**Overview**

The CCDAParser application is designed to parse CCDA (Consolidated Clinical Document Architecture) XML files and process their contents to extract and store relevant healthcare information. The application leverages several libraries, including Serilog for logging, Topshelf for hosting as a Windows service, and Dapper for database interaction.

**Dependencies**

* **Serilog**: Logging library used for structured logging.
* **Topshelf**: Library used to host the application as a Windows service.
* **Dapper**: Micro ORM for database operations.
* **System.Data.SqlClient**: ADO.NET provider for SQL Server.
* **System.Xml**: XML manipulation and parsing.
* **System.Xml.Linq**: LINQ to XML.
* **System.Xml.XPath**: Provides XPath query capabilities.

**Project Structure**

* CCDAParser.DAL: Data Access Layer for interacting with the database.
* CCDAParser.Service: Contains the service logic for starting and stopping the application as a service.
* CCDAParser.Parser: Contains parsing logic for different sections of the CCDA document.
* Program.cs: Entry point of the application.

**Program.cs**

The main entry point of the application, responsible for configuring logging, reading the XML file, and starting the service using Topshelf.

#### Key Components

1. **Logging Configuration**

Log.Logger = new LoggerConfiguration()

.WriteTo.Logger(lc => lc

.WriteTo.File("log.txt"))

.CreateLogger();

1. **File Reading**

string FileName = @"D:/CCmg DLLs/170.315\_b5\_toc\_amb\_rn\_r21\_sample1\_v12.xml";

string ccdaFile = await File.ReadAllTextAsync(FileName);

1. **Data Parsing**

CCDAParserDAL parserDAL = new CCDAParserDAL();

int PatientId = 0;

string con = "";

await parserDAL.XmlDataParhing(PatientId, FileName, ccdaFile, con);

1. **Service Hosting with Topshelf**

var exitCode = HostFactory.Run(x =>

{

x.Service<CCDAParserService>(s =>

{

s.ConstructUsing(CCDAParserService => new CCDAParserService());

s.WhenStarted(CCDAParserService => CCDAParserService.start());

s.WhenStopped(CCDAParserService => CCDAParserService.stop());

});

x.RunAsLocalService();

x.SetServiceName("CCDAParserService");

x.SetDisplayName("CCDA Parser Service");

x.SetDescription("CCDA Parser Background Job Service");

});

int exitCodeValue = (int)Convert.ChangeType(exitCode, exitCode.GetTypeCode());

Environment.ExitCode = exitCodeValue;

### CCDAParserDAL

This class is responsible for parsing the XML data and interacting with the database to store the parsed information.

#### Key Methods

* **XmlDataParhing**

public async Task<int> XmlDataParhing(int PatientId, string FileName, string ccdaFile, string con)

{

IDbConnection db = new SqlConnection(con);

string FacilityCode = "KPCA\_FC";

int orgID = 3;

string FullName = ccdaFile;

int \_result = 2;

try

{

\_result = 1;

// XML Document Loading and Parsing

XmlDocument document = new XmlDocument();

document.LoadXml(FullName);

string xmltext = document.InnerXml;

xmltext = xmltext.Replace("xmlns=\"urn:hl7-org:v3\"", "").Replace("xsi:type=\"CD\"", "");

XDocument doc = XDocument.Parse(xmltext);

var Demography = doc.XPathSelectElement("ClinicalDocument");

document.LoadXml(xmltext);

// Script Building for Database Insertion

StringBuilder scriptBuilder = new StringBuilder();

scriptBuilder.Append("BEGIN \nSET XACT\_ABORT ON \nBEGIN TRANSACTION \n");

// Extract Data and Populate Script

var encountersNode = Demography?.XPathSelectElement("componentOf");

if (encountersNode != null)

{

new Encounters().EncountersReading(encountersNode, scriptBuilder, \_ExternalMappingId, orgID, FileName, \_APUID);

}

var maingroup = doc.XPathSelectElement("ClinicalDocument/component/structuredBody")?.Elements().Where(w => w.Name.LocalName == "component");

if (maingroup != null)

