Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Module: CSU22041
Date: 03/12/2021
Group: 21

Information Management XML Report

Contents

Abdelaziz Abushark : 20332134

Ali Al Ani : 19314029

Saif Ali : 203332469

Noel Austin : 20333159

Adam Beatty 20332203

Anastasiya Bogoslovskaya 20332234

1: Introduction

1.1 Background Research:

For the first six weeks of the semester, we were tasked with creating a UML use case diagram, class diagram and activity diagram. Once finished we were then asked to undertake an XML implementation for our domain.

Our group meetings were in person, as a team we discussed possible issues and problems that can arise during our conversion of UML to XML. We discussed problems in installing our XML software. We also considered possible problems that our UML domain may have had and may impact our XML experience. However, with help from demonstrators as well as the given examples of XML, DTD and xQuery projects, we found ourselves in a very comfortable position to convert our UML domain.

Our background research consisted of examples given to us from our instructors as well as a huge library of online resources that helped us make our lives easier during the transition. The blackboard tutorials and lectures were particularly helpful in getting us started.

1.2 Communication:

Communication can be considered the easiest part for us as a group as we had a compulsory tutorial at 5PM in Trinity on a Tuesday where we discussed issues, problems and solutions. We regularly used the app ‘WhatsApp’ to directly communicate with each other in a group chat format to ensure everyone is on the same page when we are not all present together on campus.

1.3 Platform and Research:

We used Google Docs to fill in our report. Google docs makes it very easy to work together as a team in writing a report as you can have all members writing simultaneously into the report.
Our software of choice was ‘BaseX’ which was recommended to us by our demonstrator during our first lab. Although it was slightly tricky to install, it was more than worth the effort as the editor was extremely useful in helping us write our XML files and allow us to test it smoothly.

1.4 Work Distribution:

During our first few labs in Trinity as well as using ‘WhatsApp’ we immediately distributed our work as evenly as possible. There are six team members and most tasks required six implementations which made our lives easier in terms of distribution. We assigned every group member to do at least 1 XML, a DTD document and it’s xQueries.

Abdelaziz Abushark : Whole report , Bank.XML , Bank.DTD ,and 2 XQueries for Bank which are bankCustomerDetails.xq , bankCustomerPaymentDetails.xq . Finder.XML , Finder.DTD and 1 XQuery which is finderResturantandFoodType.

Ali Al Ani : Whole report, Just_Eat.XML, Just_Eat.DTD, xQuery for Just_Eat.

Saif Ali : Strengths and weaknesses for XQueries and Customer.XML , Customer.DTD and 2 XQueries for Customer which are customerPhoneNumber.x q and customerBill.xq

Noel Austin : Settings.XML , Settings.DTD and 2 XQueries for settings which are checkIfValidPhoneNumber.xq , checkIfValidEmail.xq

Adam Beatty : order.XML, order.DTD and two XQueries for order.

Anastasiya Bogoslovskaya : Payment.XML , Payment.DTD and 2 XQueries for payment which are paymentTypes.xq and paymentSum.xq

2.XML Design:

2.1 What (if anything) did we need to change when going from a UML design to an XML implementation?

In order to make XML easier to understand, we had to make slight adjustments to our existing UML classes to suit our new XML model. The changes made included changing elements as well as adding new attributes/elements to meet the minimum/standard XML model requirements. We didn’t make any changes to our ethics model as we believed it was suitable for both our UML and XML documents.

2.2 XML and DTD documents

UML and XQuery Table from most to least important

XML Case	**Query**	**Related Use Case**	**For**	**Interlinked**	**User Defined Function**	**LET**	**Built in function**

Customer.xml	Total bill	Customer	X		X		
Bank.xml	Bank Details	Bank	X				
Bank.xml	Bank Payment Details	Bank	X				
Just Eat.xml	Just Eat Information	Just Eat	X	X			
Finder.xml	Finder for restaurant and food	Search	X	X			
Settings.xml	Settings to check if phone is valid	Settings				X	X
Settings.xml	Settings to check email	Settings				X	X
Payment.xml	Payment Sum	Payment	X			X	X
Payment.xml	Payment Types	Payment	X			X	X
Customer.xml	Customer phone number	Customer	X		X		
Order.xml	Order Type	Order	X				
Order.xml	Order Numbe	Order	X				

	r						
Just Eat.xml	Just Eat information	Just Eat	X	X			
Payment.xml	Max amount Paid	Payment	X			X	X

