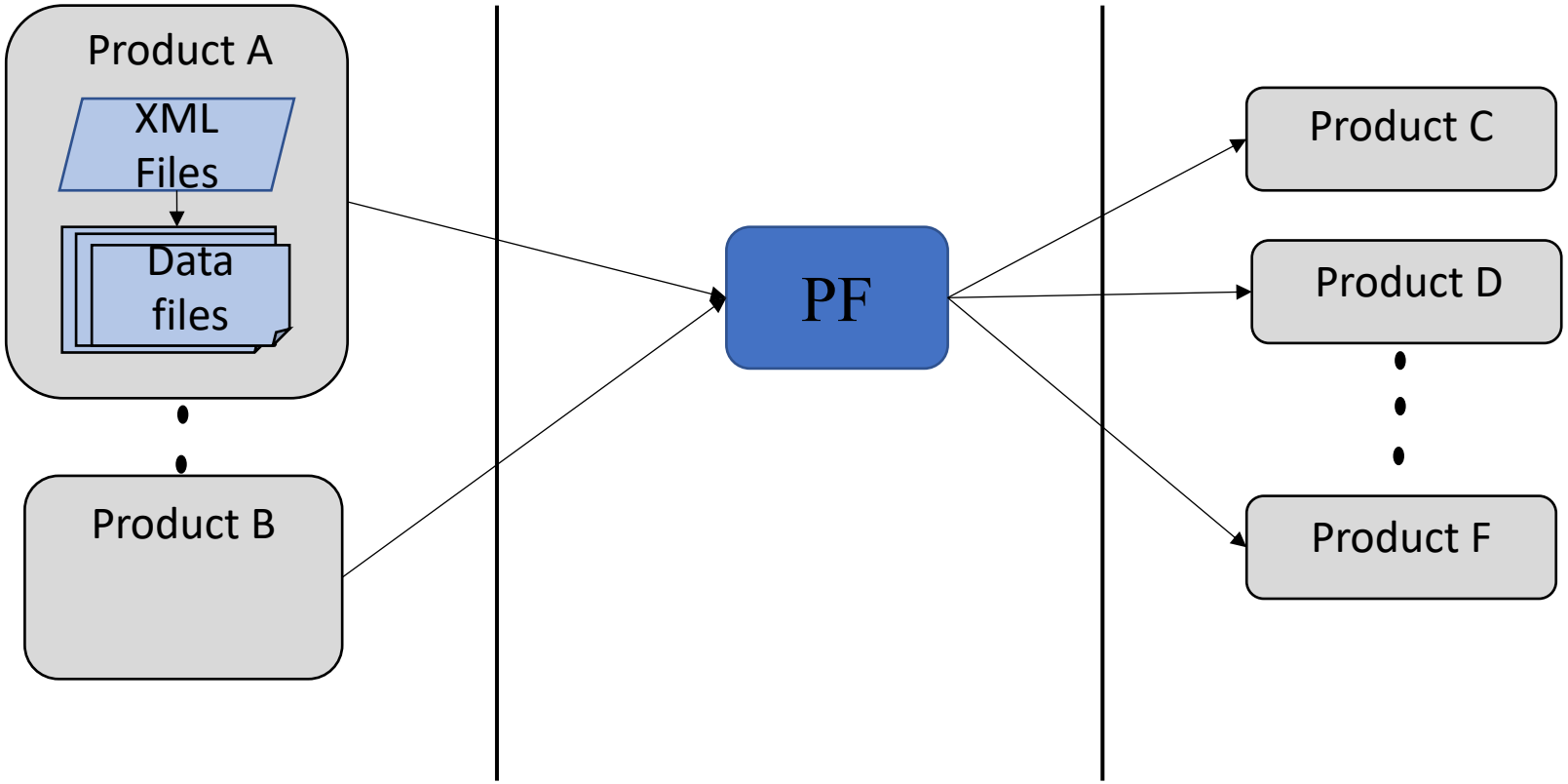
Data Model Bindings

Vanshika Kansal

OU-LE3/SDC-FR

Data model Product file

- XML Files: have four sections
  1. Generic Header: Common to all products
  2. Data: This section can contain pointers to files and related to product
  3. Quality Flags
  4. Parameters



