML validation started.
Checking file:/data/gitrepos/winter2015-lab06/data/order1.xml...
Referenced entity at "file:/data/gitrepos/winter2015-lab06/data/bonanza.dtd".
XML validation finished.
```

Order 1

Add a couple of sauces.

This may not validate, but you know how to fix that!

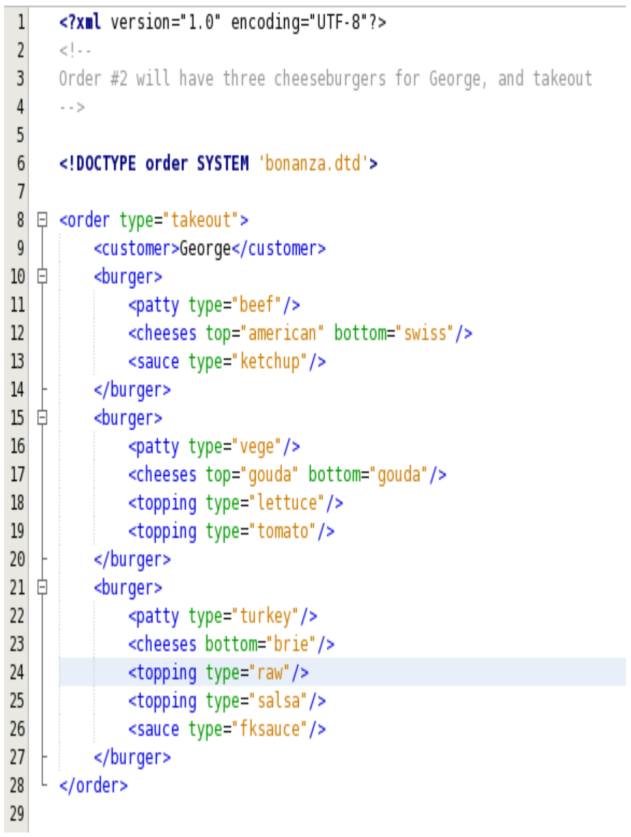


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Jim wants a #8 for eat-in
-->

<!DOCTYPE order SYSTEM 'bonanza.dtd'>

<order type="eatin">
    <customer>Jim</customer>
    <burger>
        <patty type="beef"/>
        <cheeses top="gruyere"/>
        <topping type="shrooms"/>
        <sauce type="ketchup"/>
        <sauce type="mayo"/>
    </burger>
</order>
```

Order 2

Let's make another order, #2, for George and takeout.

He wants three cheeseburgers. Makes sure it validates as you go.



```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order #2 will have three cheeseburgers for George, and takeout
-->

<!DOCTYPE order SYSTEM 'bonanza.dtd'>

<order type="takeout">
    <customer>George</customer>
    <burger>
        <patty type="beef"/>
        <cheeses top="american" bottom="swiss"/>
        <sauce type="ketchup"/>
    </burger>
    <burger>
        <patty type="vege"/>
        <cheeses top="gouda" bottom="gouda"/>
        <topping type="lettuce"/>
        <topping type="tomato"/>
    </burger>
    <burger>
        <patty type="turkey"/>
        <cheeses bottom="brie"/>
        <topping type="raw"/>
        <topping type="salsa"/>
        <sauce type="fksauce"/>
    </burger>
</order>
```

5. MAKE YOUR DREAM BURGER ORDER

You should have the hang of this by now!

Make an order for yourself, eat-in or takeout.

Construct the most awesome burger you can think of, with at least one cheese, three toppings, and two sauces.

One of your burgers should ask for peanut butter on it as well.

Feel free to have two burgers in your order!

Congratulations!

You have completed tutorial #tutorial06: Working With XML and DTDs

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: lab06 XML 2015.03.22 17:30

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.