Converting Medical Service Provider Data into a Unified Format for Processing

Brandon Krugman
Marquette University

CONVERTING MEDICAL SERVICE PROVIDER DATA INTO A UNIFIED FORMAT FOR PROCESSING

by

Brandon Krugman

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

August 2015

ABSTRACT
CONVERTING MEDICAL SERVICE PROVIDER DATA
INTO A UNIFIED FORMAT FOR PROCESSING

Brandon Krugman

Marquette University, 2015

Most organizations process flat files regularly. There are different options for processing files, including SQL Server Integration Services (SSIS), BizTalk, SQL import job, and other Extract, Transform, and Load (ETL) processes. All of these options have very strict requirements for file formats. If the format of the file changes, all of these options throw a catastrophic error, and implementing a fix to handle the new format is difficult. With each of the methods, the new format needs to be configured in the development environment, and the data flow must be modified to process all of the changes.

Due to the inflexibility of options in processing flat files, there was a request by Dr. Corliss to build an alternative solution. The team of Ivan Paez, Niharika Jain, and Brandon Krugman created an alternative solution called FileParser. While the solution originally was built to meet the needs of Dr. Corliss and the GasDay team at Marquette University, the end result was a file parser that allows additional flexibility in processing of a variety of flat file formats.

This thesis provides an alternative way to parse data, transform a flat file, and consume the data into a generic format; this process is called Provider Processing. Provider File Processing consists of the FileParser command line executable handling the file parsing and data transformation. After FileParser generates a provider output file, a health insurance domain-specific command line executable called DelegatedProviderProcessing performs data cleansing, address normalization, and imports the provider output file into an internal database. The difference between the strict format examples and Provider Processing is that if the format of the input files change, Provider Processing can adapt to the change with minimal work being completed.

ACKNOWLEDGEMENTS

Brandon Krugman

I would first like to thank Dr. Kaczmarek and the rest of the committee for helping me to reach the end of my Master's program and helping me to continue on my path forward. Also, I would like to thank my wife Kim, who put in a lot of time proofreading, editing, and providing guidance towards the creation of my thesis. Finally, I would like to thank my son Finn for providing the little distractions that a person sometimes needs to be able to keep their mind sharp.

TABLE OF CONTENTS

Problem

Many health insurance organizations are building efficient processes to keep costs down, while providing the maximum level of care to members. There are some key areas that can be improved to reduce the quantity of work and also reduce the overhead costs required to operate an insurance plan. In Wisconsin, the Wisconsin Statewide Health Information Network (WISHIN) [2] is laying the ground work to distribute information efficiently to member health insurance organizations. WISHIN will reduce the quantity of work required to process information, while also allowing insurance companies to provide the maximum level of care to their members.

Health insurance organizations frequently struggle with a lack of standards for information formatting. This results in inconsistent formatting of information that is exchanged. An example of this is the information available about healthcare providers. WISHIN Provider Directories [3] currently only contain the name, organization, address, and contact information for a provider, but are missing important information such as the provider's specialty. In order for health insurance organizations to supply the correct information to insurance plan members, this is a key detail. Files that medical providers submit directly to a health insurance plan usually contain a provider's specialty area formatted as either an expertise or a specialty. In Figure 1 this is shown at a high level, without any data processing represented. The high level view shows the two provider data types that a health insurance organization receives and the processing that occurs to allow the provider data to be imported into an internal database.