Example xml file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<p:DpdTwoDMassClusterCatalog xmlns:p="http://euclid.esa.org/schema/dpd/le3/wl/twodmass/inp/clustercatalogs"
  xmlns:p1="http://euclid.esa.org/schema/sys"
  xmlns:p10="http://euclid.esa.org/schema/bas/cot"
  xmlns:p11="http://euclid.esa.org/schema/bas/imp"
  xmlns:p12="http://euclid.esa.org/schema/bas/imp/stc"
  xmlns:p13="http://euclid.esa.org/schema/pro/le3/wl"
  xmlns:p14="http://euclid.esa.org/schema/bas/dqc"
  xmlns:p15="http://euclid.esa.org/schema/bas/ppr"
  xmlns:p2="http://euclid.esa.org/schema/pro/le3/wl/twodmass"
  xmlns:p3="http://euclid.esa.org/schema/sys/dss"
  xmlns:p4="http://euclid.esa.org/schema/bas/fit"
  xmlns:p5="http://euclid.esa.org/schema/bas/imp/fits"
  xmlns:p6="http://euclid.esa.org/schema/bas/dtd"
  xmlns:p7="http://euclid.esa.org/schema/bas/utd"
  xmlns:p8="http://euclid.esa.org/schema/bas/cat"
  xmlns:p9="http://euclid.esa.org/schema/ins"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://euclid.esa.org/schema/dpd/le3/wl/twodmass/inp/clustercatalogs euc-test-le3-wl-twodmass-ClusterCatalog.xsd
  <Header>...
  <Data>
    <SpatialCoverage>
      <Polygon>
        <Vertex>
          <Position>
            <C1>0.0</C1>
            <C2>0.0</C2>
          </Position>
        </Vertex>
      </Polygon>
    </SpatialCoverage>
    <CatalogDescription>
      <CatalogOrigin>MEASURED_WIDE</CatalogOrigin>
      <CatalogType>PROXY</CatalogType>
      <PathToCatalogFile>PathToCatalogFile</PathToCatalogFile>
      <CatalogName>Ext-Des-Single-Epoch-Catalog</CatalogName>
    </CatalogDescription>
    <ClusterCatalog format="le3.wl.2dmass.input.clustercatalog" version="0.1">
      <DataContainer filestatus="PROPOSED">
        <FileName>FileName</FileName>
      </DataContainer>
    </ClusterCatalog>
  </Data>
</p:DpdTwoDMassClusterCatalog>
```

Meta Data

File Pointer

WHOHOO

Let’s start with Bindings

Setup for Exercise:

1. git clone https://gitlab.euclid-sgs.uk/CT-DEVWS/DevWS7/DmProject.git
2. git checkout develop
3. Follow the tutorial

To get the solution, go to the solution branch using command:

git checkout solution

Exercise 1:

Add Data Model Bindings as a project dependency in your project.

How to generate bindings locally

1. Compile the Data model locally (link: https://gitlab.euclid-sgs.uk/ST-DM/ST_DataModel).

2. Compile ST_DataModelBindings as well (link: https://gitlab.euclid-sgs.uk/ST-DM/ST_DataModelBindings).

3. The headers are then in InstallArea/x86_64-co7-gcc48-o2g/include and you can use them for binding

Solution:

STEP 1: Add “ST_DataModelBindings 8.0.3” into your top project CMakeList.txt file.

File (DmProject/CMakeList.txt):

- elements_project(DmProject 1.0 USE Elements 5.10.0 ST_DataModelBindings 8.0.3)

Solution:

STEP 2: Add subdirectories ST_DataModelBindings as your module dependencies and Link ST_DataModelBindings Libraries in your module CMaleList.txt

File (DmProject/DmModule/CMakeList.txt):

- elements_depends_on_subdirs(ST_DataModelBindings)
- elements_add_library(DmModule src/lib/*.cpp
  INCLUDE_DIRS ElementsKernel boost Cfitsio
  LINK_LIBRARIES ElementsKernel ST_DataModelBindings
  PUBLIC_HEADERS DmModule)

Data Model Bindings

Use of bindings in C++ and Python

Reading Input Products

Include header file / Import Python Package and module name

XML Schema: ../dpd/le3/wl/twodmass/inp/euc-test-le3-wl-twodmass-LensMCCatalog.xsd

C++	Python
#include "ST_DataModelBindings/dpd/le3/wl/twodmass/inp/euc-test-le3-wl-twodmass-LensMCCatalog.h"	import ST_DataModelBindings.dpd.le3.wl.twodmass.inp.raw.lensmccatalog_stub as lensCat

NOTE: BE CAREFUL WITH NESTED NAMESPACE (dpd::le3::wl::twodmass::inp::lensmccatalog)

Parse Input XML file

XML Top level element :DpdTwoDMassLensMCCatalog

C++	Python
auto in_xml = dpd::le3::wl::twodmass::inp::lensmccatalog::DpdTwoDMassLensMCCatalog (in_xml_filename.string(), xml_schema::flags::dont_validate);	with open(in_xml_filename, "r") as f: xml_string = f.read() in_xml = lensCat.CreateFromDocument(xml_string)

The function can parse from a file. It returns an auto_ptr to the object representing the element.

Tip: The flag is optional and is used to enable parsing of the files without access to the Data Model XSD files

Exercise:

Par*se the input XML file into an object*

- **Step 1:** Open the file **src/lib/DmInput.cpp & DmProject/DmInput.h**
- **Step 2:** Add the required **include** statements for the **input (DpdTwoDMassLensMCCatalog) product**
- **Step 3:** Parse the **input_xml_filename** XML file in a binding class
- **Step 4:** Log a message with the **type** of the object

Use of Element Object

XML Element Value:	<Data> <ShearCatalog format="le3.wl.2dmass.input.lensmccatalog" version="0.1"> <DataContainer filestatus="PROPOSED">\ <FileName>FileName</FileName> </DataContainer> </ShearCatalog> </Data>
C++	std::string value = in_xml->Data().ShearCatalog().DataContainer().FileName();
Python	value = in_xml.Data.ShearCatalog.DataContainer.FileName

Exercise:

Read Filename from the XML

Reading optional element

XML Optional element (minOccur = ‘0’)	<DenoiseParams> <DenoisingAlgo>GaussFilter</DenoisingAlgo> <GaussSTD>0.0</GaussSTD> <ThresholdFDR>0.0</ThresholdFDR> <!--This can be missing -- > </DenoiseParams>

C++	// The present() method - check if the element exists in the XML
	// The get() method - to retrieve the element.
	if (Param_xml->Data().DenoiseParams().ThresholdFDR().present()) {
	m_thresholdFDR = Param_xml->Data().DenoiseParams().ThresholdFDR().get();
	}

Python	No Special class If the element is not present, the member is set to None

Reading optional element

- XML Optional element (maxOccurs > 1):

