HEUG
CANADA

THE XML'S OF PEOPLECODE

SESSION 6002
November 13, 2018
1:30PM – 2:30PM

PRESENTER

Adam Woodhouse

PeopleSoft Programmer/Analyst

Fleming College, Ontario

Website: www.flemingcollege.ca

E-mail: adam.woodhouse@flemingcollege.ca

FLEMING COLLEGE

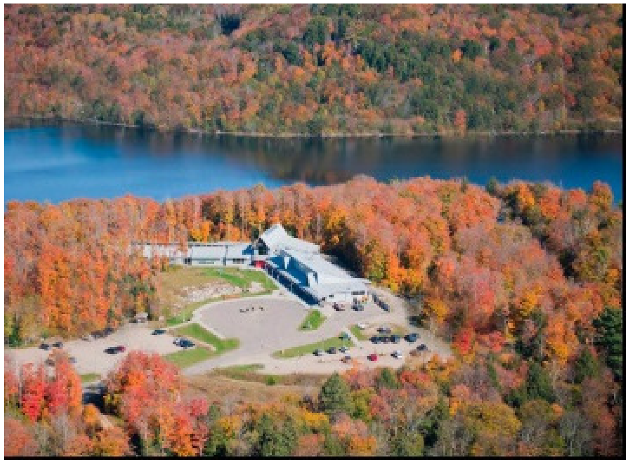
Located in Peterborough, Ontario, approx 90 min North-East drive from Toronto. Fleming was founded in 1967 and named after Sir Sandford Fleming, the founder of Universal Standard Time.

FLEMING COLLEGE: SUTHERLAND CAMPUS

Located in Peterborough is Flemings main campus. Notable programs are Police, Fire, Paramedic, Culinary, Computer Security & Investigations.

FLEMING COLLEGE: FROST CAMPUS

Located in Lindsay is the School of Environmental & Natural Resource Sciences. Notable programs are Fish & Wildlife, Heavy Machinery, Geographic Information Systems, Urban Forestry and the Centre for Advancement of Water & Wastewater Technologies.



FLEMING COLLEGE: HALIBURTON CAMPUS

Located in the beautiful Haliburton Highlands is the School of Arts + Design. A leader in unique arts programs like Glass Blowing, Blacksmith, Digital Imaging, Photo Arts, Visual & Creative Arts.

OUR PEOPLESOFT ENVIRONMENT

Human Capital 9.2

Campus Solutions 9.2

Finance 9.2

PeopleTools 8.55.20

Oracle 12c

OVERVIEW

In this session I will be covering the way Fleming College implemented a bolt-on solution to import and export XML files (also called a XML Message or XML Document).

The goods:

1. Who we do the data exchange with and why XML
2. Coding sample of the importing of XML files
3. Coding sample of the exporting of XML files

PRESENTATION GOAL

The goal of this presentation is to help PeopleSoft developers implement custom bolt-on programs for the purpose of XML data exchanges using PeopleSoft's Application Engine.

THE WHO & WHY

Why a custom bolt on to exchange XML files?

WHO IS OCAS?

- OCAS is an organization that is the focal point for students to apply to Ontario colleges
- OCAS = Ontario College Application Service
- Not only do student apply to Ontario colleges via OCAS, colleges exchange student information with other colleges via OCAS (i.e. student transcripts)
- OCAS sets the standard for how data will be exchanged with them
- Previously the 3rd party software EDI was licensed to exchange data between colleges and OCAS. The exchange was a text flat file
- OCAS is now pushing for all colleges to use the American PESC XML standard (Postsecondary Electronic Standards Council).

WHAT'S XML DEAL?

An intro to XML

CANADA ALLIANCE 12-14 NOVEMBER 2018

XML BASICS

- Let me share a few basic tidbits about XML from the website https://www.w3schools.com/xml
- XML stands for eXtensible Markup Language
- XML is Extensible
  - ➢ Most XML applications will work as expected even if new data is added or removed from the XML file and the application reading the file isn't changed to accommodate
- XML's presentation is much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive.

XML BASICS

With HTML (designed to display data), we work with predefined tags:

```
<html>
<body>
<h1 style="font-family:verdana;">This is a heading</h1>
<p style="font-family:courier;">This is a paragraph.</p>
</body>
</html>
```

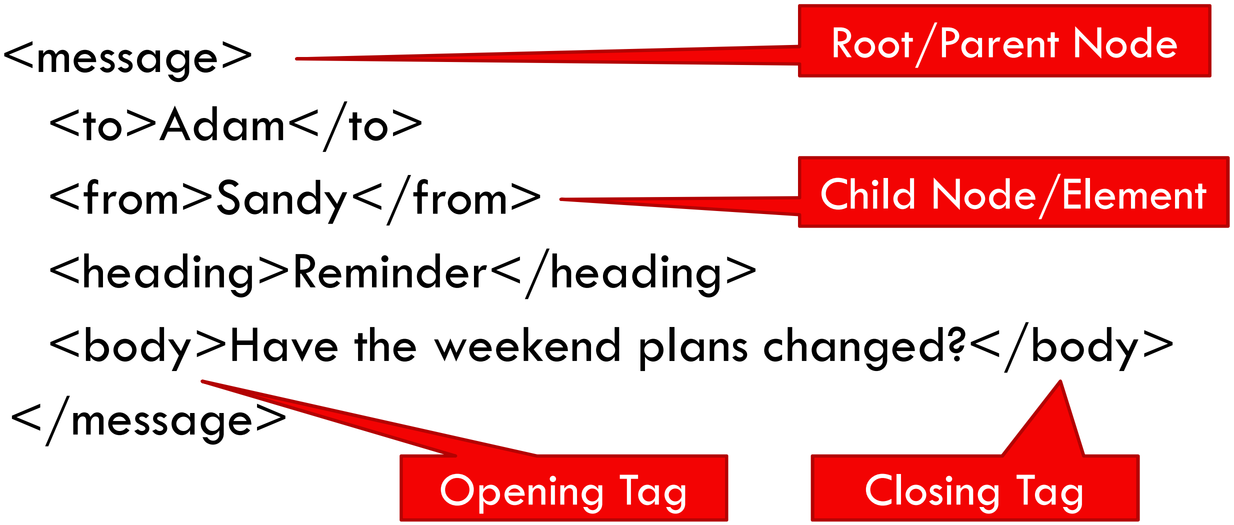
With XML (designed to carry data), we define the tags as we see fit:

```
<thisisthebook>
  <booktitle>XML and PeopleSoft</booktitle>
  <bookpublisher>Oracle Publishers</bookpublisher>
  <yearofrelease>2018</yearofrelease>
  <ISBNcode>123091209382</ISBNcode>
  <theblahblahblah>Misc text</theblahblahblah>
</thisisthebook>
```

