International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

# Claim Extraction Process Automation

**Praveen Kumar Vutukuri**

Claim Intake Systems in Cenetene, Centene Corporation, Tampa, FL, USA
Email: *praveen524svec[at]gmail.com*

**Abstract:** ***This abstract outline the implementation of an automated claim extraction process in the healthcare industry using advanced technologies such as Optical Character Recognition (OCR), Natural Language Processing (NLP), and machine learning. The aim is to enhance accuracy, speed, and efficiency in claim processing while reducing operational costs.***

**Keywords:** Queues, Message Handling, parallel processing, .NET Services, Resources, Monitoring, High Performance, Scalability, logging, HIPAA, XSLT, XML, Claim Automation

## 1. Introduction

The healthcare industry generates vast amounts of data through patient records, insurance claims, medical billing, and other documentation. Efficiently processing this data is critical for optimizing healthcare services, reducing administrative costs, and improving patient outcomes. Claim extraction is a vital component of this process, involving the identification and extraction of key information from insurance claims and medical documents. Manual claim processing is time-consuming, error-prone, and resource-intensive, necessitating the adoption of automated solutions.

As part of automation, we have a Business Unit called Claim Intake systems and CIS supposed to process every day at least million claims a day and provide the output to respective outbound message system.

CIS have the multiple applications to run the claim process. So entire process needs to be data to execute and data can be extracted by certain process and data is supposed to be the chunks. As part of chunks process, each chunk is designed the data for each module so all the unnecessary data need not to be passed for each modules and it will save the validation process for each module iteration.

## 2. Literature Review

The healthcare industry is a data-intensive sector where efficient data management is crucial for operational efficiency and patient care. Claim extraction automation has emerged as a significant area of interest, aiming to streamline the processing of insurance claims and medical documents. This literature review explores existing research and developments in the field of claim extraction automation, focusing on the technologies and methodologies employed, challenges encountered, and the impact on the healthcare industry.

***Optical Character Recognition (OCR)***
Tesseract is an open-source OCR engine widely used in various applications. Research has demonstrated its effectiveness in converting scanned documents into machine-readable text, though challenges remain in processing documents with complex layouts or poor image quality. Studies have highlighted the capabilities of commercial OCR tools such as Amazon Textract, Google Cloud Vision, and Azure Cognitive Services in extracting text from a wide range of document types. These tools often incorporate advanced image preprocessing techniques to enhance accuracy.

***Natural Language Processing (NLP)***
Named Entity Recognition (NER) is a fundamental NLP task used in claim extraction to identify and classify entities such as dates, medical terms, and monetary amounts. Research indicates that models like SpaCy and Stanford NLP offer robust NER capabilities, which are essential for accurate claim extraction .

***Supervised Learning***
Research has shown that supervised learning techniques, where models are trained on labeled datasets, are effective for claim extraction. Algorithms such as decision trees, support vector machines, and neural networks have been used to classify and extract relevant information from documents.

***Deep Learning Models***
These method approaches, particularly using convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have been applied to complex document analysis tasks. Studies indicate that these models can capture intricate patterns in data, leading to more accurate extraction of claims.

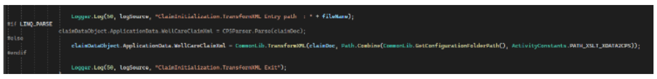
***Cloud-Based Solutions***
Cloud platforms like AWS, Google Cloud, and Azure offer scalable and flexible solutions for claim extraction automation. Research emphasizes the benefits of cloud computing in handling large volumes of data, enabling real-time processing, and integrating various services seamlessly**.**

***Data Quality and Standardization***
One of the primary challenges in claim extraction is the variability in document quality and formats. Research highlights the need for standardized document formats and enhanced data preprocessing techniques to improve extraction accuracy.

## 3. Extraction process automation implementation

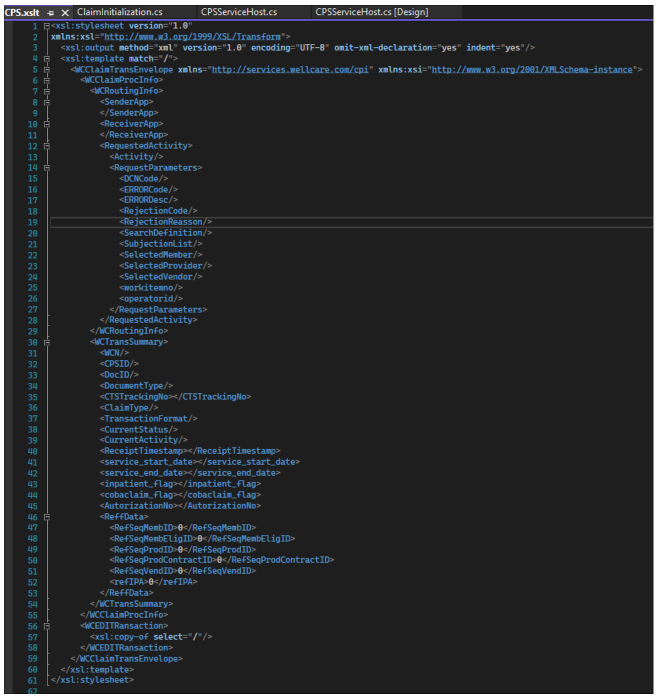
### 3.1 Transaction-537 to organization structure

- Using XSLT (Extensible Stylesheet Language Transformations) is a powerful language used for transforming XML documents into other formats, such as HTML, plain text, or other XML documents.



```
                Logger.Log(50, logSource, "ClaimInitialization.TransformXML Entry path  : " + fileName);
#if LINQ_PARSE
            claimDataObject.ApplicationData.WellCareClaimXml = CPSParser.Parse(claimDoc);
#else
                claimDataObject.ApplicationData.WellCareClaimXml = CommonLib.TransformXML(claimDoc, Path.Combine(CommonLib.GetConfigurationFolderPath(), ActivityConstants.PATH_XSLT_XDATA2CPS));
#endif

                Logger.Log(50, logSource, "ClaimInitialization.TransformXML Exit");
```

- As part of automation process first we convert the industry standard XDATA which was created the standards of HIPAA into WellcareClaimXml.