Bank:

Please note:
The addition of a bank name as an attribute to the Bank class.

```
<?xml version="1.0" ?>
<!DOCTYPE doc [
<!ELEMENT doc (payment* , bank*)>


<!ELEMENT payment ( bank*,amount? , type+)>
<!ELEMENT bank ( number*, name?, type+)>
<!ATTLIST doc type CDATA #REQUIRED >
<!ATTLIST doc xml:lang CDATA #IMPLIED >
<!ATTLIST bank name CDATA #IMPLIED >

<!ELEMENT amount (#PCDATA)  >
<!ELEMENT name (#PCDATA)    >
<!ELEMENT type (#PCDATA)    >
<!ELEMENT number (#PCDATA)  >


] >
<doc type="bank" xml:lang="en">
  <payment>
         <bank name="Allied Irish Banks">
           <number>20331234</number>
           <name>Folabi</name>
           <type>Student account</type>
        </bank>
<bank name="Bank Of Ireland">
           <number>20554553</number>
           <name>Mousa</name>
           <type>Business account</type>
        </bank>
<bank name="Permenant TSB">
           <number>20784553</number>
           <name>Anton</name>
```

```
            <type>Personal account</type>
         </bank>

               <amount>1000</amount>
               <type>Apple pay</type>
     </payment>
      <payment>


               <amount>12.95</amount>
               <type>Cash</type>
     </payment>
        <payment>
                <amount>43.65</amount>
               <type>card</type>
       </payment>

</doc>
```

Just_Eat:

Please note:
resutrant_search and food_search was added as an attribute.

```
<?xml version="1.0" ?>

<!DOCTYPE Just_Eat [
<!ELEMENT Just_Eat (Search*)>
<!ELEMENT Search (name+, email, phone, dateofbirth)>
<!ATTLIST Search restaurant_search CDATA #REQUIRED>
<!ATTLIST Search food_search CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
]>

<Just_Eat>
      <Search restaurant_search = 'Mad Egg' food_search = 'Beef Burger'>
             <name> Abdel Azuzzy </name>
             <email> elrayah@hotmail.com </email>
             <phone> "089334252 </phone>
             <dateofbirth> 9/01/2000 </dateofbirth>
      </Search>
      <Search restaurant_search = 'Burger King' food_search = 'Triple Cheesy
Whopper'>
             <name> Ben Dover </name>
             <email> bdove@gmail.com </email>
             <phone> 0872637478 </phone>
             <dateofbirth> 27/09/1998 </dateofbirth>
```

```
	</Search>
	<Search restaurant_search = 'Dominos Pizza' food_search = 'Mighty Meaty'>
		<name> John Kennedy </name>
		<email> jken@gmail.com </email>
		<phone> 0856974751 </phone>
		<dateofbirth> 06/03/1991 </dateofbirth>
	</Search>
</Just_Eat>
```

Finder:

Please note :
This class was changed from search to finder.
Food_search and resutrant_search were added as attribute.
Adding more details to search such as the orderID and orderNumber.

```
<?xml version="1.0" ?>
<!DOCTYPE Finder[
<!ELEMENT Finder (Search* )>
<!ELEMENT Search (orderID* , orderNumber*)>
<!ATTLIST Search restaurant_search CDATA #REQUIRED>
<!ATTLIST Search food_search CDATA #REQUIRED>
<!ELEMENT orderID (#PCDATA)>
<!ELEMENT orderNumber (#PCDATA)>


] >


<Finder>

<Search restaurant_search = 'Mad Egg' food_search = 'Beef Burger'>
<orderID>  2020213 </orderID>
<orderNumber> 20 </orderNumber>
</Search>
<Search restaurant_search = 'Burger King' food_search = 'Triple Cheesy Whopper'>
<orderID>  2020213 </orderID>
<orderNumber> 30 </orderNumber>
</Search>
<Search restaurant_search = 'Dominos Pizza' food_search = 'Mighty Meaty'>
<orderID>  2020213 </orderID>
<orderNumber> 40 </orderNumber>
</Search>
</Finder>
```