You create the names of the tags

BASIC XML LAYOUT

The most common terminology used:



```
<message>
   <to>Adam</to>
   <from>Sandy</from>
   <heading>Reminder</heading>
   <body>Have the weekend plans changed?</body>
</message>
```

notes: i. XML tags are case sensitive

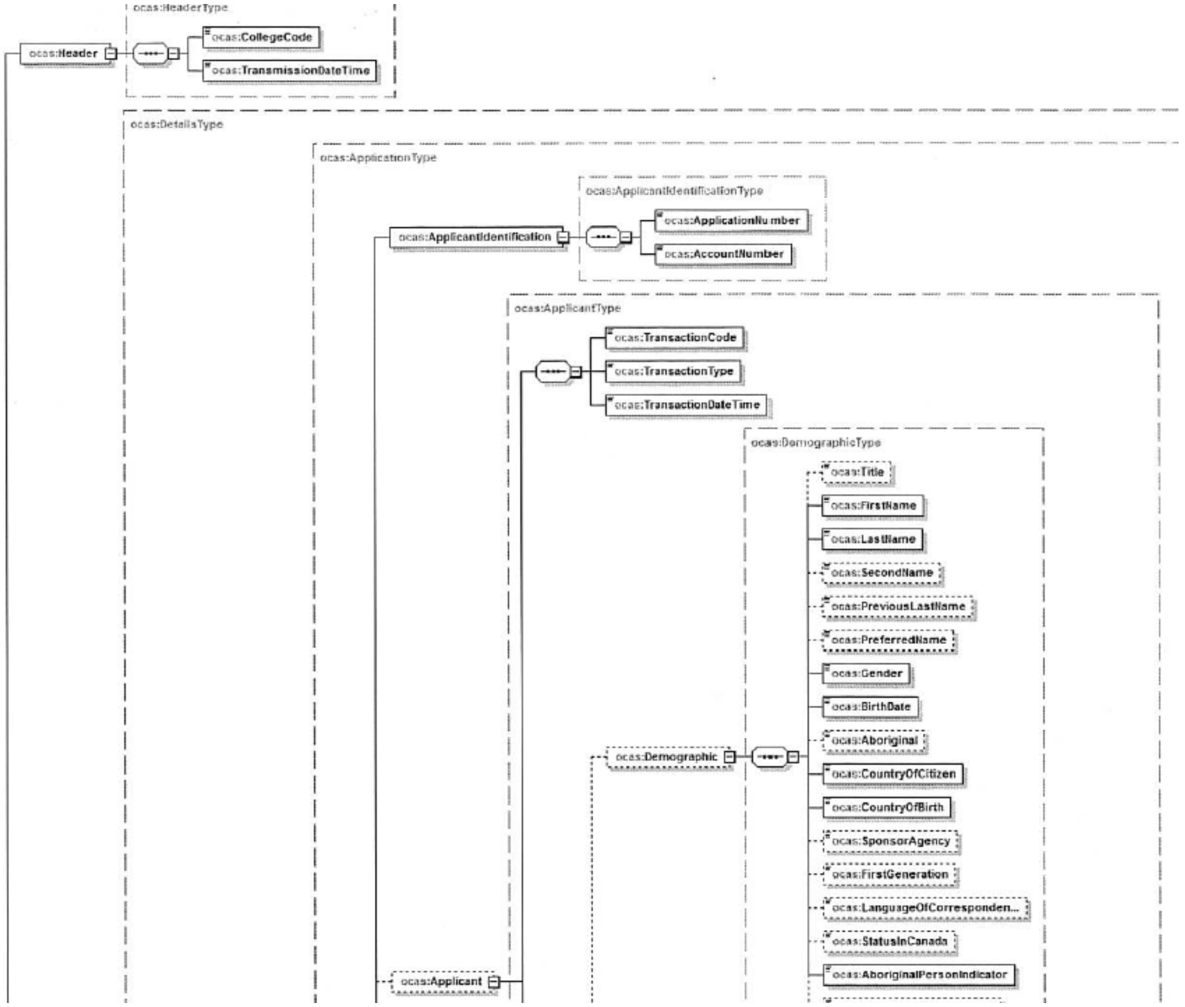
ii. The opening and closing tag of a node must be the same

iii. Make the tags something meaningful to the data it represents

XML INSTRUCTIONS – MY PROJECT

- At the beginning of the project we were provided instructions on the layout (schema) of the XML file
- Being new to XML I found the schema diagram confusing to interpret
- It was a single diagram spread across several pages in a PDF

SMALL PIECE OF SCHEMA DIAGRAM



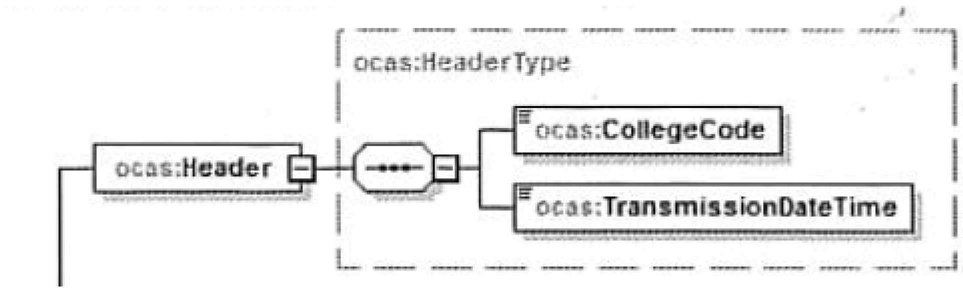
XML INSTRUCTIONS

- I found the schema diagram presented many questions such as how many times a certain piece of data could occur, its type, what constitutes a valid value, etc
- Fortunately in addition to the schema illustration we received a document that represented the XML layout in a traditional file format presentation (a table) which helped me to better visualize what the file could* contain

** I say could contain because if a piece of data isn't mandatory, an XML file may not include that data or its tags; unlike in a comma delimited file where the field would still exist but have no value.*