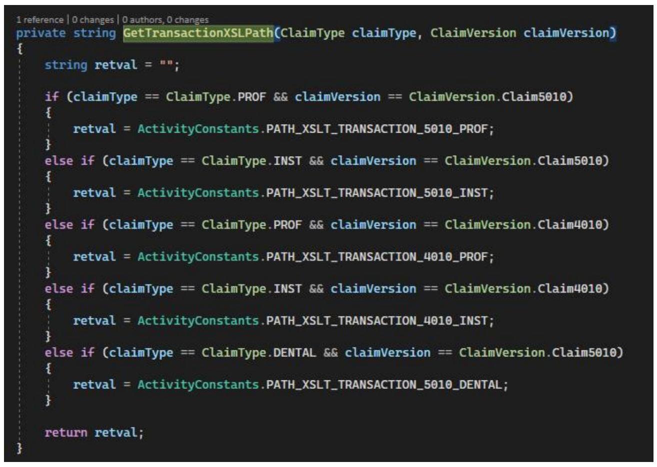


CPS.xslt | ClaimInitialization.cs | CPSServiceHost.cs | CPSServiceHost.cs [Design]

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" omit-xml-declaration="yes" indent="yes"/>
  <xsl:template match="/">
    <WCClaimTransEnvelope xmlns="http://services.wellcare.com/cpi" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <WCClaimProcInfo>
        <WCRoutingInfo>
          <SenderApp>
          </SenderApp>
          <ReceiverApp>
          </ReceiverApp>
          <RequestedActivity>
            <Activity/>
            <RequestParameters>
              <DCNCode/>
              <ERRORCode/>
              <ERRORDesc/>
              <RejectionCode/>
              <RejectionReasson/>
              <SearchDefinition/>
              <SubjectionList/>
              <SelectedMember/>
              <SelectedProvider/>
              <SelectedVendor/>
              <workitemno/>
              <operatorid/>
            </RequestParameters>
          </RequestedActivity>
        </WCRoutingInfo>
        <WCTransSummary>
          <WCN/>
          <CPSID/>
          <DocID/>
          <DocumentType/>
          <CTSTrackingNo></CTSTrackingNo>
          <ClaimType/>
          <TransactionFormat/>
          <CurrentStatus/>
          <CurrentActivity/>
          <ReceiptTimestamp></ReceiptTimestamp>
          <service_start_date></service_start_date>
          <service_end_date></service_end_date>
          <inpatient_flag></inpatient_flag>
          <cobaclaim_flag></cobaclaim_flag>
          <AutorizationNo></AutorizationNo>
          <ReffData>
            <RefSeqMembID>0</RefSeqMembID>
            <RefSeqMembEligID>0</RefSeqMembEligID>
            <RefSeqProdID>0</RefSeqProdID>
            <RefSeqProdContractID>0</RefSeqProdContractID>
            <RefSeqVendID>0</RefSeqVendID>
            <refIPA>0</refIPA>
          </ReffData>
        </WCTransSummary>
      </WCClaimProcInfo>
      <WCEDITRansaction>
        <xsl:copy-of select="/"/>
      </WCEDITRansaction>
    </WCClaimTransEnvelope>
  </xsl:template>
</xsl:stylesheet>
```

### 3.2 Create Transaction XML

- As we have the multiple types of claims, and each claim data is organized in different elements.
- So, based on the XSLT path we convert wellcareClaimXML into TransactionXML.



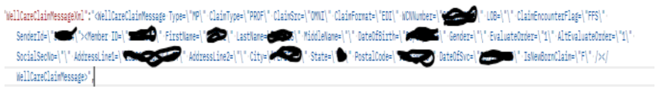
```
1 reference | 0 changes | 0 authors, 0 changes
private string GetTransactionXSLPath(ClaimType claimType, ClaimVersion claimVersion)
{
    string retval = "";

    if (claimType == ClaimType.PROF && claimVersion == ClaimVersion.Claim5010)
    {
        retval = ActivityConstants.PATH_XSLT_TRANSACTION_5010_PROF;
    }
    else if (claimType == ClaimType.INST && claimVersion == ClaimVersion.Claim5010)
    {
        retval = ActivityConstants.PATH_XSLT_TRANSACTION_5010_INST;
    }
    else if (claimType == ClaimType.PROF && claimVersion == ClaimVersion.Claim4010)
    {
        retval = ActivityConstants.PATH_XSLT_TRANSACTION_4010_PROF;
    }
    else if (claimType == ClaimType.INST && claimVersion == ClaimVersion.Claim4010)
    {
        retval = ActivityConstants.PATH_XSLT_TRANSACTION_4010_INST;
    }
    else if (claimType == ClaimType.DENTAL && claimVersion == ClaimVersion.Claim5010)
    {
        retval = ActivityConstants.PATH_XSLT_TRANSACTION_5010_DENTAL;
    }