Customer:

Please note:

Interlinking payment , order , search and bank to customer XML document
Customer id was added as an attribute.

```
<?xml version="1.0"?>
<!DOCTYPE Doc[
<!ELEMENT Doc ( Customer* , Payment* , Order+ , Search+ , Bank*)>
<!ELEMENT Customer (first_name , last_name , email , phone_number , address)>
<!ELEMENT Payment ( amount+ , type* )>
<!ELEMENT Order (name+ , order_type* , number*)>
<!ELEMENT Search ( resturant_search+ , food_search+)>
<!ELEMENT Bank ( bank_Name* , account_Number* , account_Name* , account_Type*)>
<!ATTLIST Customer id CDATA #REQUIRED>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone_number (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT order_type (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT resturant_search (#PCDATA)>
<!ELEMENT food_search (#PCDATA)>
<!ELEMENT bank_Name (#PCDATA)>
<!ELEMENT account_Number (#PCDATA)>
<!ELEMENT account_Name (#PCDATA)>
<!ELEMENT account_Type (#PCDATA)>

]>



<Doc>



<Customer id = '22323'>
<first_name>Philip  </first_name>
<last_name> Tairov </last_name>
<email> tariovp@gmail.com </email>
<phone_number> 089433435 </phone_number>
<address> 1 spagethi mafia  </address>
</Customer>

<Payment>
<amount> 100 </amount>
<type> Apple Pay </type>
</Payment>
```

```
<Payment>
<amount> 12.95</amount>
<type> Cash </type>
</Payment>

<Payment>
<amount> 43.65 </amount>
<type> card </type>
</Payment>





<Order>
<name> Chicken Nuggets </name>
<order_type> Takeaway</order_type>
<number> 203992123 </number>
</Order>

<Search>
<resturant_search> Mad eggs </resturant_search>
<food_search> Beef Burger  </food_search>
</Search>


<Bank>
<bank_Name> Allied Irish Banks </bank_Name>
<account_Number> 20331234
</account_Number>
<account_Name> Folabi
</account_Name>
<account_Type> Student account </account_Type>
</Bank>

</Doc>
```

Order:

Please note:
Interlinked between payment and order xml documents.
Order_payment idref was added as an attribute.

```
<?xml version="1.0"?>
<!DOCTYPE Orders [
<!ELEMENT Orders (Order*)>
<!ELEMENT Order ( Order_name* , Order_type+ , Order_number* , Order_Payment*)>
<!ATTLIST Order_Payment idref CDATA #REQUIRED>
<!ELEMENT Order_name (#PCDATA)>
<!ELEMENT Order_type (#PCDATA)>
<!ELEMENT Order_number (#PCDATA)>
<!ELEMENT Order_Payment (#PCDATA)>


]>



<Orders>
  <Order>
    <Order_name>Chicken Nuggets</Order_name>
    <Order_type>Takeaway</Order_type>
    <Order_number>203992123</Order_number>
    <Order_Payment idref="201">
    </Order_Payment>
  </Order>
  <Order>
    <Order_name>Chicken Burger</Order_name>
    <Order_type>Delivery</Order_type>
    <Order_number>203992123</Order_number>
<Order_Payment idref="207">
    </Order_Payment>


  </Order>
</Orders>
```

Payment:

Please note:
Payment id attribute was added.
Payment type was added.

```
<?xml version = "1.0" encoding = "UTF-8"?>

<!DOCTYPE Payments[ <!ELEMENT Payments(Payment)>
<!ELEMENT Payment (payment_amount, payment_type)>
<!ELEMENT payment_amount (#PCDATA)>
<!ELEMENT payment_type (#PCDATA)>
<!ATTLIST Payment id CDATA #REQUIRED>
]>


<Payments>

<Payment id = "201">
<payment_amount> 100 </payment_amount>
<payment_type> Apple Pay </payment_type>
```

```
</Payment>

<Payment id = "101">
<payment_amount> 12.95</payment_amount>
<payment_type> Cash </payment_type>
</Payment>

<Payment id = "301">
<payment_amount> 43.65 </payment_amount>
<payment_type> card </payment_type>
</Payment>

</Payments>
```