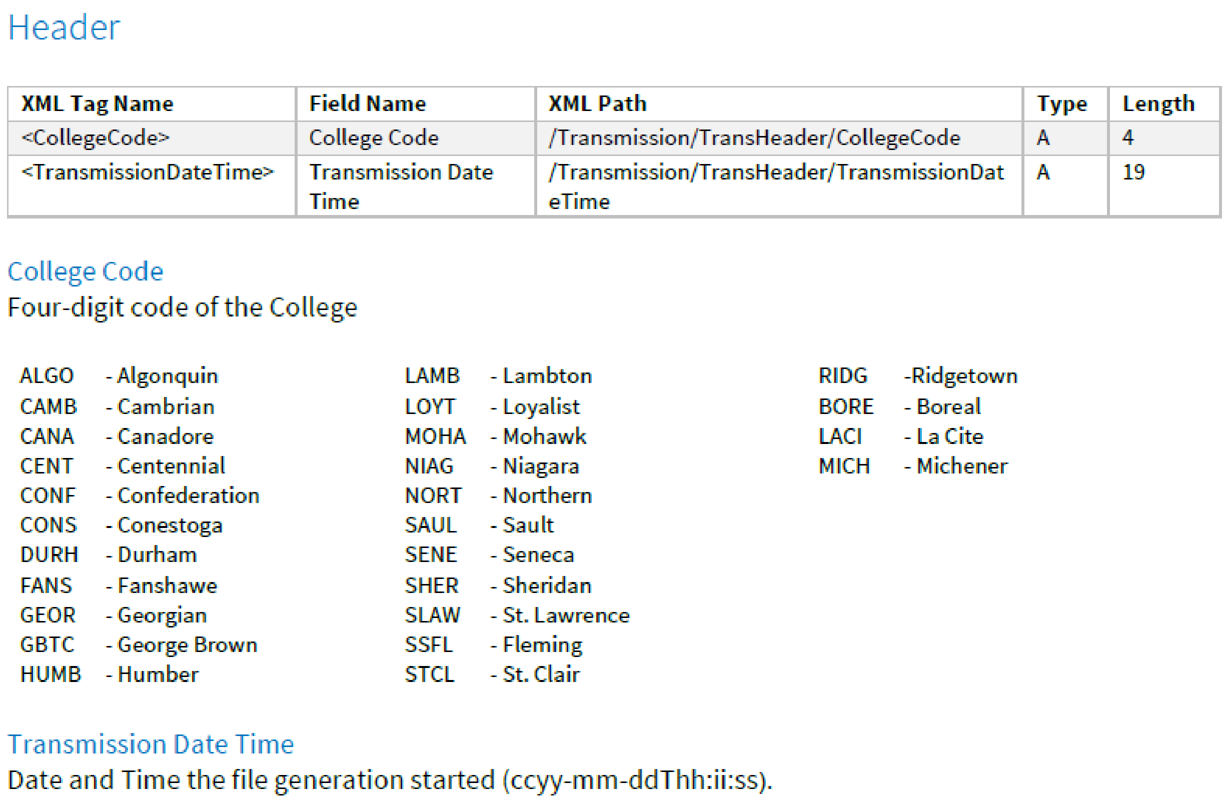
SMALL PIECE OF SCHEMA DIAGRAM

- Here is a small piece from the previous schema diagram



SMALL PIECE OF SCHEMA DIAGRAM

- This was in the 2nd document



Header

XML Tag Name	Field Name	XML Path	Type	Length
<CollegeCode>	College Code	/Transmission/TransHeader/CollegeCode	A	4
<TransmissionDateTime>	Transmission Date Time	/Transmission/TransHeader/TransmissionDateTime	A	19

College Code

Four-digit code of the College

ALGO	- Algonquin	LAMB	- Lambton	RIDG	-Ridgetown
CAMB	- Cambrian	LOYT	- Loyalist	BORE	- Boreal
CANA	- Canadore	MOHA	- Mohawk	LACI	- La Cite
CENT	- Centennial	NIAG	- Niagara	MICH	- Michener
CONF	- Confederation	NORT	- Northern		
CONS	- Conestoga	SAUL	- Sault		
DURH	- Durham	SENE	- Seneca		
FANS	- Fanshawe	SHER	- Sheridan		
GEOR	- Georgian	SLAW	- St. Lawrence		
GBTC	- George Brown	SSFL	- Fleming		
HUMB	- Humber	STCL	- St. Clair		

Transmission Date Time

Date and Time the file generation started (ccyy-mm-ddThh:ii:ss).

XML HEADER SAMPLE

- When I looked at the sample XML file provided I saw this at the top of the file:

```
<Header>
  <CollegeCode>SSFL</CollegeCode>
  <TransmissionDateTime>2018-08-03T09:42:30</TransmissionDateTime>
</Header>
```

- When previously in the flat text file the data would look something like this:

```
HSSFL20180803T094230
```

Header Indicator

SAMPLE DATA & PEOPLECODE

Using App Engine to upload multiple XML files

XML DATA SAMPLE

Root node.
Only occurs once

This parent node will
repeat for each student

A Student

```
<Student_Data>

   <Applicant>

        <ID>123456</ID>

        <Demographic>

            <FirstName>Frankie</FirstName>
            <LastName>Bates</LastName>
            <Gender>M</Gender>
            <BirthDate>1960-03-01</BirthDate>

        </Demographic>

       <Email>

           <EmailAddress>personal@gmail.com</EmailAddress>
           <EmailAddress>work@hotmail.com</EmailAddress>

        </Email>

  </Applicant>

</Student_Data>
```