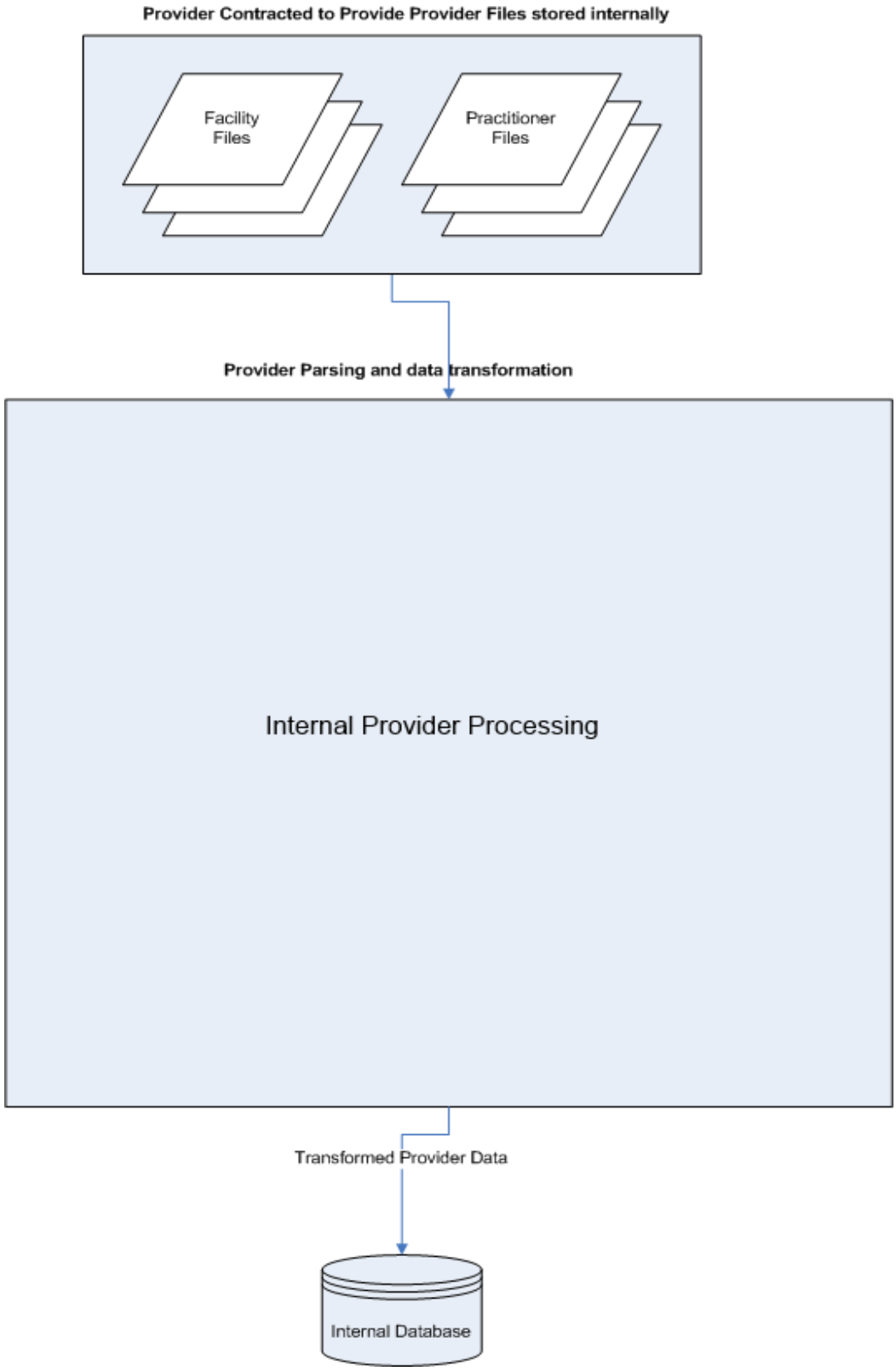


Figure 1: High Level Representation of the Provider Data Process

Problems encountered with this process:

- Different provider file types contain different information attributes
- Different files from the same provider can use different formats
- Different providers use different formats
- Formats are able to be changed without notice

There are a few ways in which the problems listed above are commonly handled: a unique process can be created for each format typically in SQL Server Integration Services (SSIS) or a similar Extract, Transform, and Load (ETL) process; a team of staff members can manually update and add the information from the files; or some other mechanism can parse, transform, and import the data. Health insurance organizations need to ensure that they will have all information they need about medical providers and that the information is formatted consistently.

In this thesis, the FileParser command line executable created for GasDay [1] is implemented, along with a health insurance domain-specific command line executable called DelegatedProviderProcessing. The overall process that combines the two executables is referred to as Provider Processing. This process will parse the provider data and transform it to match the provider formats described in Appendix 1.2 by using the FileParser command line executable. Then after FileParser creates the provider output file, the DelegatedProviderProcessing command line executable will import, cleanse, and normalize all of the data before it is imported into the internal database.

To articulate the benefits of Provider Processing, this thesis will first offer a Situation Analysis. Then the thesis will investigate the Previous Solution and the Current Solution. Finally, this thesis will provide an Evaluation of Provider Processing.

Situation Analysis

Insurance organizations commonly deal with both servicing and billing information. Servicing information is sent to the network development group, and billing information is sent to the claims processing group. There are major differences between these two types of information. The servicing information provides attributes such as the provider's specialty/expertise, the location(s) from which they operate, the language(s) they speak, whether their location is a primary location, and other information that someone typically would find in a directory or on a health care provider's web site. Billing information typically contains only the necessary servicing detail to allow the health insurance organization to be able to pay a claim. The servicing information and billing information can be present in the provider's files that are processed in Provider Processing, but the billing information is not required since some provider organizations do not supply it in the files that they submit. Both sets of information are very important to the health insurance organization; however, the focus of Provider Processing is to solve the issue of processing medical provider servicing information to ensure the accuracy of provider data for the health insurance organization's members.

Provider Processing addresses the problems associated with servicing information that is received from medical providers. The provider files indicate who the health insurance organization is currently contracted with to provide services. There is an agreement between the health insurance organizations and the medical provider organizations to provide information files at a regular interval, ranging from monthly to yearly. The main requirement for processing this data is the ability to provide information to health insurance organization's members, which allow the insurance company's care coordinators to ensure that a member is receiving the care needed and to provide regulatory data.

Provider Processing implements the FileParser command line executable to meet similar needs that existed for GasDay. The similarities between the problem Provider Processing solves and GasDay are that the format of the data can be different with each file received, the format can change without any notice, and the file is received on a consistent basis which causes the data to need to be updated regularly. To meet these needs of GasDay, the FileParser was created as a dynamic-link library (DLL) and accessed through a command line executable. Provider Processing FileParser also has a command line executable that allows it to be passed an XML configuration file containing the desired output. By using the XML configuration file, Provider Processing is able to produce files in a single provider file format for all provider files that are received the output file is independent of whether the provider file is a facility or a practitioner file.

Previous Solution

The previous solution is an SQL Server Integration Services (SSIS) package that contains different DTSX [4] files, Data Transformation Services (DTS) settings files that Microsoft developed to provide a simple user interface to create, update and maintain DTS files, for each data migration task. Each medical provider input file has a DTSX file that contains the provider file location that is imported and inserted into a working table that matches the provider input format. Figure 2 shows a data flow of how this process operates.