Settings:

Please note:
Interlinks between settings and Just_eat xml documents.
Doc number was added as an attribute.
settings.User_Email was added new type.

```
<?xml version="1.0"?>
<!DOCTYPE doc[
<!ELEMENT doc ( settings, Just_eat )>
<!ELEMENT settings ( firstname*, lastname*, emailaddress?, password?,
phonenumber*, address)>
<!ELEMENT Just_eat ( settings.User_Name, settings.User_Email,
settings.User_Phone)>
<!ATTLIST doc number CDATA #REQUIRED>
<!ELEMENT settings.User_Name (#PCDATA)>
<!ELEMENT settings.User_Email (#PCDATA)>
<!ELEMENT settings.User_Phone (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT emailaddress (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT phonenumber (#PCDATA)>
<!ELEMENT address (#PCDATA)>


]>



<doc number = "2">

<settings>
 <firstname> Matt </firstname>
    <lastname> Murdock </lastname>
    <emailaddress> mattie@tcd.ie </emailaddress>
    <password> ******** </password>
    <phonenumber>0874532242</phonenumber>
    <address> 69 White Manor </address>
    </settings>
  <Just_eat>
<settings.User_Name> Abdel</settings.User_Name>
<settings.User_Email> abdel232@gmail.com </settings.User_Email>
<settings.User_Phone> 0834355532</settings.User_Phone>
  </Just_eat>


</doc>
```

2.4 Strengths and Weaknesses of the XML design

Strengths:

XML documents:

- You can add comments easily anywhere not like UML documents.
- Communication between the reader and the author.
- Converting from UML to XML was easy and straightforward (especially in xBase).
- DTD documents summarize everything that's included in the XML document.
- You can create your own tags that have the attributes you need and make sense to your users and you.
- It is 100 percent portable and fully compatible with JavaTM. Your data can be used by any application that can parse XML, regardless of platform.
- Easy language to be understood as anyone can read it and understand it as it uses human language.

Weaknesses:

XML documents:

- XML syntax can be hard to understand due to its redundancy.
- It does not offer much use for binary data as it is only useful for text data
- XML is not a small format. There is no standard XML compression technique. Uncompressed text is expected by XML parsers. Either you have to deal with big text

files or you have to build a sophisticated method for compressing and decompressing data on the fly, which will increase your processing time.
- XML documents can be very resource intensive due to the redundancy of the language.
- Lack of array support.

XQuery Design

3.1 xQueries , their supporting UML case and their purposes:

Bank Details:

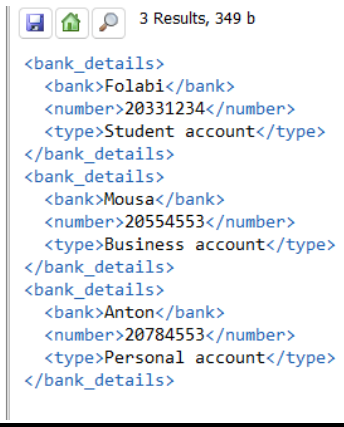
```
 for $x in doc("bankDone.xml")/doc/payment/bank
return
<bank details>
<bank>
{ string($x/name )}
</bank>
<number>
{ string($x/number) }
</number>
<type>
{ string($x/type) }
</type>

</bank details>
```

- Supporting UML Use-Case : Bank Details
- Purpose of Query : presents the customer’s name , number and bank account type

Example Output

Result


3 Results, 349 b

```
<bank_details>
  <bank>Folabi</bank>
  <number>20331234</number>
  <type>Student account</type>
</bank_details>
<bank_details>
  <bank>Mousa</bank>
  <number>20554553</number>
  <type>Business account</type>
</bank_details>
<bank_details>
  <bank>Anton</bank>
  <number>20784553</number>
  <type>Personal account</type>
</bank_details>
```