THE DESTINATION RECORDS

- FC_STUDENT

FIELDS
EMPLID
LASTNAME
FIRSTNAME

- FC_EMAIL

FIELDS
EMPLID
EMAIL_ADDRESS

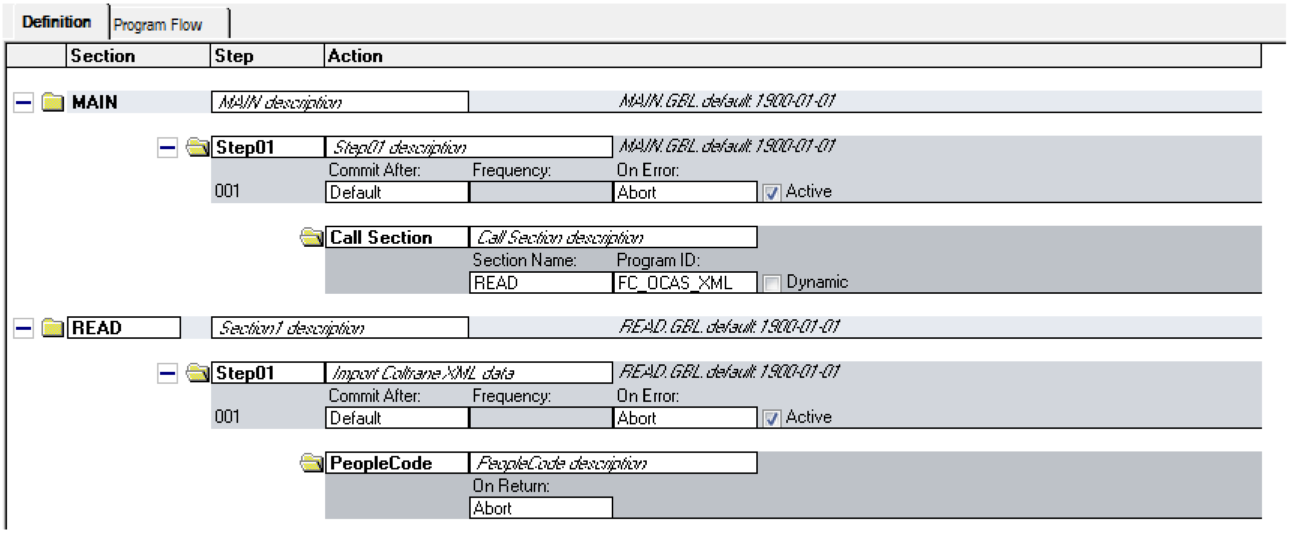
LOAD XML FILE - PSEUDOCODE

- This is a high level of what the App Engine will do

```
Create the instance of an XML object
Create a log file
Count how many XML files there are for this upload
Loop through XML files
  Point to the XML file
  Count how many students exist in the XML file
  Loop through each student
    Load basic student data in custom STUDENT record
    Write to log file
    Count occurrences of email address
    Loop through each email address
      Load student email in custom EMAIL record
      Write to log file
    End email loop
  End student loop
End XML file loop
```

PEOPLECODE – APP ENGINE

- I wrote an App Engine to process XML files
- There are many different ways an App Engine can be constructed to process an XML file
- I choose the simplest route and did the processing in a single Step



MAIN — *MAIN description* — *MAIN.GBL.default.1900-01-01*

Step01 — *Step01 description* — *MAIN.GBL.default.1900-01-01*

001 — Commit After: Default — Frequency: — On Error: Abort — ☑ Active

Call Section — *Call Section description*

Section Name: READ — Program ID: FC_OCAS_XML — □ Dynamic

READ — *Section1 description* — *READ.GBL.default.1900-01-01*

Step01 — *Import Coltrane XML data* — *READ.GBL.default.1900-01-01*

001 — Commit After: Default — Frequency: — On Error: Abort — ☑ Active

PeopleCode — *PeopleCode description*

On Return: Abort

PEOPLECODE TO LOAD A XML FILE

- What follows is the PeopleCode to load a simple XML file into two PeopleSoft records

```
Local XmlDoc &inXMLDoc;
Local XMLNode &StudentInfo;
Local boolean &return_code;
Local array of XmlNode &Node_arrApplicant, &Node_arrEmailAddress;

/* Define a log file to record the raw data */

&LOGFILE = GetFile("/fileserver_path/xml_raw_data_log_file.txt", "W",
%FilePath_Absolute);

&LOGFILE.WriteLine("Start of Process ......" | %Datetime);

&LOGFILE.WriteLine(" ");
```

PEOPLECODE TO LOAD A XML FILE

```
/* In this example we have received multiple XML files. */
/* Build an array of the file names so that we can loop through them. */


&arrFILES = FindFiles ("/fileserver_path/students_file*.xml", %FilePath_Absolute);

&FILECOUNT = &arrFILES.Len;  /* How many occurrences of the XML files exist. */

&LOGFILE.WriteLine("Number of files to process = " | &FILECOUNT);

&LOGFILE.WriteLine(" ");

/* Define objects to be used */

&inXMLDoc = CreateXmlDoc(""); /* Instantiate an XML object */

&FC_STUDENT = CreateRecord(Record.FC_STUDENT); /* We will load demographic
data to this PeopleSoft record */
&FC_EMAIL = CreateRecord(Record.FC_EMAIL); /* We will load e-mail data to this
Peoplesoft record */
```

PEOPLECODE TO LOAD A XML FILE

```
/* Loop through each XML file */

For &loop_files = 1 To &FILECOUNT

   &READFILE = GetFile(&arrFILES.Shift(), "R", %FilePath_Absolute); /* Get the XML file from the array. */

   &FILENAME = &READFILE.Name;

   &LOGFILE.WriteLine(“XML File = " | &FILENAME);

   /* Establish the input file is an XML document */

   &return_code = &inXMLDoc.ParseXmlFromURL(&FILENAME);

   /* Recognize the XML document nodes */

   &InputFile = &inXMLDoc.DocumentElement;
```

PEOPLECODE TO LOAD A XML FILE

```
If &return_code Then /* If a valid XML file, continue */
    &LOGFILE.WriteLine(“File passed XML parser, processing … “);

    /* How many students are in this file to be processed? */
    &Node_arrApplicant = &InputFile.GetElementsByTagName(“Applicant");
    &LOGFILE.WriteLine("Students to process (occurrences of node <Applicant>) = " |
    &Node_arrApplicant.Len);

    /* Perform the below loop for every instance of a student (the occurrence of <Applicant>) */
    For &I = 1 To &Node_arrApplicant.Len /* Loop through every occurrence of a student */
      &StudentInfo = &Node_arrApplicant.Get(&I); /* Point the object &StudentInfo to the student
      in the XML file we are processing */
      &Node_ID = &StudentInfo.FindNode("ID"); /* Find node ID */
      &FC_STUDENT.EMPLID.Value = &Node_ID.NodeValue; /* Move the data in node ID to the
      record.field */
      &LOGFILE.WriteLine(“XML <ID> = " | &Node_ID.NodeValue);
```

PEOPLECODE TO LOAD A XML FILE

```
/* Navigate to node Demographic and process child nodes */
&Node_Demographic = &StudentInfo.FindNode("Demographic"); /* Find node Demographic */

&Node_FirstName = &Node_Demographic.FindNode(“FirstName"); /* Find node FirstName */
&FC_STUDENT.FIRSTNAME.Value = &Node_FirstName.NodeValue; /* Move the data in node
FirstName to the record.field */
&LOGFILE.WriteLine(“XML <FirstName> = " | &Node_FirstName.NodeValue);

&Node_LastName = &Node_Demographic.FindNode(“LastName"); /* Find node LastName */
&FC_STUDENT.LASTNAME.Value = &Node_LastName.NodeValue; /* Move the data in node
LastName to the record.field */
&LOGFILE.WriteLine(“XML <LastName> = " | &Node_LastName.NodeValue);
```

note: same process would apply for Gender and DOB. But … if we didn’t code for those nodes it wouldn’t matter to the program, the program would still run!