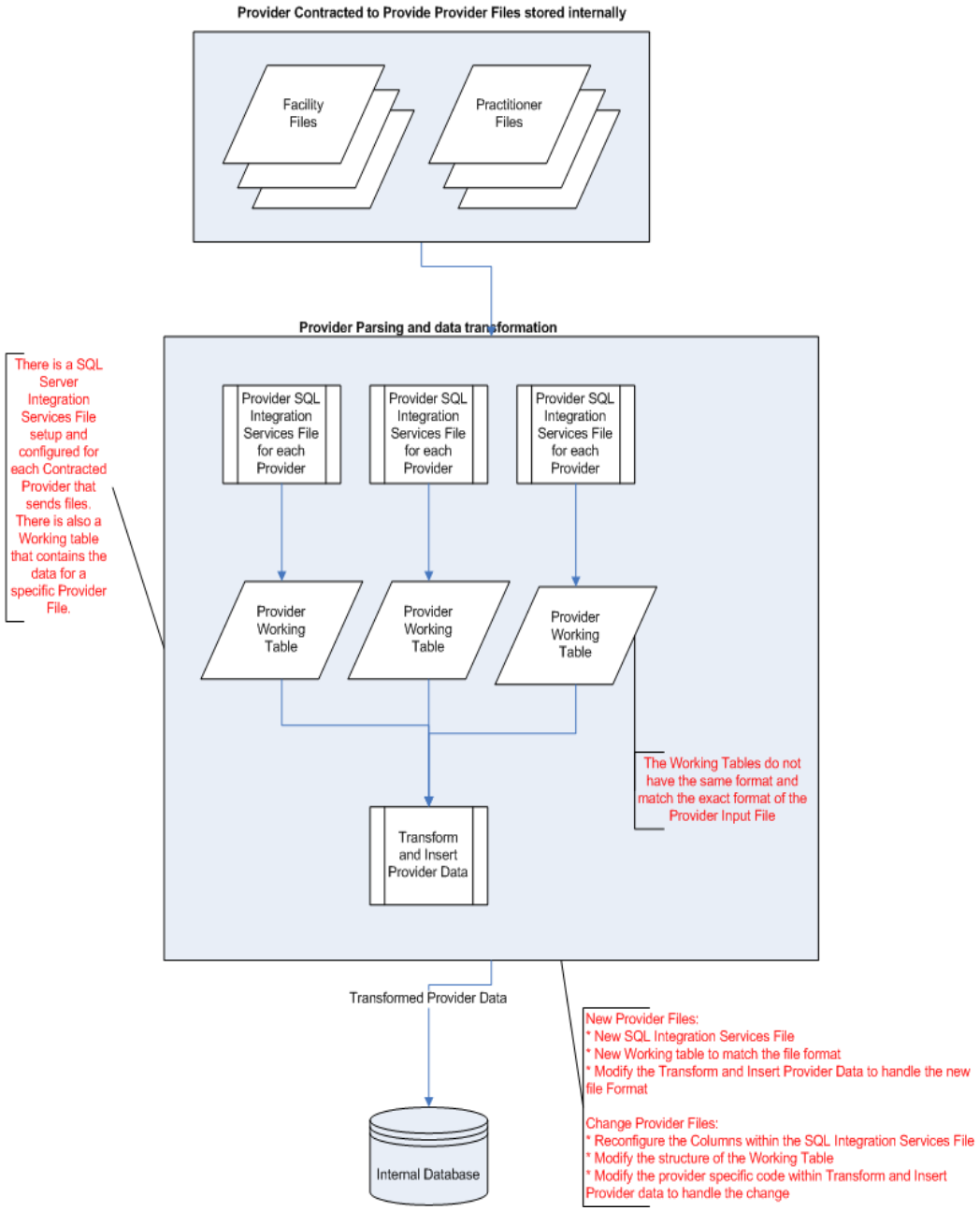


Figure 2: High-level data flow of the previous solution to process provider data

This process is currently handles nine provider input files. The solution works as long as the file formats remain consistent, which happens infrequently. Implementing the SSIS package for new file formats is time consuming, due to the entirely manual column mapping that is required and creating a working table structure for each file. Also, every new format, and changes to an existing format, requires manual code changes to the Transform and Import Provider Data application that take the data from the working tables and insert the data into the internal database. The DTSX file generates the columns by opening the file reference that is contained in the file connection manager. If the flat file connection manager and the current column configuration are using different structures, someone manually updates the connection manager to ensure that the input file and the connection manager are synchronized. Once the input file is imported, it is inserted into a working table that mirrors the file format of the input file. If someone has to update the file connection manager, they also have to drop the working table and create a new one with the correct format for the parsing to work. Someone also has to manually update the application that reads the working table to ensure that the new format is acceptable and that the application is able to import the data into the internal database.

The custom processing for each file accounts for missing information that might be present on one file, but not on the others such as languages spoken, or if the location is the provider's primary location. An additional difficulty in working with the data in this way is a situation where the provider represents the city, state, and zip code of a location in a single column rather than separating the information. To handle this, custom code needs to be developed to make sure that the information for the address is separated correctly and validated.

While SSIS is a good method for performing process automation and data transformation, it tends to be a complicated and very rigid when trying to adapt to any change in the process, whether it is in the input, processing, or output. A simple processing example of applying the

format from the flat file connection manager and inserting the data into a working table can be seen in Figure 3. Strengths and weaknesses of using SSIS include:

Strengths

- Ability to build a visible process flow with descriptive names for each step in the process
- Ability to set up data sources as dynamic or static depending on the needs of the solution
- Ability to build a process flow that can import the data and insert directly into a database structure
- Ability to configure logging and monitoring to help troubleshoot issues mid-process
- Ability to perform data type transformation
- Ability to add custom script code to handle non-standard processing requirements
- Custom code is similar to Microsoft .Net

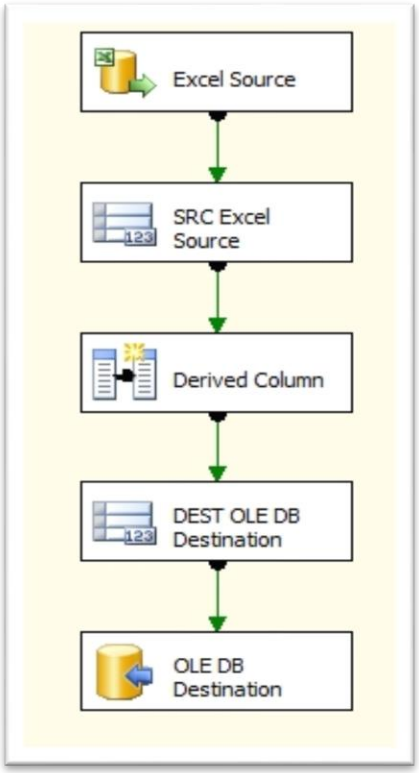


Figure 3 DTSX process to parse a flat file into a working table

Weaknesses

- Can take a lot of time to create the original process setup
- Unable to adapt quickly and seamlessly to new or changing requirements in file formats or processing
- Requires SQL Job Agent to be configured to run a package in a production environment
- Unexpected changes can break the cause the process to fail

- Limited in the type of custom code that can be developed
- Can require a new file to be built to handle different file formats

Current Solution

Provider Processing provides the flexibility, in implementation and use, needed to handle an organization's unique challenges. Figure 4 shows how Provider Processing handles the multiple provider files differently than the SSIS solution. Provider Processing achieves this by using FileParser to change the structure of the input files to match a set, common format. DelegatedProviderProcessing then imports, cleanses, and normalizes the address data to ensure that all provider data is handled the same way. This is also a unique position, because if the backend database needs to change, the DelegatedProviderProcessing is the piece that has to change allowing for minimal work to implement any database changes.

