CDA Import Module Prototype

The source code for this prototype module is currently located on GitHub within the feature_fw branch: https://github.com/jembi/openmrs-module-shr-cdahandler.git . This module is in early stages and there are still issues with it, it is intended to be an experiment and requires further work before being put "in the field".

Overall Design

The design of the CDA Content Handler Module prototype is based on a series of “processors” which are instantiated based on the template id that the structure currently in “scope” of processing. Processors implement the Processor interface which itself is specialized into document, section and entry processor interfaces.

Each processor is responsible for interpreting/converting the HL7v3 CDA RMIM structure, and performing any updates the OpenMRS data store. A processor is also responsible for validating that it can parse the structure provided to it (not only CDA conformance but conformance of the constructed OpenMRS objects) and throwing a DocumentValidationException if this validation fails.

Processors are constructed using one of three processor factories:

- Document Processor Factory
- Section Processor Factory
- Entry Processor Factory

Each processor factor calls the ClassPathScannerUtil to construct the most appropriate processor for the particular structure. The utility uses the @ProcessTemplates annotation to determine which templateId a particular processor can handle. For example:

```
@ProcessTemplates( templateIds = { “x.x.x.x.x.x.x” })
public final class FooSectionProcessor implements SectionProcessor
```

Would indicate that any structure carrying “x.x.x.x.x.x” templateId should be processed by FooSectionProcessor. If a document/section/entry carries more than one templateId then the most specific implementation (that which is furthest down the inheritance tree) is selected. For example, consider the following processors:

```
@ProcessTemplates( templateIds = { “x.x.x.x.x.x.x” })
public class FooSectionProcessor implements SectionProcessor

@ProcessTemplates( templateIds = { “y.y.y.y.y.y.y” })
public final class BarSectionProcessor extends FooSectionProcessor
```

If a section has both OIDs (x.x.x.x.x.x and y.y.y.y.y.y) then BarSectionProcessor would be selected, even though FooSectionProcessor is able to process at least some of the constraints. Additionally processors carrying the @ProcessTemplates annotation may support more than one template identifier by adding additional OIDs to the templateIds annotation parameter.

All processors are responsible for cascading any values down the RMIM graph (for example: context conduction) and are responsible for calling processors for any sub-sections, entries, components, etc.

Any processor may obtain the context within which it is being executed via its getContext() method. This method returns the context within which a particular processor is being run, and does not represent the current node. For example, a SectionProcessor processing a section at the document level would have a getContext() value of DocumentProcessorContext (i.e. the document is the context not the section). Same applies to entries where the getContext() method would return the section which contains the entry.

Completed Document Processors

- Antepartum History and Physical (APHP) - AntepartumHistoryAndPhysicalDocumentProcessor
- Antepartum Summary (APS) - AntepartumSummaryDocumentProcessor
- MedcalDocumentsDocumentProcessor
- XDS Medical Summaries (XDS-MS) - MedicalSummariesDocumentProcessor
- History and Physical (HP) - HistoryAndPhysicalDocumentProcessor
- Generic Level 2 CDA - GenericDocumentProcessor

Completed Section Processors

- All Level 2 Sections - GenericLevel2SectionProcessor , including
  - Chief Complaint
  - Assessment and Plan
  - Consultations
  - History of Surgical Procedures
  - Intake and Output
  - Pain Assessment Panel
  - Birth Plan
  - Discharge Status
  - Event Outcomes
  - Newborn Status at Maternal Discharge
  - History of Blood Transfusion
  - Review of Systems
  - Care Plan
  - Discharge Disposition

    - Discharge Diet
    - Advance Directives
    - History of Present Illness
    - Hospital Course
    - Review of Systems
- Active Problems - ActiveProblemsSectionProcessor
- History of Infection - CodedHistoryOfInfectionSectionProcessor (Coded and non-coded)
- Pregnancy History - PregnancyHistorySectionProcessor
- Detailed Physical Examination - DetailedPhysicalExaminationSectionProcessor (Coded and non-coded)
- Allergies and Other Adverse Reactions - AllergiesAndOtherAdverseReactionsSectionProcessor
- Family History - FamilyHistorySectionProcessor (Coded and non-coded)
- Social History - SocialHistorySectionProcessor (Coded and non-coded)
- Vital Signs - VitalSignsSectionProcessor (Coded and non-coded)
- History of Past Illness - HistoryOfPastIllnessSectionProcessor (Coded and non-coded)
- Physical Exam - PhysicalExaminationSubSectionProcessor (handles all physical exam sub-sections)
- Antenatal Testing and Surveillance - AntenatalTestingAndSurveillanceSectionProcessor
- Antepartum Visit Summary Flowsheet - AntepartumVisitSummaryFlowsheetSectionProcessor
- Coded Results - CodedResultsSectionProcessor
- Estimated Delivery Dates - EstimatedDeliveryDatesSectionProcessor
- Medications - MedicationsSectionProcessor