PEOPLECODE TO LOAD A XML FILE

```
/* Write to the database the values for the record FC_STUDENT. */

  If &FC_STUDENT.Insert() = False Then

        &LOGFILE.WriteLine(“The insert to the record FC_STUDENT failed for EMPLID = ” |
        &Node_ID.NodeValue);

  End-If;


  /* The next piece of code is to read the e-mail addresses */
```

XML DATA SAMPLE

```
<Student_Data>

  <Applicant>

      <ID>13551321231</ID>

      <Demographic>

         <FirstName>Frankie</FirstName>
         <LastName>Bates</LastName>
         <Gender>M</Gender>
         <BirthDate>1960-03-01</BirthDate>

      </Demographic>

     <Email>

        <EmailAddress>personal@gmail.com</EmailAddress>
        <EmailAddress>work@hotmail.com</EmailAddress>

      </Email>

 </Applicant>

</Student_Data>
```

PEOPLECODE TO LOAD A XML FILE

```
  /* For multiple e-mail nodes, we need to see how many occurrences and loop through them. */

     &FC_EMAIL.EMPLID.Value = &Node_ID.NodeValue; /* Use this earlier established node */
     &Node_Email = &StudentInfo.FindNode(“Email"); /* Find parent node Email */
     &Node_arrEmailAddress = &Node_Email.FindNodes(“EmailAddress”); /* Find child nodes */
     For &E = 1 to &Node_arrEmailAddress.Len /* Loop through all of the EmailAddress nodes */
       &Node_EmailAddress = &Node_arrEmailAddress.Get(&E);
       &FC_EMAIL.EMAIL.Value = &Node_EmailAddress.FindNode(“EmailAddress”).NodeValue;
       If &FC_EMAIL.Insert() = False Then
           &LOGFILE.WriteLine(“The insert to the record FC_EMAIL failed for EMPLID = ” |
           &Node_ID.NodeValue);
       End-If;
     End-For; /* For &E = 1to &Node_arrEmailAddress.Len  - loop through email addresses */

   End-For; /* For &I = 1 To &Node_arrApplicant.Len – loop through each student */

End-If; /* If &return_code Then – checked to see if a XML file */
```

THE DATA WRITTEN TO 2 TABLES

- FC_STUDENT

EMPLID	LAST_NAME	FIRST_NAME
13551321231	Bates	Frankie

- FC_EMAIL

EMPLID	EMAIL_ADDR
13551321231	personal@gmail.com
13551321231	work@hotmail.com

CREATING A XML FILE

Using App Engine to create a single XML file

THE DATA FOR OUR OUTPUT XML FILE

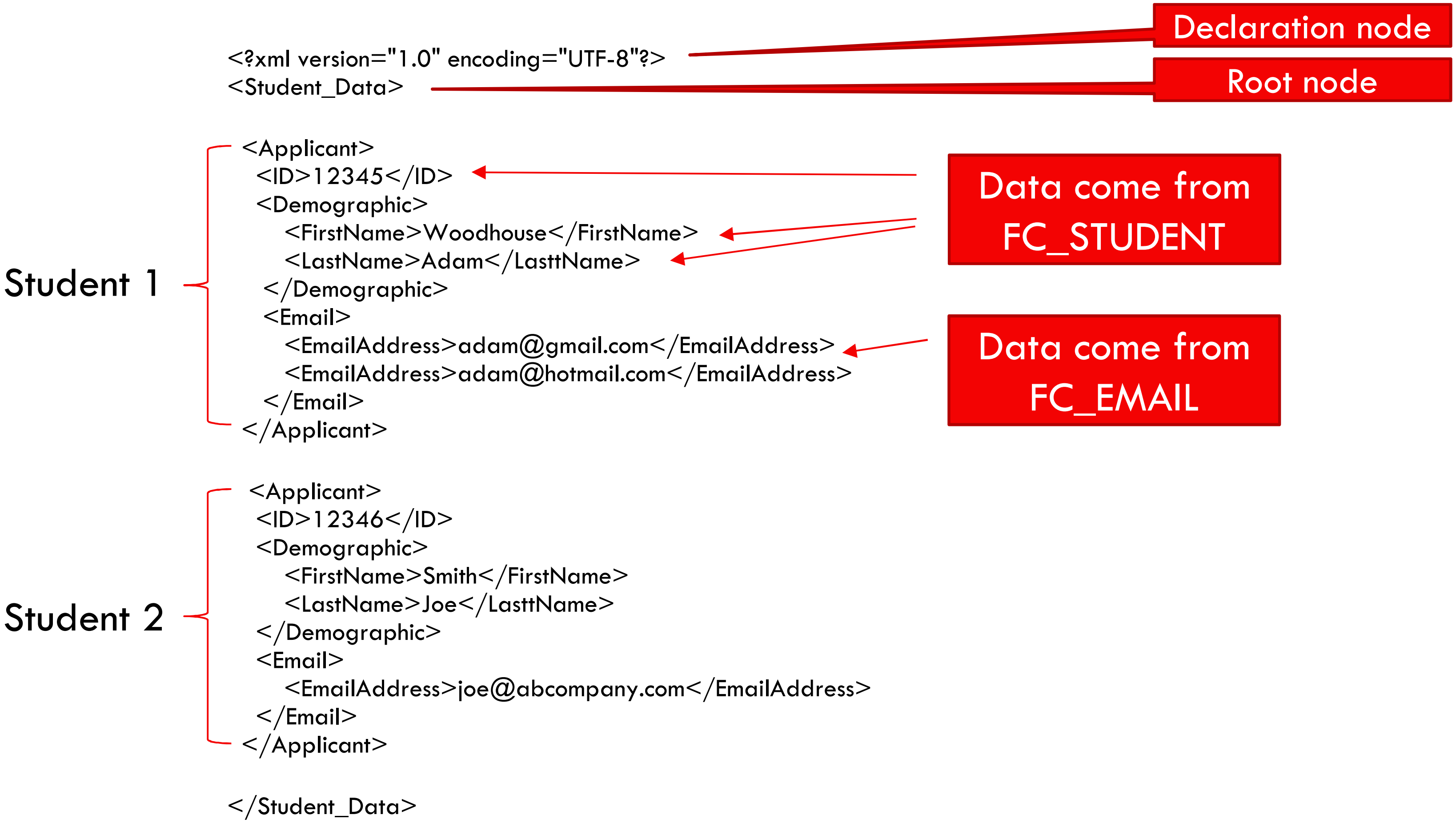
- FC_STUDENT

EMPLID	LAST_NAME	FIRST_NAME
12345	Woodhouse	Adam
12346	Smith	Joe

- FC_EMAIL

EMPLID	EMAIL_ADDR
12345	adam@gmail.com
12345	adam@hotmail.com
12346	joe@abcompany.copm