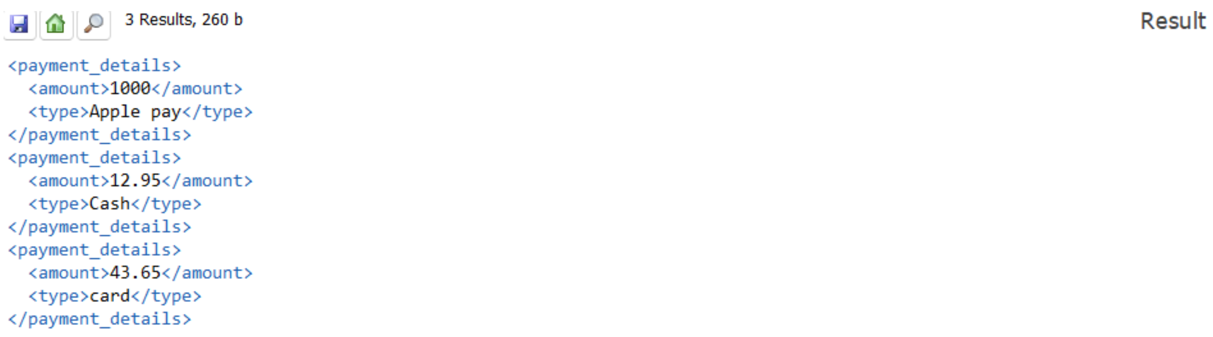
Payment Details :

```
for $x in doc("bankDone.xml")/doc/payment
return
<payment_details>
<amount>
{ string($x/amount) }
</amount>
<type>
{ string($x/type) }
</type>

</payment_details>
```

- **Supporting UML Use-Case** : Bank & Payment Details
- **Purpose of Query** : Present the payments amounts and types

Example output



```
3 Results, 260 b                                                    Result
<payment_details>
  <amount>1000</amount>
  <type>Apple pay</type>
</payment_details>
<payment_details>
  <amount>12.95</amount>
  <type>Cash</type>
</payment_details>
<payment_details>
  <amount>43.65</amount>
  <type>card</type>
</payment_details>
```

Max Payment paid by customer

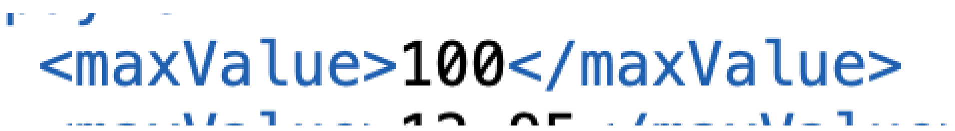
```
declare function local:payment_values()
{
for $payment-value in doc("payment.xml")/Payments/Payment/payment_amount

return
<payment_value>
   {$payment_value}
</payment_value>
};

<payment>
{local:payment_values()}
</payment>
```

- **Supporting UML Use-Case** : Bank & Payment Details
- **Purpose of Query** : Present the max payment

Example output



```
<maxValue>100</maxValue>
```

Just Eat Customer search

```
for $b in /Just_Eat/Search
    where $b/name = "John Kennedy" and $b/phone = 0856974751
      return
      <Search>
      <dob>{string ($b/dateofbirth)}</dob>
      <email>{string ($b/email)}</email>
      </Search>
```

- **Supporting UML Use-Case** : Just Eat
- **Purpose of Query :** Present the date of birth and email address of the customer

Example output

```
<Search>
  <dob>06/03/1991</dob>
  <email>jken@gmail.com</email>
</Search>
```

Just Eat Customer search

```
for $b in /Just_Eat/Search
    where $b/name = "Ben Dover" and $b/phone = 0872637478
      return
      <Search>
      <dob>{string ($b/dateofbirth)}</dob>
      <phone>{string ($b/phone)}</phone>
      </Search>
```

- **Supporting UML Use-Case** : Just Eat
- **Purpose of Query :** Present the date of birth and phone of the customer

Example output

```
<Search>
  <dob>27/09/1998</dob>
  <phone>0872637478</phone>
</Search>
```

Finder

```
for $b in /Finder/Search
    where $b/orderID = 2020213 and $b/orderNumber = 20
      return
      <Search>
      <OI>{string ($b/orderID)}</OI>
      <OD>{string ($b/orderNumber)}</OD>
      </Search>
```