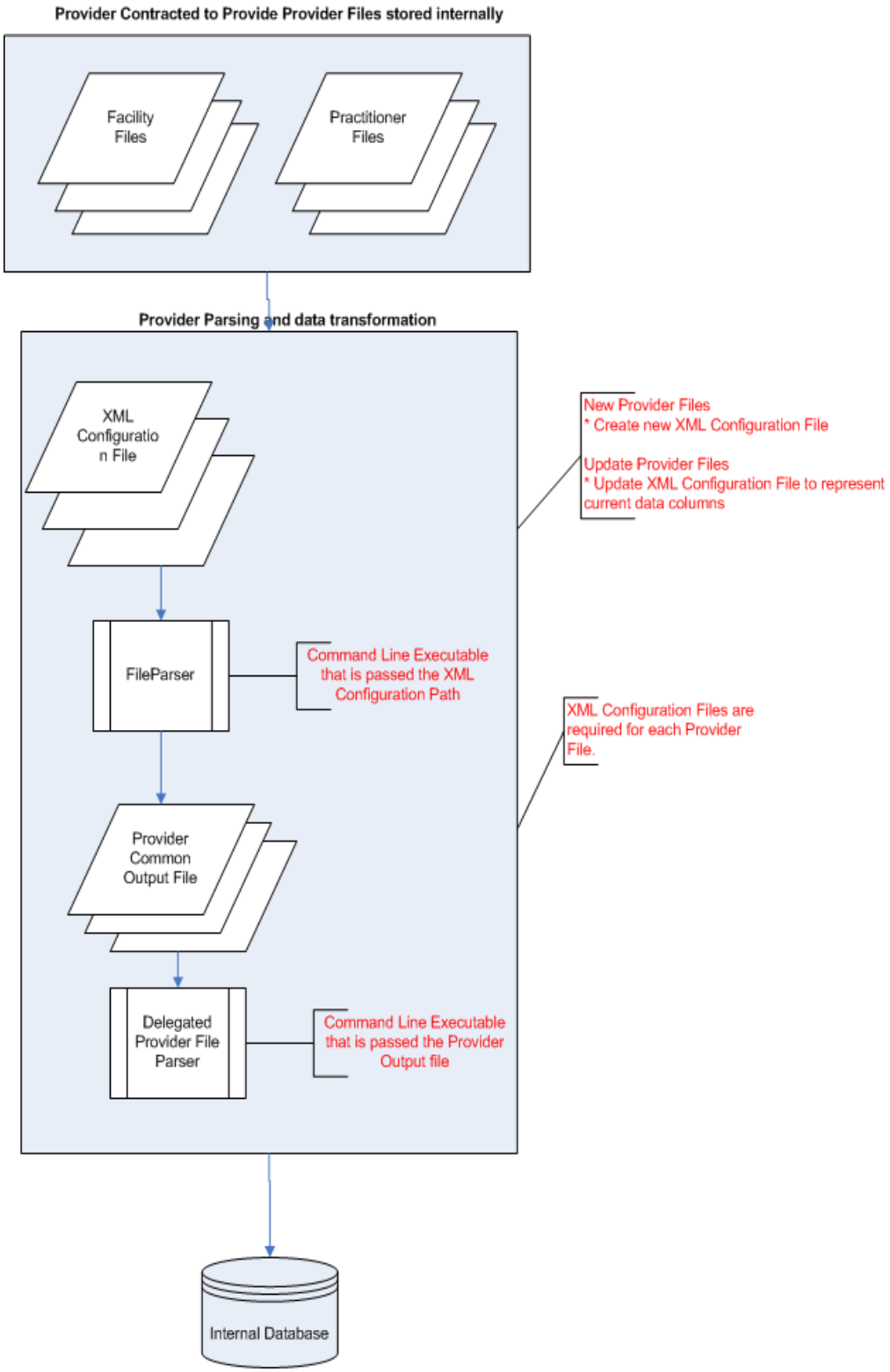


Figure 4: High-level data flow of the current process to handle provider data

Provider Processing

Goal of Provider Processing

Provider Processing looks to provide a way to make the processing of medical provider data simpler and more uniform for an insurance company.

Objective

Reduce the maintenance costs, user interaction costs, and development costs from what they would have been if a process was built for handling each file format. This reduction in costs can be seen in Table 1 and Table 2 in the Opportunities section. By using the Provider Processing process, the amount of work needed to create a new provider configuration is under 10 minutes. Also, the amount of time to update a provider database becomes a fraction of a second per provider record.

Developed Process

Define Information Requirement: Establish required (for provider data) fields through review and health insurance team member input to create a single provider object containing all fields.

Develop Process and Technology: Develop a procedure to convert a variety of medical provider file formats into a single format, which will provide a benefit of a single point of entry for medical provider data. This will ensure that the provider data will be treated the same no matter where the data comes from.

Use Case

UC 1	Process Delegated Provider File	
Description	Delegated Provider files are received regularly monthly, quarterly or annually depending on the contract with the Provider.	
Used By	Data in the Provider File is used by various departments in an organization, such as Marketing and Care Coordination.	
Preconditions	We know what Delegated Entity sent the file.	
Success End Condition	Provider File is Parsed, Transformed and Imported into the internal database.	
Failed End Condition	FileParser throws a catastrophic error or the Import Provider process throws an error consuming the data.	
Trigger	Delegated Entity submits its Full Roster List of Providers (Practitioners and Facilities)	
Description	Step	Action
	1	Delegated Entity submits a file flat comma delimited file.
	2	File is moved to the Input folder to wait for automated processing from a scheduled job.
	3	Scheduled job executes processing for all configurations defined in the Configuration folder (See Appendix 1.4 for code) and archives the input files to prevent multiple parsing and transformation.
	4	Files able to be parsed and transform saved to the Output folder
	5	Files in the Output folder are manually validated with the Input folders to ensure data is parsed and transformed correctly.

	6	A process is manually executed to cleanse and processes the data from each output file.
Variations		Branching Action
	1a	No Configuration defined for File

	1b	Create a Configuration file for the file using the Provider Template.
		• Templates configured
		  ○ Practitioner
		  ○ Facility

Other Information	Average Processing and creation times • Creation of Provider Configuration File (using template) : 5 minutes • Modifying file to handle changes : 2 minutes • Generating additional provider configuration file for same input source: 2 minutes File Parsing and Transformation • 0.004 seconds per row in file Data Cleanse and processing • 0.0038 seconds per row in file	

Implementing Provider Processing

Define Information Requirement: Provider Object

To create a generic provider format, all current medical provider input files were reviewed, and certain fields were flagged as required for provider data. Along with those fields, a team in a health insurance organization also established fields they require to meet the needs of the health care organization's members, marketing staff, care coordinators and government regulators. All of the fields as key provider data were used to create a provider object, as seen in Figure 5. A more detailed class diagram is available in Appendix 1.5.