{

foreach (var element in maingroup)

{

var nodes = element.XPathSelectElement("section");

if (nodes == null) continue;

var codeNode = nodes.Descendants("code").FirstOrDefault();

if (codeNode == null) continue;

var title = codeNode.Attribute("code")?.Value;

if (string.IsNullOrEmpty(title)) continue;

switch (title)

{

case "11450-4": //problems

new Problems().ProblemsReading(nodes, scriptBuilder, \_ExternalMappingId, FileName, \_APUID);

break;

case "10160-0": //medications

new Medications().MedicationsReading(nodes, scriptBuilder, \_ExternalMappingId, FileName, \_APUID);

break;

// Other cases for different sections...

default:

break;

}

}

}

scriptBuilder.Append("\nCOMMIT Transaction \nEND \n");

string finalScript = scriptBuilder.ToString();

// Execute the script

var parameter = new DynamicParameters();

var importPatientList = await db.QueryMultipleAsync(finalScript, null, null, null, CommandType.Text);

}

catch (Exception ex)

{

Log.Error(ex.Message, ex);

\_result = 0;

}

finally

{

db?.Dispose();

}

return \_result;

}

### CCDAParser.Parser.Problems

This class handles the parsing and processing of the "Problems" section in the CCDA document.

#### Key Methods

* **ProblemsReading**

public class Problems

{

string obspath = "act/entryRelationship/observation";

string obsPathAct = "act/";

public void ProblemsReading(XElement element, StringBuilder scriptBuilder, string externalMappingID, string FileName, string \_APUID)

{

var patientproblem = element.XPathSelectElements("entry");

int i = 1;

foreach (var s in patientproblem)

{

scriptBuilder.Append("\n");

scriptBuilder.Append("EXEC usp\_SaveUpdatePatientDiagnosis ");

scriptBuilder.Append("@ParamSectionName = '"); scriptBuilder.Append(element.XPathSelectElement("title").Value); scriptBuilder.Append("',");

scriptBuilder.Append("@ParamSectionChildName = '"); scriptBuilder.Append("section'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamDiagnosisDate = '"); scriptBuilder.Append(CommonLib.UpdateDateFormated(StartDates(s)));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamResolveDate = '"); scriptBuilder.Append(CommonLib.UpdateDateFormated(EndDates(s)));

scriptBuilder.Append("'");

var Confidential = element.GetElementValue("text/paragraph");

if (!string.IsNullOrEmpty(Confidential))

{

scriptBuilder.Append(","); scriptBuilder.Append("@ParamIsConfidential = 'true'");

}

else

{

scriptBuilder.Append(","); scriptBuilder.Append("@ParamIsConfidential = 'false'");

}

scriptBuilder.Append(","); scriptBuilder.Append("@ParamComment = '"); scriptBuilder.Append(getIcd(s).Replace("'", "''"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamStatus = '"); scriptBuilder.Append(s.GetAttributeValue($"{obsPathAct}/statusCode[@code]", "code"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamStatusCode = '"); scriptBuilder.Append(s.GetAttributeValue($"act/entryRelationship/observation/statusCode[@code]", "code"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@Paramicd10masterdiagnosisid = '"); scriptBuilder.Append(s.GetAttributeValue($"{obspath}/value/translation[@code]", "code"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamSnomedCode = '"); scriptBuilder.Append(SnomedCode(s));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamProblem = '"); scriptBuilder.Append(s.GetAttributeValue($"{obspath}/value/translation[@displayName]", "displayName").Replace("'", "''"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamSectionCode = '"); scriptBuilder.Append(element.GetAttributeValue("code[@code]", "code"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamSectionOId = '"); scriptBuilder.Append(element.GetAttributeValue("code[@codeSystem]", "codeSystem"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamSectionDisplayName = '"); scriptBuilder.Append(element.GetAttributeValue("code[@displayName]", "displayName"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamSectionCodeSystemName = '"); scriptBuilder.Append(element.GetAttributeValue("code[@codeSystemName]", "codeSystemName"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamSectionExtension = '"); scriptBuilder.Append(element.GetAttributeValue($"templateId[@extension]", "extension"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamSectionRoot = '"); scriptBuilder.Append(element.GetAttributeValue($"templateId[@root]", "root"));