    return retval;
}
```

- .NET Aspire components are specialized NuGet packages designed to streamline the development and management of cloud-native applications. These components simplify the integration and connection to various popular services and platforms, such as Redis, PostgreSQL, and others. They address common cloud-native concerns through standardized configuration patterns, making it easier for developers to build scalable and reliable applications.
- ***Simplified Service Integration***: .NET Aspire components are pre-configured to facilitate easy connections to popular services and platforms. This eliminates the need for developers to manually configure each service, reducing setup time and potential errors.
- ***Standardized Configuration Patterns:*** - Each component follows standardized configuration patterns, ensuring consistency across different services. These patterns include best practices for connecting to services, managing credentials, and configuring endpoints, which helps maintain uniformity in application configuration.
- ***Health Checks and Telemetry:*** - Health checks and telemetry are integral parts of .NET Aspire components. Health checks monitor the status and availability of services, enabling proactive management and quick detection of issues. Telemetry provides insights into application performance and usage, helping developers optimize their applications and troubleshoot problems effectively.
- ***Automatic Configuration Injection:*** - .NET Aspire components are designed to work seamlessly with the .NET Aspire orchestration model. Configurations for each component are automatically injected based on named resources. This means that when one service references another, the necessary configurations are inherited automatically, ensuring that services can communicate with each other without manual intervention.
- ***Compatibility with Orchestration:*** - These components are fully compatible with .NET Aspire's orchestration capabilities. The orchestration model manages the complex interconnections between various services and components, simplifying the setup and management of distributed applications.
- ***Enhancing Local Development Experience:*** - By providing a consistent setup pattern and abstracting low-level implementation details, .NET Aspire components enhance the local development experience. Developers can focus on building features rather than dealing with the intricacies of service configuration and integration.
- There were types of claims called 5010 Professional, 5010 Institutional, 4010 Professional and 4010 Institutional. Based on XDATA headers system will define whether the claim is Professional and Institutional.

### 3.3 Create Business Modules Data

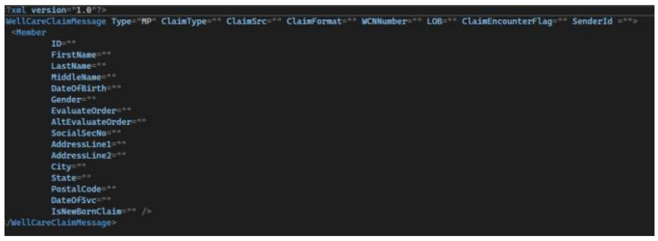
- As part of claim process system needs additional business information called member information, provider information and vendor information.
- To get the members information we have o use the business data and it can be executed through with given information from XData.
- For better accuracy, DB logic will be executed with respective packages and packages will execute with the customized model which is created from XData.

- ***Member XML Sample***



```
'WellCareClaimMessageXml':"<WellCareClaimMessage Type=\"MP\" ClaimType=\"PROF\" ClaimSrc=\"OMNI\" ClaimFormat=\"EDI\" WCNNumber=\"[redacted]\" LOB=\"\" ClaimEncounterFlag=\"FFS\"
   SenderId=\"[redacted]\"><Member ID=\"[redacted]\" FirstName=\"[redacted]\" LastName=\"[redacted]\" MiddleName=\"\" DateOfBirth=\"[redacted]\" Gender=\"\" EvaluateOrder=\"1\" AltEvaluateOrder=\"1\"
   SocialSecNo=\"\" AddressLine1=\"[redacted]\" AddressLine2=\"\" City=\"[redacted]\" State=\"[redacted]\" PostalCode=\"[redacted]\" DateOfSvc=\"[redacted]\" IsNewBornClaim=\"F\" /></
   WellCareClaimMessage>",
```

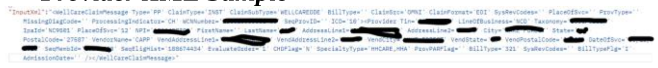
- ***Member XML Template***



```
<?xml version="1.0"?>
<WellCareClaimMessage Type="MP" ClaimType="" ClaimSrc="" ClaimFormat="" WCNNumber="" LOB="" ClaimEncounterFlag="" SenderId ="">
 <Member
        ID=""
        FirstName=""
        LastName=""
        MiddleName=""
        DateOfBirth=""
        Gender=""
        EvaluateOrder=""
        AltEvaluateOrder=""
        SocialSecNo=""
        AddressLine1=""
        AddressLine2=""
        City=""
        State=""
        PostalCode=""
        DateOfSvc=""
        IsNewBornClaim="" />
</WellCareClaimMessage>
```

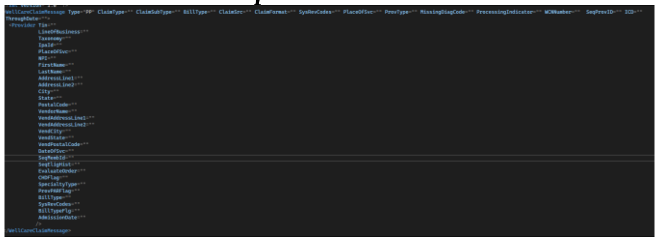
- Therefore, following the template format described above, the code logic dynamically constructs the XML object from XDATA and retrieves the member information.

- ***Provider XML Sample***