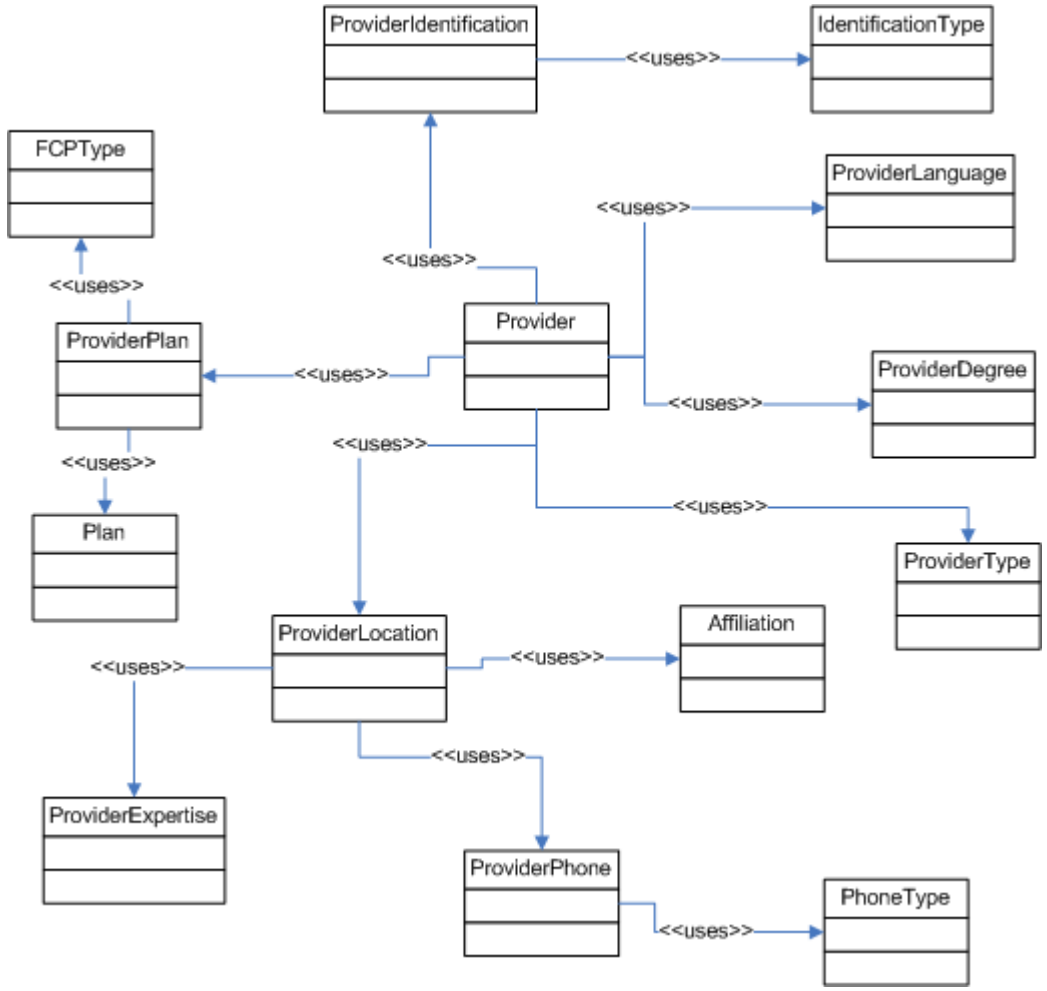


Figure 5: High-level class diagram of the provider object

Populating the provider object, requires logic that will add information that is missing from the provider files, but still required for the data to be valuable to an insurance company. This type of data includes things like the state-designated provider type, state-designated taxonomy, and federal-designated taxonomy. Also, logic is required to perform data cleansing tasks such as address normalization to help prevent duplicate information where an address is represented in two different ways, e.g. 100 Test Avenue versus 100 Test Ave. By normalizing the address information, the DelegatedProviderProcessing is able to correctly group information that shares the same provider and address. Since provider data that is being processed can be both medical practitioner data and facility information, this is handled in the Provider object by allowing DelegatedProviderProcessing to set the provider type. Defining what type of provider is currently in a Provider object allows the update to handle facility information differently if the database table structure requires that facilities be stored differently. This allows DelegatedProviderProcessing to be flexible enough to perform separate actions for medical providers and facilities if the database and the data structure require it.

For data elements such as Directory Category and Directory Sub Category, business logic needs to determine where the insurance organization wants the providers to appear in both online and printed directories for members. The business rules specifying which providers need to be published in the different categories was set by staff in the network development and marketing teams to ensure that the regulatory needs were being met, as well as the needs of the health insurance organization's members. While defining the rules for how providers appear in a directory, the network development and marketing team also defined a set of rules for which providers should not be listed in a directory, because they provide a type of service that a member would not contact directly. These types of providers include anesthesiologists, radiologists, and emergency room doctors. While all providers need to be stored in the provider data to meet

different requirements of the business, the providers should not be contacted directly by a member.

Develop Process and Technology: File Parsing

To process the various medical provider formats, a procedure is needed to convert the unique formats into a single format. Using a single format rather than inserting the data directly into a database table provides a method to archive both the information that was sent to the health insurance organization and how the data was transformed. Another benefit of the interim step of creating a single file rather than a working table is the ability to add additional validation before the data is loaded into the database to help prevent bad data from getting into the system. Performing the file parsing and data transformation is completed and contained in the FileParser command line executable. FileParser is the key piece allowing the flexibility to handle the current provider formats and any new provider format. The data flow of the FileParser is shown in Figure 7, located in the SWOT analysis for FileParser.

The FileParser requires one input, an XML path, to perform the data parsing and transformation of the provider file. Using an XML file as a configuration file provides multiple benefits. The two most significant benefits are that we can ensure uniformity and easy alteration when setting up additional files. In FileParser, the `ValidConfigurationFilePath` function takes the XML configuration file and ensures that the file can be opened. After determining that the XML configuration file can be opened, the function `SetLogFilePath` is called to set the log reporting path. The log location is used to report any errors, no matter the error level, and any other information that FileParser returns to the command line, such as how many records are parsed and transformed.