XML FILE WE ARE GOING TO CREATE



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

 <Applicant>
   <ID>12345</ID>
   <Demographic>
      <FirstName>Woodhouse</FirstName>
      <LastName>Adam</LasttName>
   </Demographic>
   <Email>
      <EmailAddress>adam@gmail.com</EmailAddress>
      <EmailAddress>adam@hotmail.com</EmailAddress>
   </Email>
 </Applicant>

  <Applicant>
   <ID>12346</ID>
   <Demographic>
      <FirstName>Smith</FirstName>
      <LastName>Joe</LasttName>
   </Demographic>
   <Email>
      <EmailAddress>joe@abcompany.com</EmailAddress>
   </Email>
 </Applicant>

</Student_Data>
```

Student 1

Student 2

Declaration node

Root node

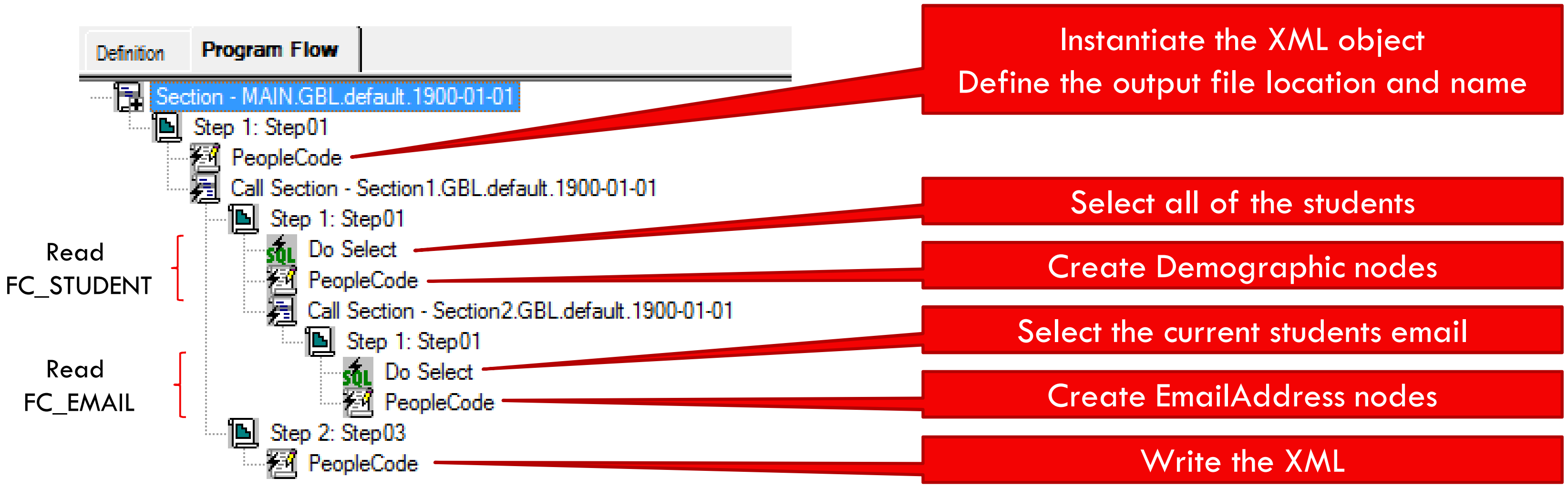
Data come from FC_STUDENT

Data come from FC_EMAIL

APP ENGINE – CREATE XML FILE

- In the previous example of importing a XML file I did everything in one App Engine Step
- In creating a XML file my App Engine has several Steps
- Several Steps are needed to use the App Engine Step "Do Select" SQL
- Each SQL Step will read all data needed for a related group of XML nodes
- The PeopleCode in the Step after the SQL will generate the XML nodes.

PEOPLECODE TO CREATE XML FILE



```
Section - MAIN.GBL.default.1900-01-01
  Step 1: Step01
    PeopleCode
    Call Section - Section1.GBL.default.1900-01-01
      Step 1: Step01
        Do Select
        PeopleCode
        Call Section - Section2.GBL.default.1900-01-01
          Step 1: Step01
            Do Select
            PeopleCode
      Step 2: Step03
        PeopleCode
```

Read FC_STUDENT

Read FC_EMAIL

Instantiate the XML object
Define the output file location and name

Select all of the students

Create Demographic nodes

Select the current students email

Create EmailAddress nodes

Write the XML

PEOPLECODE TO CREATE XML FILE

- The first Step to instantiate the XML file and define the root node

```
Global XmlDoc &outXMLDoc;
Global File &XMLFile;
Global XmlNode &Student_Node; /* NOTE: this node is referenced elsewhere, so must define. */

&filename = "/fileserver_path/output_file.xml"; /* Define the output file string */
&XMLFile = GetFile(&filename, "W", %FilePath_Absolute); /* Initialize the outbound file object */
&outXMLDoc = CreateXmlDoc(""); /* Instantiate a XML document */
MessageBox(0, "", 0, 0, "Created XML file: " | &filename); /* Add details to the job log */
&Student_Node = &outXMLDoc.CreateDocumentElement("Student_Data"); /* This is the root node */
```

XML FILE WE ARE GOING TO CREATE

The header declaration is automatically added

Student 1

Student 2

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

 <Applicant>
  <ID>12345</ID>
  <Demographic>
    <FirstName>Woodhouse</FirstName>
    <LastName>Adam</LasttName>
   </Demographic>
   <Email>
    <EmailAddress>adam@gmail.com</EmailAddress>
    <EmailAddress>adam@hotmail.com</EmailAddress>
   </Email>
 </Applicant>

  <Applicant>
  <ID>12346</ID>
  <Demographic>
    <FirstName>Smith</FirstName>
    <LastName>Joe</LasttName>
  </Demographic>
  <Email>
    <EmailAddress>joe@abcompany.com</EmailAddress>
  </Email>
 </Applicant>

</Student_Data>
```