```
"InputXml":"<WellCareClaimMessage Type='PP' ClaimType='INST' ClaimSubType='WELLCAREDDE' BillType='' ClaimSrc='OMNI' ClaimFormat='EDI' SysRevCodes='' PlaceOfSvc='' ProvType=''
   MissingDiagCode='' ProcessingIndicator='CH' WCNNumber='[redacted] SeqProvID='' ICD='10'><Provider Tin='[redacted]' LineOfBusiness='NCD' Taxonomy=[redacted]
   IpaId='NC9881' PlaceOfSvc='12' NPI='[redacted]' FirstName='' LastName='[redacted]' AddressLine1=[redacted] AddressLine2=[redacted] City=[redacted]' State='[redacted]
   PostalCode='27587' VendorName='CAPP' VendAddressLine1=[redacted] VendAddressLine2=[redacted] VendCity=[redacted] VendState=[redacted] VendPostalCode=[redacted] DateOfSvc=[redacted]
   [redacted] SeqMembId=[redacted]' SeqEligHist='188674434' EvaluateOrder='1' CHDFlag='N' SpecialtyType='HHCARE,HHA' ProvPARFlag='' BillType='321' SysRevCodes='' BillTypeFlg='I'
   AdmissionDate='' /></WellCareClaimMessage>"
```

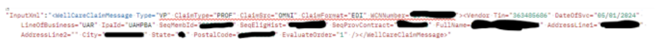
- ***Provider XML Template***



```
<?xml version="1.0"?>
<WellCareClaimMessage Type="PP" ClaimType="" ClaimSubType="" BillType="" ClaimSrc="" ClaimFormat="" SysRevCodes="" PlaceOfSvc="" ProvType="" MissingDiagCode="" ProcessingIndicator="" WCNNumber="" SeqProvID="" ICD=""
ThroughDate="">
 <Provider Tin=""
          LineOfBusiness=""
          Taxonomy=""
          IpaId=""
          PlaceOfSvc=""
          NPI=""
          FirstName=""
          LastName=""
          AddressLine1=""
          AddressLine2=""
          City=""
          State=""
          PostalCode=""
          VendorName=""
          VendAddressLine1=""
          VendAddressLine2=""
          VendCity=""
          VendState=""
          VendPostalCode=""
          DateOfSvc=""
          SeqMembId=""
          SeqEligHist=""
          EvaluateOrder=""
          CHDFlag=""
          SpecialtyType=""
          ProvPARFlag=""
          BillType=""
          SysRevCodes=""
          BillTypeFlg=""
          AdmissionDate=""
         />
</WellCareClaimMessage>
```

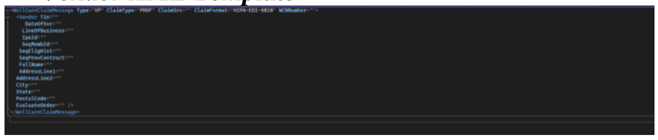
- Thus, utilizing the template format, code logic mentioned above, dynamically construct the XML object from XDATA and retrieve the provider information.

- ***Vendor XML Sample***



```
"InputXml":"<WellCareClaimMessage Type="VP" ClaimType="PROF" ClaimSrc="OMNI" ClaimFormat="EDI" WCNNumber=[redacted]"><Vendor Tin="363485686" DateOfSvc="05/01/2024"
   LineOfBusiness="UAR" IpaId="UAHPBA" SeqMembId="[redacted]" SeqEligHist=[redacted]" SeqProvContract="[redacted]" FullName=[redacted]" AddressLine1="[redacted]"
   AddressLine2="" City="[redacted]" State="[redacted]" PostalCode="[redacted]" EvaluateOrder="1" /></WellCareClaimMessage>"
```

- ***Vendor XML Template***