Once the log path is validated and set within FileParser, the function `BreakdownConfigurationFile` is called to create a list of field attributes, including the

locations of each input and output column, the data type that is expected, and if the file needs to be transposed. The high-level FileParser class diagram is represented in Figure 6, the detailed FileParser class diagram is located in Appendix 1.6, showing the class that allows FileParser to perform the parsing and transform while reading lines from the file. By saving the XML configuration file attributes into a list of Field objects, FileParser is able to use the field object list when reading each line in the file and generate the output for the line immediately after the output format is applied for all supported file structure types.

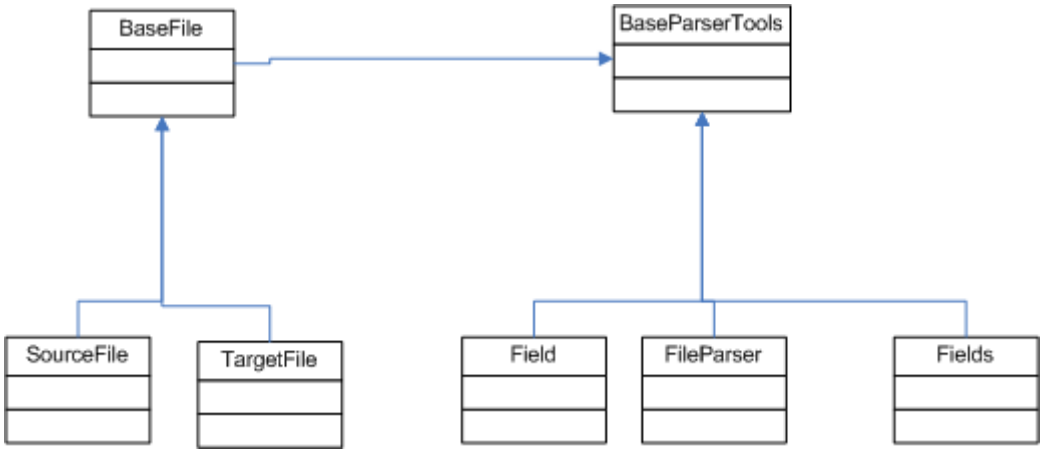


Figure 6: High-level FileParser class diagram representing the objects being used

After the XML configuration file is parsed into the list of Fields, the function `WriteOutput` is called to read a line of data in and apply the output format. This function performs the input and output of FileParser for each line in the file. By reading each line individually, FileParser is able to determine if there is an error with the line. Based on the result of the line validation, FileParser proceeds to the next line in the file or terminates the parsing due to the line having a major issue. When developing FileParser, the development team considered reading the entire file into memory and then iterating through each line. The team chose to read and write each line because there was no performance difference, and the system could validate each line. This approach can lead to higher I/O overhead, but the overall processing time is low, so the higher I/O overhead was viewed as acceptable. Another benefit of reading and writing the

lines individually is if a custom action needs to be done based on a value that is in the line being written. The functionality in FileParser allows a process to see each line being written. Based on the data in the line, FileParser can write it to a different output file. An example of this was implemented into a production environment, where FileParser is used to read a file that contains members with different types of health plan memberships, and there is a need to construct separate files for each type of membership.

Sample XML

In the Configuration XML that drives and controls the entire process, there are three XML Nodes that indicate where the Input File is located, where the Output File needs to go, and where the Log File should be stored. Each of these locations can exist on network drives, making it easier for a company to run this process on one machine, while maintaining the files in different locations. Below is an example of the File XML Nodes. In the sample, the process will take a comma delimited file and transform it into a tab delimited file.

XML File Input, Output and Log File nodes

```
  <SourceFile path="InputFiles\TestIn.csv" delimiter="," 
transpose="false" />

  <TargetFile path="OutputFiles\TestOut.txt" delimiter="/t" 
removeTextQualifiers="false" append=”false”/>

  <LogFile path="LogFiles\TestLog.log" />
```

Following the XML Nodes that reference the files needed to perform the processing, there is the Fields Node that contains all of the fields that are represented in the Output file (Appendix 1.1).

XML Field Attributes

```
<Field inName="SourceFieldName"     inPosition=""
outName="DestinationFieldname" outPosition="1" type="string"
length="" format="" lowBound="" highBound="" okNull="true"
defaultValue="" columnOptional="" includeOptionalColumn="" />
```

Process Flow

In the Provider Processing business process (Figure 4), the process flow begins with the network development team communicating to the provider network support staff that there is a provider file available. Currently, two different types of provider data files are received. The type that is received most frequently and provides the best data is the full extract of the health care organization's medical professionals and facilities. When this type of file is received, the file's XML configuration file is sent to FileParser to create the provider output format. The provider output format is then sent to DelegatedProviderProcessing, where the file data goes through data cleansing and address normalization before it is added or updated in the database. Any medical professional associated with the provider network being processed that is not included in the file and has not been updated within the last three (3) months is flagged as terminated so it will not show up in the online directories. This informs the care coordination staff that the provider is no longer in the health insurance organization network.

The second type of processing file that is received is an update to the full roster and facilities list that was previously sent. Not many providers send this type of file, but there is a

process to handle these types of files to ensure that provider data remains current. This process looks through the file that is sent and applies updates to individual provider records using the new information that is associated with the provider network that supplied the file.

The system always must ensure that it is only updating data that is associated with the provider network that is being processed. Since providers can have multiple affiliations, with different provider networks or private practices, the system only wants to update the records that are associated to the provider network being processed. By doing this Provider Processing ensures that the data being processed is done correctly and prevents non-affiliated data from being changed by the process.

Once the health insurance organization's data is updated, it is available immediately for internal use across all departments. To provide the updated information for external consumption by the health insurance organization's members and providers, the data is provided via a flat file for external search every other week. There is a user interface that allows care managers and care coordinators to directly answer questions from members on whether a medical professional is in network, and to assist with a member selecting a provider as their primary care physician. The internal provider information allows authorizations to be completed so the medical professionals can be paid for services that they provide to the health insurance organization's member base.

One of the goals for every health insurance organization is to provide the best information for its members. Hence, there is a search tool for members that provide the same functionality as the internal search tool, but with data that represents only providers that are in the health insurance organization's network. The internal search provides access to out-of-network provider information because this information is needed to create authorizations for care when a member decides that they would like to use an out-of-network medical provider.