```
<PatchParams>
  <Project>TAN</Project>
  <PatchList>
    <ProjCtr>
      <Longitude>0.0</Longitude>
      <Latitude>0.0</Latitude>
      <Frame>ICRS</Frame>
    </ProjCtr>
    <PatchWidth>0.0</PatchWidth>
    <PixelSize>0.0</PixelSize>
  </PatchList>
  <NPatches>0</NPatches>
</PatchParams>
```

Reading optional element

- **C++ : Special collection-like "sequence" class**

Class type alias for easy access:

```
pro::le3::wl::twodmass::twoDMassParamsConvergencePatch::PatchParams_sequence& pp  (Param_xml->Data().PatchParams());
```

Provides iterator functionality:

```
for (pro::le3::wl::twodmass::twoDMassParamsConvergencePatch::PatchParams_iterator i (pp.begin()); i!=pp.end(); ++i)
```

Reading optional element

- **Python: Represented as a list of objects**

```
for redshift_bin in Param_xml.Data.RedshiftBins.RedshiftBin:
    logger.info('zMax : ' + str(redshift_bin.ZMax))
```

Exercise:

Par*se the input Parameter XML file into an object and log the parameters*

Open the file **src/lib/Parameters.cpp & DmProject/Parameters.h** and start working.

Writing Output Products

Example: Output XML File

```xml
<Header>
  <ProductId>ProductId</ProductId>
  <ProductType>ProductType</ProductType>
  <SoftwareName>SoftwareName</SoftwareName>
  <SoftwareRelease>SoftwareRelease</SoftwareRelease>
  <EuclidPipelineSoftwareRelease>EuclidPipelineSoftwareRelease</EuclidPipelineSoftwareRelease>
  <ProdSDC>ProdSDC</ProdSDC>
  <DataSetRelease>DataSetRelease</DataSetRelease>
  <Purpose>DATA_RELEASE</Purpose>
  <PlanId>PlanId</PlanId>
  <PPOId>PPOId</PPOId>
  <PipelineDefinitionId>PipelineDefinitionId</PipelineDefinitionId>
  <PipelineRun>PipelineRun</PipelineRun>
  <ExitStatusCode>ExitStatusCode</ExitStatusCode>
  <ManualValidationStatus>VALID</ManualValidationStatus>
  <ExpirationDate>2001-12-31T12:00:00</ExpirationDate>
  <ToBePublished>true</ToBePublished>
  <Published>true</Published>
  <Curator>Curator</Curator>
  <CreationDate>2001-12-31T12:00:00</CreationDate>
</Header>
<Data>
  <NResamples>0</NResamples>
  <NoisyConvergence format="le3.wl.2dmass.output.patchconvergence" version="0.1">
    <DataContainer filestatus="PROPOSED">
      <FileName>FileName.fits</FileName>
    </DataContainer>
  </NoisyConvergence>
</Data>
```

Generic Header

Data: Simple type xs:int

Optional File Description with DataContainer

C++: Class constructor & arguments

- Mandatory sub-elements as parameters in correct order
- Implicit conversions to simple types are performed
- The parameters are copied
- Example:
  pro::le3::wl::twopcf::twoPCFWLShearShear2D data (output);
  The output object must be created in advance and the data object will keep a copy of it.
- **It requires bottom-up construction**

Python: Factory function with named parameters

- import ST_DataModelBindings.pro.le3.wl.raw.twopcf_stub as pro_devws
- data = pro_devws.twoPCFWLShearShear2D.Factory(twoPCFWLShearShear2DFile = output)
- You have to know the type of the class
- All sub-elements not passed as parameters are set to None
- Elements can be set manually after the creation of the object:
  data = pro_devws.twoPCFWLShearShear2D.Factory()
  data.twoPCFWLShearShear2DFile = output

Python: Factory function with named parameters

- **Warning**: If not all mandatory elements are set, the object cannot be serialized!
- The output object must be created in advance, but it can be an empty object created with its factory
- **Both bottom-up and top-down construction is possible Tip:** If you don't know the type of an element you can use the pyxb.BIND() method:
  data.twoPCFWLShearShear2DFile = pyxb.BIND(DataContainer=output_dataContainer, format=' le3.wl.2pcf.output.shearshear2d', version='0.1')

Exercise: Create Data Container / file pointer

```
<DataContainer filestatus="PROPOSED">
  <FileName>FileName</FileName>
</DataContainer>
```

- **Step 1:** Open the file **src/lib/DmOutput.cpp**
- **Step 2:** Create the **output** variable as a data container pointing to the **fits_out_filename**
- **Tip 1:** Check the type dataContainer in sys/dss
- **Tip 2:** The Filename element contains only the filename and no path
- **Tip 3:** The PF code should always set the filestatus to "**PROPOSED**"

Exercise: Create Output XML product

- **Step 1:** Create the **OutputProduct** element
- **Step 2:** Create the **Data** element
- **Step 3:** Create the product XML **root element**
- **Step 4:** Create the file **out_xml_filename** with the XML representing the product

References

- C++

https://www.codesynthesis.com/projects/xsd/documentation/cxx/tree/manual/

- Python

http://pyxb.sourceforge.net


WE DID IT