Completed Entry Processors

- Vital Signs - VitalSignsObservationEntryProcessor
- Simple Observations - SimpleObservationEntryProcessor
- Family History Organizer - FamilyHistoryOrganizerEntryProcessor
- Concern Entry - ConcernEntryProcessor
- Allergies and Intolerances Concern - AllergiesAndIntolerancesConcernEntryProcessor
- Severity Observation - SeverityObservationEntryProcessor
- Allergies and Intolerances Observation - AllergiesAndIntolerancesEntryProcessor
- Problem Concern - ProblemConcernEntryProcessor
- Family History Observation - FamilyHistoryObservationEntryProcessor
- Vital Signs Organizer - VitalSignsOrganizerEntryProcessor
- Pregnancy Observation - PregnancyObservationEntryProcessor
- Pregnancy History Organizer - PregnancyHistoryOrganizerEntryProcessor
- Antenatal Testing and Surveillance Battery - AntenatalTestingAndSurveillanceBatteryEntryProcessor
- Estimated Delivery Date Observation - EstimatedDeliveryDateObservationEntryProcessor
- External References - ExternalReferencesEntryProcessor
- Medications - MedicationsEntryProcessor (Conditional and Split Dosing)
    - Tapered Dosing - TaperedDosingMedicationsEntryProcessor
    - Normal Dosing - NormalDosingMedicationsEntryProcessor
- Procedures - ProceduresEntryProcessor

Concept Dictionary

The OpenMRS concept dictionary is used extensively by this module. Each concept that requires association to an observation, allergy, problem, etc. is selected from a reference term within the code and concept source represented by the code/codeSystem attributes of a code in the CDA respectively. Once an appropriate reference term is found the concepts that are mapped to that term are searched based on suitability to store the CDA data. The type of concept used in OpenMRS' concept dictionary and its mapping to the CDA datatype to be stored is shown below:

- BL -> Boolean
- CS/CV/CE/CD -> Coded
- INT -> Numeric
- PQ -> Numeric (units are also checked)
- ST, II, TEL -> Text
- ED / SD -> Complex
- TS -> DateTime
- CO -> Numeric if "value" is used, Coded if "code" is used
- RTO, MO -> Text

If none of concept source, reference term, or concept are found then it is created and mapped accordingly. Where possible built in OpenMRS concepts are used. Additionally, whenever a Numeric concept is found to represent a PQ where the units do not match, the module will check to see if the units are convertible. For example, if an openMRS concept for Height is found however the PQ in the CDA is represented in m instead of cm the unit is converted to cm before storage.

Because CDA uses a variety of codes from SNOMED, LOINC and others, there is a need to bulk-import concepts and their mapping to MVP/CIEL upon module install. This is done via the ReferenceTermDictionary.xml file in the resources folder. Upon build, this XML file is transformed to a liquibase.xml file and placed in the omod file. This file contains mapping to CIEL concepts where appropriate and allows the module to use those mappings in an easy to maintain XML File (editable within Microsoft Excel as a table)

Orders

Many entries within the CDA document carry what are known as moodCodes. These mood codes identify the mode of the entry.

- Entries carrying a mood code of EVN (event) represent something did occur (i.e. I did observe X)

- Entries carrying a mood code of INT (intent) represent the intent to do what the entry represents (i.e. I intend to observe X, or I would like someone to observe X)
- Entries carrying a mood code of GOL (goal) mean the entry represents a desired end state such as in a care plan (i.e. The goal is to observe X by Y date)
- Entries carrying a mood code of PRP (proposal) mean the entry represents a proposal to perform something

Within the CDA import module, entries are imported into OpenMRS' data model depending on their mood code. Below is a table representing the source CDA RMIM class by moodCode and the resulting structure in OpenMRS.

RMIM Class / Mood	EVN	INT	GOL	PRP
Observation	Obs	ObservationOrder	Obs Group (having GOAL sub-obs)	Obs Group (having PROPOSAL sub-obs)
Organizer	Obs Group	N/A	N/A	N/A
SubstanceAdministration	Obs Group (160741)	DrugOrder	N/A	N/A
Procedure	Obs Group (160714)	ProcedureOrder	Obs Group (having GOAL sub-obs)	Obs Group (having PROPOSAL sub-obs)
Act	ActiveListItem (Allergy or Problem)	N/A	N/A	N/A

Note: ProcedureOrder and ObservationOrder are extended orders having additional data placed in the procedure_order and observation_order tables in OpenSHR's data model respectively. These classes were extended to qualify the type of order as well as track additional data required such as targetSiteCode, procedure/observation requested, and goal ranges.

Known Issues / Todo

Most up-to date listed here: https://github.com/jembi/openmrs-module-shr-cdahandler/issues

1. Currently there is some header elements which cannot be mapped appropriately into OpenMRS
2. Currently many types of entries are missing INT mood code handling
    a. This should create an Order in OpenMRS which has some caveats.
3. Family History observations are kind of hacked as the oMRS mechanism for storing these are similar to CDA but not identical.
4. Performance is poor, it appears to be in the code that double checks if a concept name already exists prior to creating it, I'm not sure if this is a problem with the standalone instance of oMRS that I'm testing with or if it is an issue with the logic.
5. All observations are created within the oMRS database and are seen in the Observation management admin panel, however the encounter summary panel does not show grouped obs in the UI.