SWOT (Strengths, Weakness, Opportunities, Threats) Analysis

Provider Processing has inherent strengths and weaknesses, and the environment offers opportunities and poses threats. Below are the strength and weaknesses of each component of Provider Processing.

FileParser

FileParser is a command line executable that takes an XML file path, which it will interpret to perform all processing. Included in the XML file are the input file location, the output file location, any delimiters used in the input and output files, and the fields that will be represented on the output file. Since the command line executable only takes the XML file path, it allows flexible options for implementation and use of the tool. The FileParser is able to be used as is or can have custom coding to perform tasks to generate different output or insert data directly into a database. The data flow of FileParser is displayed in Figure 7, showing each step that the FileParser executes.

Strengths

- Single process to transform any format
- All configuration managed by an XML file
- Easy to adapt to changes in the file formats
- Able to script processing of files on any Windows machine whether Personal Computer (PC) or a server
- Simple console application that can be scheduled and executed by any process (SSIS, Windows Scheduler, SQL Job Agent, Powershell, etc…)
- Provides common output that can be used for custom coding
- Fast and easy to leverage in other domains and areas of business

- Able to handle transposed data files (file data in columns and header information in the first column of a row)
- Able to ignore miscellaneous information above the file header
- No embedded business logic, the executable does not know anything about the data it is parsing except what is given in the XML

Weaknesses

- Could require custom code to perform additional processing
- Requires a process outside of the command line executable to import the data being generated
- Currently requires multiple passes to handle scattered cells in the data file

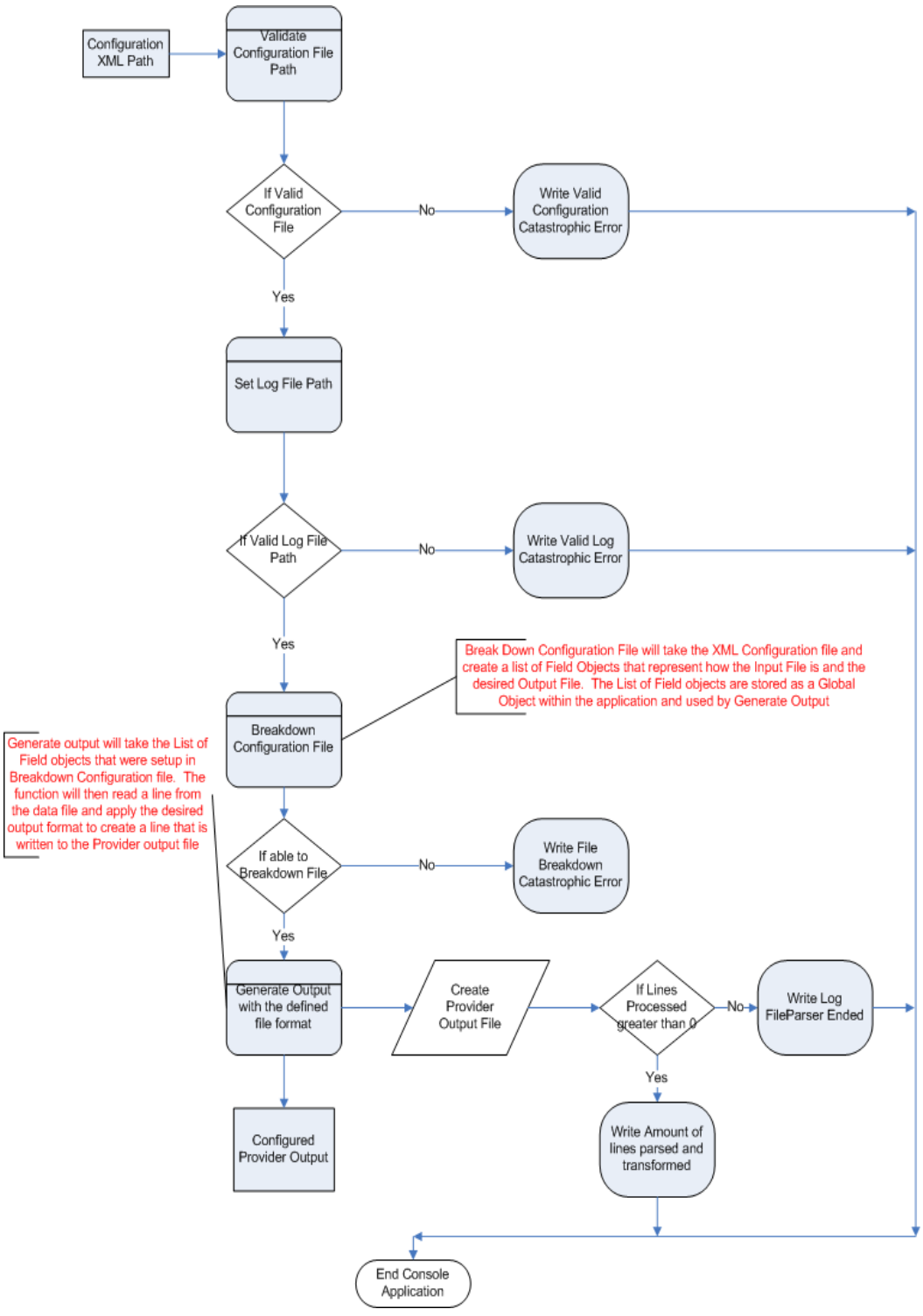


Figure 7: Data flow of the FileParser application that shows how a provider data file is handled

DelegatedProviderProcessing

DelegatedProviderProcessing is a command line executable that is built using a provider object that allows the output from FileParser to represent provider data allowing for easier inserting and updating. Since a provider in a file can have multiple locations and specialties, as well as descriptive information, using an object-oriented approach is useful and allows the Provider data object to group information based on the servicing location. Grouping the provider data together allows DelegatedProviderProcessing to update all of the provider data at the same time, without worrying that the data might be updated or overwritten by a later provider entry in the file. The DelegatedProviderProcessing data flow is represented in Figure 8 and shows all of the data cleansing and additional data that is populated outside of the file generated by FileParser to ensure that the provider data is complete.