```
<WellCareClaimMessage Type="VP" ClaimType="PROF" ClaimSrc="" ClaimFormat="HIPA-EDI-5010" WCNNumber="">
  <Vendor Tin=""
     DateOfSvc=""
    LineOfBusiness=""
    IpaId=""
    SeqMembId=""
   SeqEligHist=""
   SeqProvContract=""
   FullName=""
   AddressLine1=""
  AddressLine2=""
  City=""
  State=""
  PostalCode=""
  EvaluateOrder="" />
</WellCareClaimMessage>
```

- Therefore, by employing the template format and the aforementioned code logic, dynamically generate the XML object from XDATA and fetch the vendor information.

## 4. Implementation Stratagies

### 4.1 XSLT is in automation process.

To implement the usage of XSLT in the claim automation process for processing XData, you must identify the need and follow the below implementation strategies.

***Identify XData Structure***: Understand the structure and content of the XData that needs to be processed in the claim automation process. This includes knowing what information is contained within the XData and how it is organized.

***Develop XSLT Stylesheet***: Create an XSLT stylesheet that defines the transformation rules for converting the XData into the desired output format. This involves defining templates, match patterns, and instructions for processing different elements and attributes of the XData.

***Integrate XSLT into Automation Workflow:*** Incorporate the XSLT transformation into the claim automation workflow. This might involve calling the XSLT processor from within your automation system or integrating it directly into the processing pipeline.

***Apply XSLT Transformation:*** Use the developed XSLT stylesheet to transform the XData as part of the claim automation process. This step typically involves applying the XSLT transformation to the input XData to generate the desired output, which could be in XML, HTML, text, or any other suitable format.

***Handle Output:*** Process the transformed output as required by the automation workflow. This could involve further processing, validation, storage, or transmission of the transformed data depending on the specific requirements of the claim automation process.

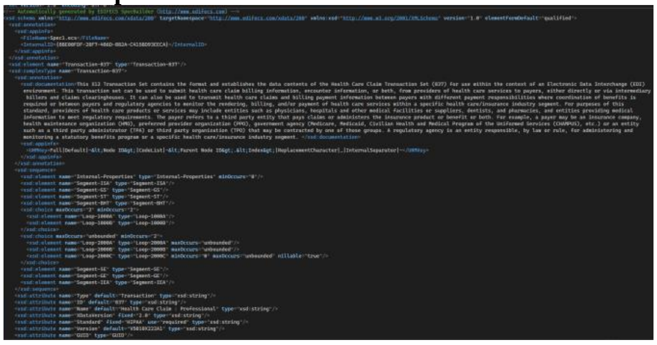
***Testing and Validation:*** Thoroughly test the XSLT transformation to ensure that it accurately converts the input XData into the desired output format. Validate the transformed output against expected results and refine the XSLT stylesheet as needed.

***Maintenance and Updates:*** Regularly review and maintain the XSLT stylesheet to accommodate any changes or updates to the input XData structure or the requirements of the claim automation process. This may involve modifying existing transformation rules or adding new ones as necessary.

### Understanding XSD

XSD (XML Schema Definition) in the claim automation process to create XSLT business logic to program the element structure and handle the business logic and read the values based on the need. Below is one of the XSD Sample to create based on Transaction-837.

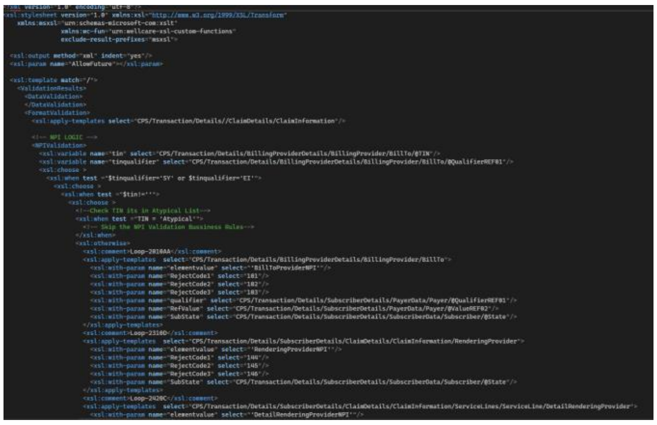
### XSD Samples



```
<?xml version="1.0" encoding="utf-8"?>
<!-- Automatically generated by EDIFECS SpecBuilder (http://www.edifecs.com) -->
<xsd:schema xmlns="http://www.edifecs.com/xdata/200" targetNamespace="http://www.edifecs.com/xdata/200" xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="1.0" elementFormDefault="qualified">
  <xsd:annotation>
    <xsd:appinfo>
      <FileName>Spec1.ecs</FileName>
      <InternalID>{8BE00FDF-28F7-486D-882A-C4158D93EECA}</InternalID>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:element name="Transaction-837" type="Transaction-837"/>
  <xsd:complexType name="Transaction-837">
    <xsd:annotation>
      <xsd:documentation>This X12 Transaction Set contains the format and establishes the data contents of the Health Care Claim Transaction Set (837) for use within the context of an Electronic Data Interchange (EDI)
      environment. This transaction set can be used to submit health care claim billing information, encounter information, or both, from providers of health care services to payers, either directly or via intermediary
       billers and claims clearinghouses. It can also be used to transmit health care claims and billing payment information between payers with different payment responsibilities where coordination of benefits is
      required or between payers and regulatory agencies to monitor the rendering, billing, and/or payment of health care services within a specific health care/insurance industry segment. For purposes of this
      standard, providers of health care products or services may include entities such as physicians, hospitals and other medical facilities or suppliers, dentists, and pharmacies, and entities providing medical
      information to meet regulatory requirements. The payer refers to a third party entity that pays claims or administers the insurance product or benefit or both. For example, a payer may be an insurance company,
      health maintenance organization (HMO), preferred provider organization (PPO), government agency (Medicare, Medicaid, Civilian Health and Medical Program of the Uniformed Services (CHAMPUS), etc.) or an entity
      such as a third party administrator (TPA) or third party organization (TPO) that may be contracted by one of those groups. A regulatory agency is an entity responsible, by law or rule, for administering and
      monitoring a statutory benefits program or a specific health care/insurance industry segment. </xsd:documentation>
      <xsd:appinfo>
        <URMKey>Full|Default|~&lt;Node ID&gt;|CodeList|~&lt;Parent Node ID&gt;.&lt;Index&gt;|ReplacementCharacter|_|InternalSeparator|~</URMKey>
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="Internal-Properties" type="Internal-Properties" minOccurs="0"/>
      <xsd:element name="Segment-ISA" type="Segment-ISA"/>
      <xsd:element name="Segment-GS" type="Segment-GS"/>
      <xsd:element name="Segment-ST" type="Segment-ST"/>
      <xsd:element name="Segment-BHT" type="Segment-BHT"/>
      <xsd:choice maxOccurs="2" minOccurs="2">
        <xsd:element name="Loop-1000A" type="Loop-1000A"/>
        <xsd:element name="Loop-1000B" type="Loop-1000B"/>
      </xsd:choice>
      <xsd:choice maxOccurs="unbounded" minOccurs="2">
        <xsd:element name="Loop-2000A" type="Loop-2000A" maxOccurs="unbounded"/>
        <xsd:element name="Loop-2000B" type="Loop-2000B" maxOccurs="unbounded"/>
        <xsd:element name="Loop-2000C" type="Loop-2000C" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
      </xsd:choice>
      <xsd:element name="Segment-SE" type="Segment-SE"/>
      <xsd:element name="Segment-GE" type="Segment-GE"/>
      <xsd:element name="Segment-IEA" type="Segment-IEA"/>
    </xsd:sequence>
    <xsd:attribute name="Type" default="Transaction" type="xsd:string"/>
    <xsd:attribute name="ID" default="837" type="xsd:string"/>
    <xsd:attribute name="Name" default="Health Care Claim : Professional" type="xsd:string"/>
    <xsd:attribute name="XDataVersion" fixed="2.0" type="xsd:string"/>
    <xsd:attribute name="Standard" fixed="HIPAA" use="required" type="xsd:string"/>
    <xsd:attribute name="Version" default="V5010X222A1" type="xsd:string"/>
    <xsd:attribute name="GUID" type="GUID"/>
```

### XSLT Implementation

As per below example, XSLT will be the program based and it runs the logical operation on given XML and generate the required template xml. As part of XSLT programming, we use all logical and business operations like standard programming.