scriptBuilder.Append("'");

scriptBuilder.Append(","); scriptBuilder.Append("@ParamAPUID = @APUID,");

scriptBuilder.Append("@ParamExternalMappingId = @ExternalMappingId,");

scriptBuilder.Append("@ParamPatientId = @PatientId,");

scriptBuilder.Append("@ParamFileName = @FileName");

scriptBuilder.Append(" ;\n");

i++;

}

}

private string SnomedCode(XElement s)

{

try

{

string DiagnosisStartDate = string.Empty;

var refvalue = s.GetAttributeValue("act/id[@root]", "root");

var Sdate = s.XPathSelectElement($"//tr[@ID='{refvalue}']");

DiagnosisStartDate = Sdate.GetElementValue("td[1]");

return DiagnosisStartDate;

}

catch (Exception ex)

{

return "";

}

}

private string StartDates(XElement s)

{

try

{

string DiagnosisStartDate = string.Empty;

var refvalue = s.GetAttributeValue("act/id[@root]", "root");

var Sdate = s.XPathSelectElement($"//tr[@ID='{refvalue}']");

DiagnosisStartDate = Sdate.GetElementValue("td[last()-1]");

return DiagnosisStartDate;

}

catch (Exception ex)

{

return "";

}

}

private string EndDates(XElement s)

{

try

{

string DiagnosisStartDate = string.Empty;

var refvalue = s.GetAttributeValue("act/id[@root]", "root");

var Sdate = s.XPathSelectElement($"//tr[@ID='{refvalue}']");

DiagnosisStartDate = Sdate.GetElementValue("td[last()]");

return DiagnosisStartDate;

}

catch (Exception ex)

{

return "";

}

}

private string getIcd(XElement s)

{

try

{

object obj = new object();

lock (obj)

{

string desc = "";

//var ent = s.XPathSelectElement($"{obspath}/value/translation[@codeSystemName='ICD-10']");

var ent = s.XPathSelectElement($"{obspath}/value/translation");

if (ent != null)

{

desc = ent.GetAttributeValue("displayName");

}

else

{

var entsnomed = s.XPathSelectElement($"{obspath}/value[@code]");

if (entsnomed != null)

{

var displayattr = entsnomed.Attribute("displayName");

if (displayattr != null)

{

desc = displayattr.Value;

}

}

else

{

desc = s.FindContentValue("ID", s.XPathSelectElement(obspath).GetReference());

}

}

return desc;

}

}

catch (Exception ex)

{

return "";

}

}

}

### Example Classes for Other Data Sections

* **Demography**

public class Demography

{

public void DemographyReading(XElement demography, StringBuilder scriptBuilder, string externalMappingId, string apuid, int orgID, string fileName, string facilityCode, string assigningAuthorityName, int patientId)

{

// Implementation for reading demography section

}

}

* **Encounters**

public class Encounters

{

public void EncountersReading(XElement encountersNode, StringBuilder scriptBuilder, string externalMappingId, int orgID, string fileName, string apuid)

{

// Implementation for reading encounters section

}

}

* **Medications**

public class Medications

{

public void MedicationsReading(XElement nodes, StringBuilder scriptBuilder, string externalMappingId, string fileName, string apuid)

{

// Implementation for reading medications section

}

}

### Conclusion

This documentation provides an overview of the CCDAParser application, its dependencies, key components, and detailed examples of how the data parsing logic is implemented. The application reads a CCDA XML file, processes its contents, and stores the extracted data in a database. The process is managed by a Windows service, with logging handled by Serilog and service management by Topshelf. The Problems class example illustrates the detailed parsing and database script generation for the "Problems" section of the CCDA document.