- **Supporting UML Use-Case** : Finder
- **Purpose of Query :** Present the Order ID and Number of the order

Example output

```
<Search>
  <OI>2020213</OI>
  <OD>20</OD>
</Search>
```

Settings to check if phone number is valid

```
let $x := doc("settings.xml") /doc/settings/phonenumber
return
<phone_number_check>
{$x}
{if(fn:string-length($x) = 10)
then "this phone number is valid"
else "phone number is not valid"}
</phone_number_check>
```

- **Supporting UML Use-Case** : Settings
- **Purpose of Query :** Checks if the phone number is valid or not valid

Example output

```
<phone_number_check>
  <phonenumber>0874532242</phonenumber>this phone number is valid</phone_number_check>
```

Settings to check if email is valid

```
let $x := doc("settings.xml") /doc/settings/emailaddress
return
<email_address_check>
{$x}
{if(fn:contains($x, "@") )
then "email address is valid"
else "email address is not valid"}
</email_address_check>
```

- **Supporting UML Use-Case** : Settings
- **Purpose of Query :** Checks if the email is valid or not valid

Example output

```
<email_address_check>
  <emailaddress>mattie@tcd.ie</emailaddress>email address is valid</email_address_check>
```

Customer to search the phone number

```
declare function local:all_customers ()
{
  for $x in
doc("Customer/customer.xml")/Doc/Customer/phone_number
  return
  <phone>
  {($x)}
  </phone>
};
<all>
{local:all_customers() }
</all>
```

- **Supporting UML Use-Case** : Customer
- **Purpose of Query :** Customer phone number

Example output

```
<all>
  <phone>
    <phone_number>089433435</phone_number>
  </phone>
</all>
```

Customer to print the total bill

```
declare function local:find_id ($paymentT as xs:string)
{
for $j in
doc("Customer/customer.xml")/Doc/Payment
return
<bill>
{ string($j/amount )}
</bill>
};
local:find_id("All three orders")
```

- **Supporting UML Use-Case** : Customer
- **Purpose of Query :** Customer total bill for orders

Example output

```
<bill>100</bill>
<bill>12.95</bill>
<bill>43.65</bill>
```

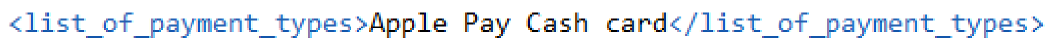
Customers Payment types for the orders

```
let $j := doc("payment.xml")/Payments/Payment/payment_type

return
<list_of_payment_types>
     {distinct-values($j)}
</list_of_payment_types>
```

- **Supporting UML Use-Case** : Payment
- **Purpose of Query :** type of payments used by customers

Example output



```
<list_of_payment_types>Apple Pay Cash card</list_of_payment_types>
```

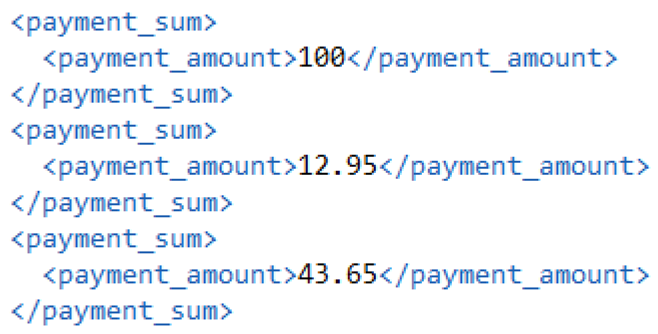
Customers Payment Sum

```
for $j in doc
doc("payment.xml")/Payments/Payment/payment_amount
order by $j/payment_amount descending

return
<payment_sum>
  {$j}
</payment_sum>
```

- **Supporting UML Use-Case** : Payment
- **Purpose of Query :** payment sum by customers

Example output



```
<payment_sum>
  <payment_amount>100</payment_amount>
</payment_sum>
<payment_sum>
  <payment_amount>12.95</payment_amount>
</payment_sum>
<payment_sum>
  <payment_amount>43.65</payment_amount>
</payment_sum>
```