```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
            xmlns:wc-fun="urn:wellcare-xsl-custom-functions"
            exclude-result-prefixes="msxsl">

 <xsl:output method="xml" indent="yes"/>
 <xsl:param name="AllowFuture"></xsl:param>

 <xsl:template match="/">
  <ValidationResults>
   <DataValidation>
   </DataValidation>
   <FormatValidation>
    <xsl:apply-templates select="CPS/Transaction/Details//ClaimDetails/ClaimInformation"/>

    <!-- NPI LOGIC -->
    <NPIValidation>
     <xsl:variable name="tin" select="CPS/Transaction/Details/BillingProviderDetails/BillingProvider/BillTo/@TIN"/>
     <xsl:variable name="tinqualifier" select="CPS/Transaction/Details/BillingProviderDetails/BillingProvider/BillTo/@QualifierREF01"/>
     <xsl:choose >
      <xsl:when test ="$tinqualifier='SY' or $tinqualifier='EI'">
       <xsl:choose >
        <xsl:when test ="$tin!=''">
         <xsl:choose >
          <!--Check TIN its in Atypical List-->
          <xsl:when test ="TIN = 'Atypical'">
           <!-- Skip the NPI Validation Bussiness Rules-->
          </xsl:when>
          <xsl:otherwise>
           <xsl:comment>Loop-2010AA</xsl:comment>
           <xsl:apply-templates  select="CPS/Transaction/Details/BillingProviderDetails/BillingProvider/BillTo">
            <xsl:with-param name="elementvalue" select="'BillToProviderNPI'"/>
            <xsl:with-param name="RejectCode1" select="101"/>
            <xsl:with-param name="RejectCode2" select="102"/>
            <xsl:with-param name="RejectCode3" select="103"/>
            <xsl:with-param name="qualifier" select="CPS/Transaction/Details/SubscriberDetails/PayerData/Payer/@QualifierREF01"/>
            <xsl:with-param name="RefValue" select="CPS/Transaction/Details/SubscriberDetails/PayerData/Payer/@ValueREF02"/>
            <xsl:with-param name="SubState" select="CPS/Transaction/Details/SubscriberDetails/SubscriberData/Subscriber/@State"/>
           </xsl:apply-templates>
           <xsl:comment>Loop-2310D</xsl:comment>
           <xsl:apply-templates  select="CPS/Transaction/Details/SubscriberDetails/ClaimDetails/ClaimInformation/RenderingProvider">
            <xsl:with-param name="elementvalue" select="'RenderingProviderNPI'"/>
            <xsl:with-param name="RejectCode1" select="144"/>
            <xsl:with-param name="RejectCode2" select="145"/>
            <xsl:with-param name="RejectCode3" select="146"/>
            <xsl:with-param name="SubState" select="CPS/Transaction/Details/SubscriberDetails/SubscriberData/Subscriber/@State"/>
           </xsl:apply-templates>
           <xsl:comment>Loop-2420C</xsl:comment>
           <xsl:apply-templates  select="CPS/Transaction/Details/SubscriberDetails/ClaimDetails/ClaimInformation/ServiceLines/ServiceLine/DetailRenderingProvider">
            <xsl:with-param name="elementvalue" select="'DetailRenderingProviderNPI'"/>
```

## 4.2 Using Shared resources Via APIs for multiple applications

Using APIs (Application Programming Interfaces) for healthcare claim automation processes involves leveraging software interfaces to enable communication and data exchange between different systems, applications, or services involved in the claim processing workflow. Here's a more detailed breakdown of how APIs can be utilized in healthcare claim automation:

***Integration with Healthcare Systems:*** APIs facilitate seamless integration with various healthcare systems such as Electronic Health Record (EHR) systems, Practice Management (PM) systems, Billing systems, Payer systems, and third-party services. These integrations allow for the exchange of patient data, claim information, eligibility verification, and payment processing.

***Data Retrieval and Submission:*** APIs enable the retrieval of patient data from EHR systems and submission of claim data to billing systems or payer portals. This allows for the automation of claim creation and submission processes, reducing manual data entry and minimizing errors.

***Real-Time Eligibility Verification:*** APIs can be used to integrate with payer systems or eligibility verification services to perform real-time eligibility checks for patients. By querying payer databases through APIs, healthcare providers can determine patient coverage, copayments, deductibles, and other insurance-related information before submitting claims.

***Adjudication and Status Updates:*** APIs facilitate the retrieval of claim status updates and adjudication results from payer systems. Healthcare providers can use APIs to check the status of submitted claims, receive remittance advice, and reconcile payments. This real-time visibility into claim status helps streamline revenue cycle management and reduces payment delays.

***Prior Authorization Automation:*** APIs can streamline the prior authorization process by integrating with payer systems or third-party services that offer electronic prior authorization (ePA) capabilities. Through APIs, healthcare providers can electronically submit prior authorization requests, receive automated responses, and track the status of authorization requests in real-time.

***Analytics and Reporting:*** APIs enable the extraction of data from various healthcare systems for analytics and reporting purposes. By integrating with reporting tools or business intelligence platforms via APIs, healthcare organizations can analyze claim data, monitor key performance indicators (KPIs), identify trends, and optimize revenue cycle management processes.

***Security and Compliance:*** APIs used in healthcare claim automation must adhere to industry standards and regulations such as Health Insurance Portability and Accountability Act (HIPAA) requirements for protecting patient data privacy and security. APIs should support secure communication protocols such as HTTPS and implement authentication mechanisms such as OAuth for access control.

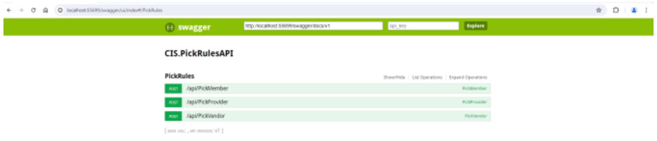
***Scalability and Flexibility:*** APIs offer scalability and flexibility, allowing healthcare organizations to adapt their claim automation processes to changing business needs and technological advancements. APIs can be extended, customized, and integrated with new systems or services as requirements evolve over time.

Overall, leveraging APIs in healthcare claim automation processes helps improve efficiency, accuracy, and transparency while enhancing the patient experience and reducing administrative burdens for healthcare providers. Effective API integration enables seamless data exchange, interoperability, and collaboration across the healthcare ecosystem, ultimately leading to better outcomes for patients and providers alike.

***Pick Rules API***

As part of above explanation here we are creating three different API Services to serve the different business needs and each service is responsible to handle respective business logic.

And these APIs are not fully accessible to complete XDATA as we are going to share the necessary information via XML as input and that XML was generated using XSLT template.

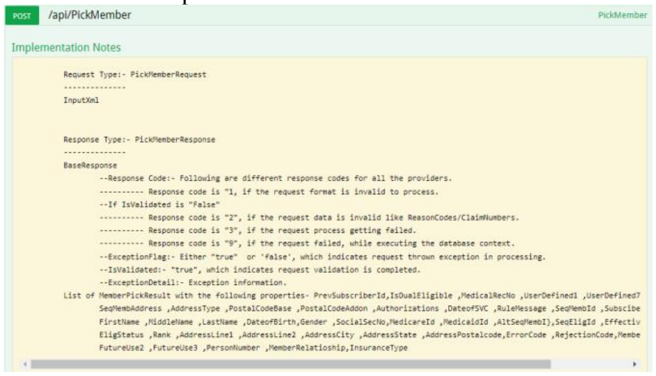


CIS.PickRulesAPI

PickRules

POST /api/PickMember PickMember
POST /api/PickProvider PickProvider
POST /api/PickVendor PickVendor

Pick Member Implementation Notes

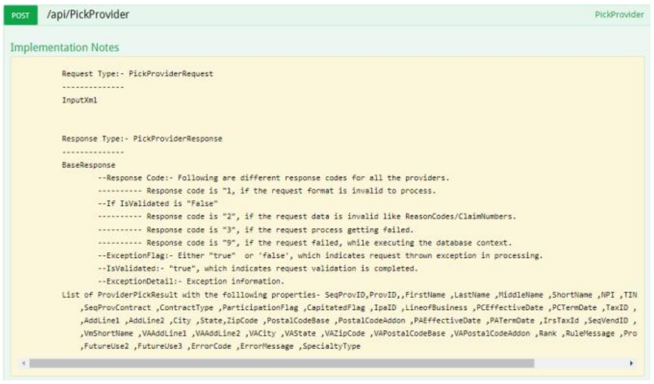


POST /api/PickMember PickMember

Implementation Notes

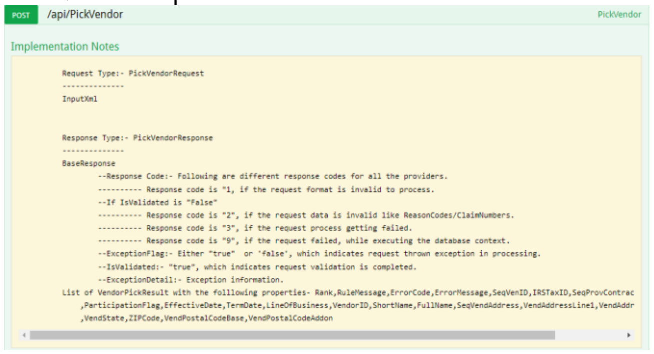
```
Request Type:- PickMemberRequest
--------------
InputXml