PEOPLECODE TO CREATE XML FILE

- The second Step after the SQL is to create the Demographic node & start the Email nodes

```
/* Populate the XML file with the students Demographics details. Each student is its own Applicant node */

Global XmlDoc &outXMLDoc;
Global File &XMLFile;
Global XmlNode &Student_Node, &Email_Node; /* NOTE: nodes are referenced elsewhere, so must define. */

MessageBox(0, "", 0, 0, "Processing student: " | FC_STUDENT_AET.EMPLID); /* Add details to the job log */

&Applicant_Node = &Student_Node.AddElement("Applicant");
&ID_Node = &Applicant_Node.AddElement("ID");
&textNode = &ID_Node.AddText(FC_STUDENT_AET.EMPLID); /* FC_STUDENT_AET is populated in the SQL step */

&Demographic_Node = &Applicant_Node.AddElement("Demographic");  /* Parent */
&FirstName_Node = &Demographic_Node.AddElement("FirstName");  /* Child */
&textNode = &FirstName_Node.AddText(FC_STUDENT_AET.FIRST_NAME);
&LastName_Node = &Demographic_Node.AddElement("LastName");  /* Child */
&textNode = &LastName_Node.AddText(FC_STUDENT_AET.LAST_NAME);

/* Create the Email parent node */

&Email_Node = &Applicant_Node.AddElement("Email"); /* Parent */
```

XML FILE WE ARE GOING TO CREATE

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

Student 1

```
 <Applicant>
  <ID>12345</ID>
  <Demographic>
    <FirstName>Woodhouse</FirstName>
    <LastName>Adam</LasttName>
   </Demographic>
   <Email>
    <EmailAddress>adam@gmail.com</EmailAddress>
    <EmailAddress>adam@hotmail.com</EmailAddress>
   </Email>
 </Applicant>
```

Student 2

```
  <Applicant>
  <ID>12346</ID>
  <Demographic>
    <FirstName>Smith</FirstName>
    <LastName>Joe</LasttName>
  </Demographic>
  <Email>
    <EmailAddress>joe@abcompany.com</EmailAddress>
  </Email>
 </Applicant>

</Student_Data>
```

PEOPLECODE TO CREATE XML FILE

- The third Step after the read e-mail SQL is to create the Email node

```
/* Populate the XML file with the students email details (the <EmailAddress> nodes). */

Global XmlDoc &outXMLDoc;
Global File &XMLFile;
Global XmlNode &Email_Node;

&EmailAddress_Node = &Email_Node.AddElement("EmailAddress"); /* Child */

&textNode = &EmailAddress_Node.AddText(FC_EMAIL_AET.EMAIL_ADDR);
```

XML FILE WE ARE GOING TO CREATE

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

Student 1

```
 <Applicant>
  <ID>12345</ID>
  <Demographic>
    <FirstName>Woodhouse</FirstName>
    <LastName>Adam</LasttName>
   </Demographic>
   <Email>
    <EmailAddress>adam@gmail.com</EmailAddress>
    <EmailAddress>adam@hotmail.com</EmailAddress>
   </Email>
 </Applicant>
```

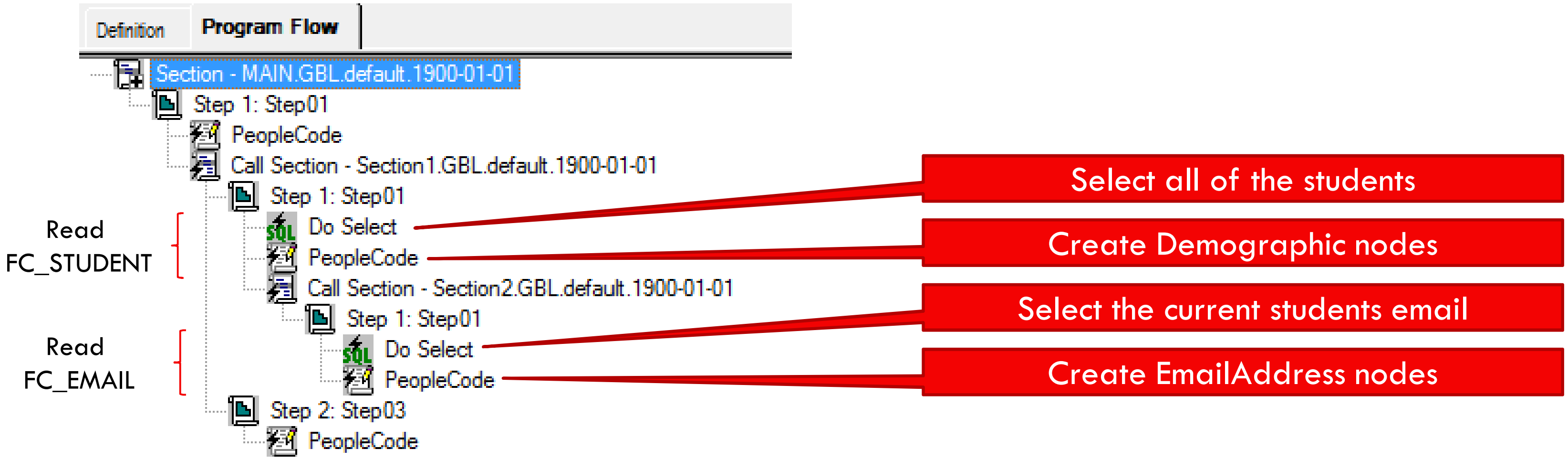
Student 2

```
  <Applicant>
  <ID>12346</ID>
  <Demographic>
    <FirstName>Smith</FirstName>
    <LastName>Joe</LasttName>
  </Demographic>
  <Email>
    <EmailAddress>joe@abcompany.com</EmailAddress>
  </Email>
 </Applicant>

</Student_Data>
```

PEOPLECODE TO CREATE XML FILE

- Once the <Applicant> node is built, the App Engine continues to loop through all rows found in Step 1: Step01 "Do Select"



```
Section - MAIN.GBL.default.1900-01-01
  Step 1: Step01
    PeopleCode
    Call Section - Section1.GBL.default.1900-01-01
      Step 1: Step01
        Do Select
        PeopleCode
        Call Section - Section2.GBL.default.1900-01-01
          Step 1: Step01
            Do Select
            PeopleCode
      Step 2: Step03
        PeopleCode
```