Customer Order Type

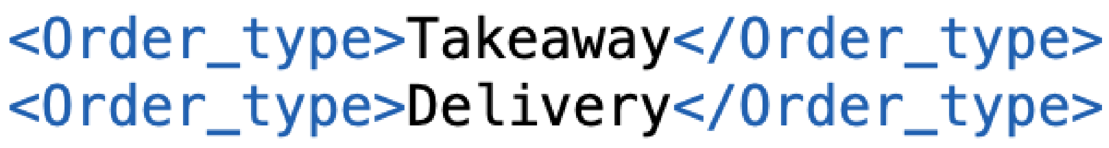
```
for $x in doc("order.xml")/Orders/Order
return
 <Order_type>

  {string($x/Order_type)}

</Order_type>
```

- **Supporting UML Use-Case** : Order
- **Purpose of Query :** present the customers order type

Example output



```
<Order_type>Takeaway</Order_type>
<Order_type>Delivery</Order_type>
```

Customer Order Number

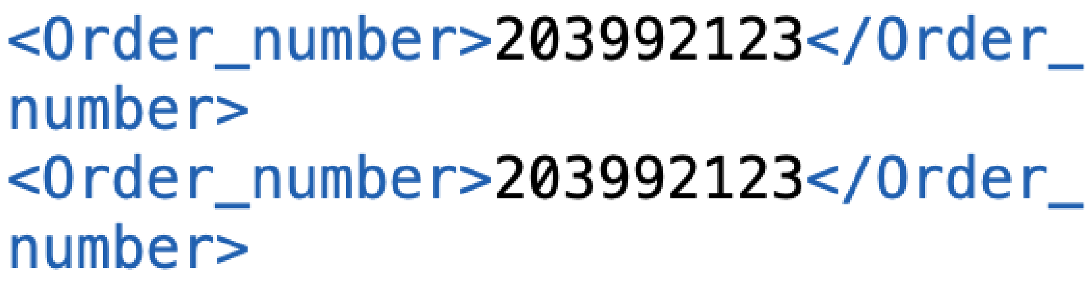
```
for $x in doc("order.xml")/Orders/Order
return
 <Order_number>

  {string($x/Order_number)}

</Order_number>
```

- **Supporting UML Use-Case** : Order
- **Purpose of Query :** present the customers order number

Example Output



```
<Order_number>203992123</Order_
number>
<Order_number>203992123</Order_
number>
```

3.2 Strengths and Weaknesses of the Query design

Strengths:

- It was easier for most people in our group to attempt the Xquery due to our previous experience and familiarity with other coding languages such as Java.
- Xquery allowed us to grab and output specific data that we needed from the XML databases.
- The Xqueries that we attempted can now be used as a foundation to build webpages.
- The Xquery has helped us to improve our problem solving skills as a group.

Weaknesses:

- Xqueries only allow us to pinpoint information that we have specifically designed for our XML databases, it does not take into account all cases.
- Xqueries lacks the features for dynamic polymorphism (i.e. allowing us to output data in many different types such as Java).
- At first glance, it was difficult for out group to understand and adapt to the syntax
- Xqueries can be considered a smaller language and because of this longer and more diverse tasks may be harder to perform using the XML database.

Reflection

After the great amount of work that we've put into this project. There is no doubt that we have learnt alot from it; both academically and personally. Thereafter, a glance at the trial exam papers, you can see that this project was the best integration for our preparation of the exam.

Looking back now, there are many changes that we wish we had made when facing this project. Our main factor of change would have been our organisation of time. Instead of our previous approach of having irregular meeting appointments, we should have co-ordinated on a set time and date each week to go back over our assignment. This would have assured us consistency during our time with this project.

Another aspect of change we would have gone through, is the background research undertaken for both the UML and XML aspects of the assignment. For instance, we believe that we should've undergone more in depth research about 'Just Eat' and how the organisation itself is managed and arranged. In terms of XML, it would have been great if each individual from our group had did more self research about the syntax and how XML is processed. From this it would have assured less time consumption to the adaption of the text editor

Nevertheless, we had great enjoyment undertaking this challenging yet satisfying project.

(: Thank you :)