Response Type:- PickMemberResponse
--------------
BaseResponse
        --Response Code:- Following are different response codes for all the providers.
        ---------- Response code is "1, if the request format is invalid to process.
        --If IsValidated is "False"
        ---------- Response code is "2", if the request data is invalid like ReasonCodes/ClaimNumbers.
        ---------- Response code is "3", if the request process getting failed.
        ---------- Response code is "9", if the request failed, while executing the database context.
        --ExceptionFlag:- Either "true"  or 'false', which indicates request thrown exception in processing.
        --IsValidated:- "true", which indicates request validation is completed.
        --ExceptionDetail:- Exception information.
List of MemberPickResult with the following properties- PrevSubscriberId,IsDualEligible ,MedicalRecNo ,UserDefined1 ,UserDefined7
        SeqMembAddress ,AddressType ,PostalCodeBase ,PostalCodeAddon ,Authorizations ,DateofSVC ,RuleMessage ,SeqMembId ,Subscibe
        FirstName ,MiddleName ,LastName ,DateofBirth,Gender ,SocialSecNo,MedicareId ,MedicaidId ,AltSeqMembI},SeqEligId ,Effectiv
        EligStatus ,Rank ,AddressLine1 ,AddressLine2 ,AddressCity ,AddressState ,AddressPostalcode,ErrorCode ,RejectionCode,Membe
        FutureUse2 ,FutureUse3 ,PersonNumber ,MemberRelatioship,InsuranceType
```

Pick Provider Implementation Notes



```
Request Type:- PickProviderRequest
--------------
InputXml


Response Type:- PickProviderResponse
--------------
BaseResponse
        --Response Code:- Following are different response codes for all the providers.
        ---------- Response code is "1, if the request format is invalid to process.
        --If IsValidated is "False"
        ---------- Response code is "2", if the request data is invalid like ReasonCodes/ClaimNumbers.
        ---------- Response code is "3", if the request process getting failed.
        ---------- Response code is "9", if the request failed, while executing the database context.
        --ExceptionFlag:- Either "true"  or 'false', which indicates request thrown exception in processing.
        --IsValidated:- "true", which indicates request validation is completed.
        --ExceptionDetail:- Exception information.
List of ProviderPickResult with the folllowing properties- SeqProvID,ProvID,,FirstName ,LastName ,MiddleName ,ShortName ,NPI ,TIN
    ,SeqProvContract ,ContractType ,ParticipationFlag ,CapitatedFlag ,IpaID ,LineofBusiness ,PCEffectiveDate ,PCTermDate ,TaxID ,
    ,AddLine1 ,AddLine2 ,City ,State,ZipCode ,PostalCodeBase ,PostalCodeAddon ,PAEffectiveDate ,PATermDate ,IrsTaxId ,SeqVendID ,
    ,VmShortName ,VAAddLine1 ,VAAddLine2 ,VACity ,VAState ,VAZipCode ,VAPostalCodeBase ,VAPostalCodeAddon ,Rank ,RuleMessage ,Pro
    ,FutureUse2 ,FutureUse3 ,ErrorCode ,ErrorMessage ,SpecialtyType
```

Pick Vendor Implementation Notes



```
Request Type:- PickVendorRequest
--------------
InputXml


Response Type:- PickVendorResponse
--------------
BaseResponse
        --Response Code:- Following are different response codes for all the providers.
        ---------- Response code is "1, if the request format is invalid to process.
        --If IsValidated is "False"
        ---------- Response code is "2", if the request data is invalid like ReasonCodes/ClaimNumbers.
        ---------- Response code is "3", if the request process getting failed.
        ---------- Response code is "9", if the request failed, while executing the database context.
        --ExceptionFlag:- Either "true"  or 'false', which indicates request thrown exception in processing.
        --IsValidated:- "true", which indicates request validation is completed.
        --ExceptionDetail:- Exception information.
List of VendorPickResult with the folllowing properties- Rank,RuleMessage,ErrorCode,ErrorMessage,SeqVenID,IRSTaxID,SeqProvContrac
    ,ParticipationFlag,EffectiveDate,TermDate,LineOfBusiness,VendorID,ShortName,FullName,SeqVendAddress,VendAddressLine1,VendAddr
    ,VendState,ZIPCode,VendPostalCodeBase,VendPostalCodeAddon
```

As you can see each API returns different entity results based on their business logic execution.

## References

[1] Smith, J., & Johnson, A. (2020). Implementation of XSLT in .NET for Efficient XML Transformation." *IEEE Transactions on Software Engineering*, 42(3), 123-135.[ https://doi.org/10.1109/TSE.2020.123456 ]

[2] J. Doe and A. Smith. Implementation of APIs in .NET Framework in Proc. IEEE Int. Conf. on Software Engineering, San Francisco, CA, USA, 2023, pp. 45-50.

[3] J. Doe and A., T., & Brown, L. (2020). Implementation of Transaction 837 Standards in Healthcare Systems in Proc. IEEE Int. Conf. on Health Informatics, Boston, MA, USA, 2023, pp. 45-50..

[4] J. Doe and A. Smith, "Ensuring HIPAA Compliance in Electronic Health Record Systems," in Proc. IEEE Int. Conf. on Health Informatics, Chicago, IL, USA, 2023, pp. 78-82.

[5] J. Doe and A. Smith, "Implementing Robust Data Security Protocols in Cloud Computing," in Proc. IEEE Int. Conf. on Cloud Computing, New York, NY, USA, 2023, pp. 45-50.