Read FC_STUDENT

Read FC_EMAIL

Select all of the students

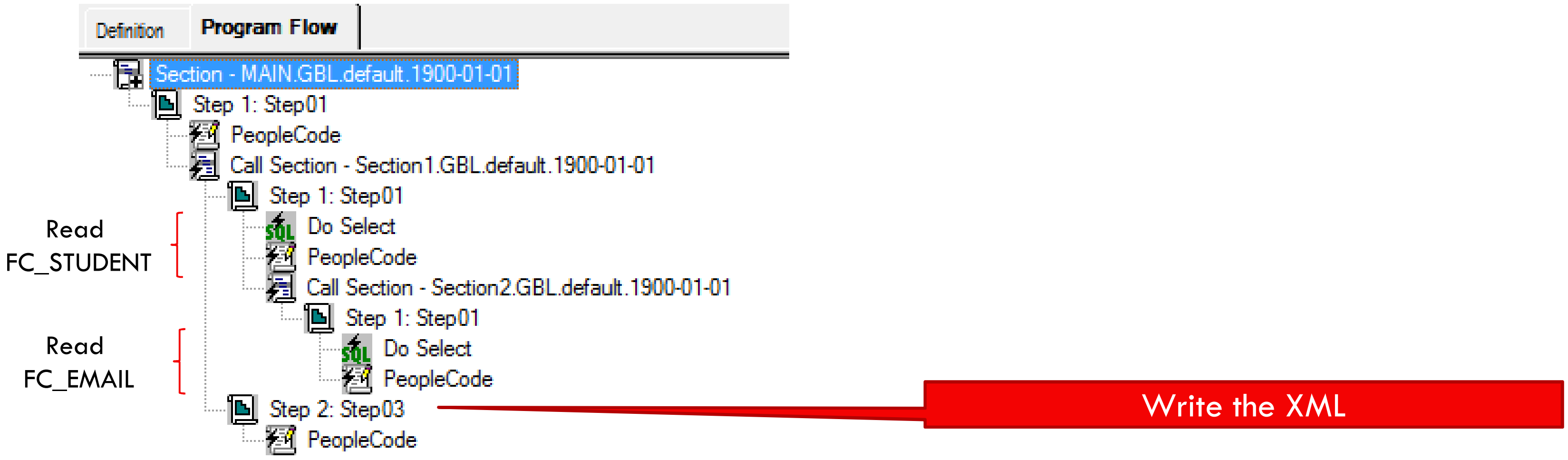
Create Demographic nodes

Select the current students email

Create EmailAddress nodes

PEOPLECODE TO CREATE XML FILE

- When all rows have been processed Step 2: Step03 is executed which writes the XML file.



Definition | Program Flow

```
Section - MAIN.GBL.default.1900-01-01
  Step 1: Step01
    PeopleCode
    Call Section - Section1.GBL.default.1900-01-01
      Step 1: Step01
        Do Select
        PeopleCode
        Call Section - Section2.GBL.default.1900-01-01
          Step 1: Step01
            Do Select
            PeopleCode
      Step 2: Step03
        PeopleCode
```

Read FC_STUDENT

Read FC_EMAIL

Write the XML

PEOPLECODE TO CREATE XML FILE

- The last Step is to finalize the XML and write the file

```
/* Format the XML content, replace the declaration tag and write to network share */

Global XmlDoc &outXMLDoc;
Global File &XMLFile;

&XMLString = &outXMLDoc.GenFormattedXmlString(); /* Format the content into structured XML */

/* Search the XML string and replace the declaration tag with new text */

&XMLString = Substitute(&XMLString, "<?xml version=""1.0""?>", "<?xml version=""1.0"" encoding=""UTF-
8""?>");  /* A great technique if we need to customize the header declaration tag */

&XMLFile.WriteLine(&XMLString); /* Populate & write the XML object with the formatted XML content */

&XMLFile.Close();
```

XML FILE WE CREATED

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

Student 1

```
 <Applicant>
  <ID>12345</ID>
  <Demographic>
    <FirstName>Woodhouse</FirstName>
    <LastName>Adam</LasttName>
   </Demographic>
   <Email>
    <EmailAddress>adam@gmail.com</EmailAddress>
    <EmailAddress>adam@hotmail.com</EmailAddress>
   </Email>
 </Applicant>
```

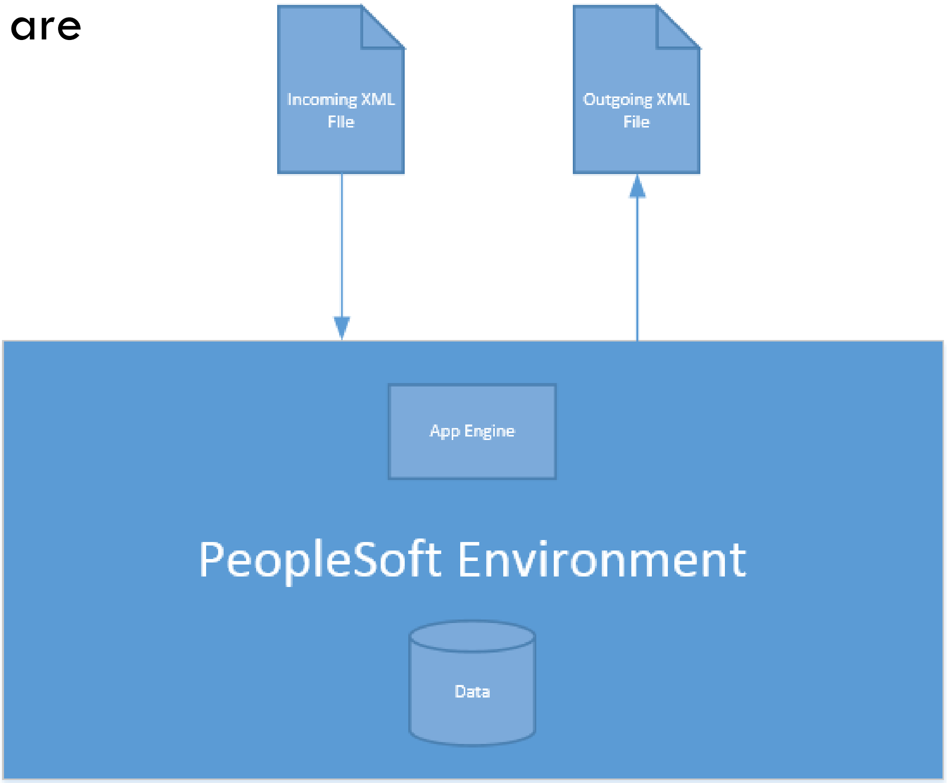
Student 2

```
  <Applicant>
  <ID>12346</ID>
  <Demographic>
    <FirstName>Smith</FirstName>
    <LastName>Joe</LasttName>
  </Demographic>
  <Email>
    <EmailAddress>joe@abcompany.com</EmailAddress>
  </Email>
 </Applicant>
```

```
</Student_Data>
```

SUMMARY

- We talked about how XML files are structured and the benefits
- We reviewed an example on how to import a file using PeopleSoft's App Engine
- We reviewed an example on how to export a file using PeopleSoft's App Engine



THANK YOU!

PRESENTER

Adam Woodhouse

Programmer/Analyst

Fleming College, Ontario

Website: www.flemingcollege.ca

E-mail: adam.woodhouse@flemingcollege.ca

ALL ALLIANCE PRESENTATIONS WILL BE AVAILABLE FOR DOWNLOAD FROM THE CONFERENCE SITE