Strengths

- Handles address normalization by using United States Postal service address information
- Populates specialty information based on the State Provider database from the state of the provider
- Populates Taxonomy from both the State and the Federal databases
- Provides a single object for each Provider that contains all known and required information pertaining to the provider
- Built without a specific database structure to allow for implementation with a health care organizations provider database
- Simple command line executable that can be scheduled and executed by any process (SSIS, Windows Scheduler, SQL Job Agent, Powershell, etc…)

Weaknesses

- Requires a developer with Microsoft .Net

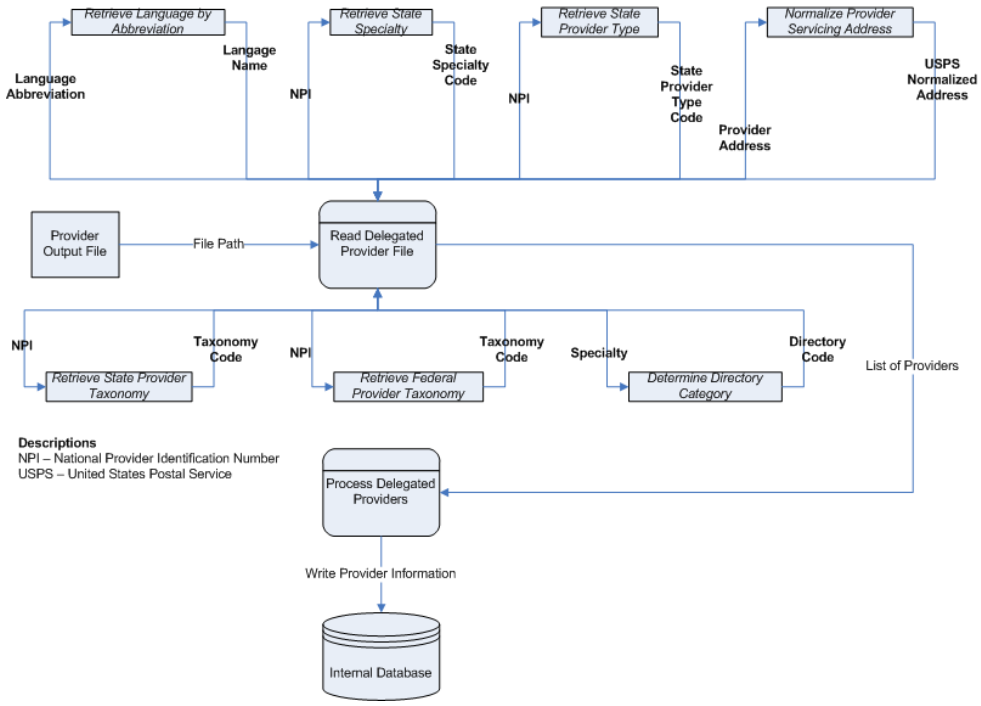


Figure 8: DelegatedProviderProcessing data flow with data cleansing and population of missing data

Opportunities

The existing environment, the health insurance organization's needs and the previous solution provide opportunities for Provider Processing.

- There is a huge opportunity for a tool that will easily parse and transform any flat file format. There are currently 55 configuration files being maintained within Provider Processing.
- Data that is not provided in a universal format can make it difficult to handle and require manual data entry because the data is not located in the same place on all provider files.
- Opportunity for a solution to automate importing and column mapping.
- Insurance companies frequently encounter difficulties when trying to build some level of automation for provider data because they are trying to automate a unique process for each unique file format. Making changes to each file format also would require significant changes to the entire process, which can be costly and time consuming.
- Opportunity to streamline provider processing.

Table 1 Time needed to create, update, and maintain XML Configuration files for FileParser

Building New File Process	**Minutes**
Initial File for Entity	5
Same File Same Address Multiple Specialties	1
Same File Additional address	2

Without a streamlined process to handle provider data, the tasks tend to be completed by someone maintaining the provider data manually. This is a very time-consuming activity and requires a full-time employee to keep the provider data updated.

Table 2: Processing time for the FileParser and DelegatedProviderProcessing tools using multiple production files to compare results

Run Time	Records	Time (seconds)
File Parser	693	4
	5761	15
	401	2
	19165	48

Run Time	Records	Time (minutes)
Data Cleansing/processing	8305	30
	119	0.5
	5491	23
	1663	5

Threats

Threats to Provider Processing are limited, given the cost and the amount of work that is required to continue maintaining and creating new DTSX files to handle the many provider files. Other options like different file parsing solutions still requires that an import application or process be used to get the data into the internal database once it is parsed and transformed. Below are some commercial file parsers that are available to replace the FileParser command line executable.

DataParse [5]

Strengths

- Run from Task Scheduler or Batch file on Business and Enterprise versions
- Able to handle multiple formats

Weaknesses

- Custom script language for parsing
- Full featured versions are costly
    - Business Version
        - $499 per user
        - $109 per year for updates
    - Enterprise Version
        - $1995 per user
        - $399 per year for updates

Template-Parser [6]

Strengths

- Supports multiple formats
- Lowest cost of commercial parsers
    - $17.50 per user

Weaknesses

- Appears unable to execute from script
- User Interface only
- Custom script language for parsing

ParseRat [7]

Strengths

- Supports multiple file formats

Weaknesses

- Dated programming language (based on User Interface early .Net or Visual Basic 6.0)
- Requires licensing per user per machine
- $49.95 per user per machine
- Cannot be executed from script

The reason why Provider Processing is considered different than the other data parsers and transformation tools is that the file parsing and transformation step, which would be the area to change, is able to be done systematically, without user interaction, and is able to use a standardized language to determine the file format. While the other options listed above can perform the same tasks that that FileParser does, they each have a monetary cost associated not only in the purchasing the tool, but in the time needed to learn the custom file parsing script languages.

Primary Audience

The primary audience for Provider Processing is health insurance organizations that require a means for efficiently processing provider servicing information. These organizations stand to benefit most from more efficient processing of data and improved data accuracy.

Secondary Uses

The FileParser command line executable can be migrated to other domains of business that focus on consuming flat files and generating a common output file to meet requirements.

Other areas in the health insurance domain where FileParser can help are in performing predictive analysis with claim information, results of patient medical appointments, and lab results since they can be represented in a flat file, which is common practice when presenting medical data. Since the format of data from different claims vendors, including vision, dental and medical, using FileParser can help with analysis of the data by allowing someone to look at the data in the same structure. In this case, a data analyst would look at all of the claim data files, determine the key pieces of information and use those key data points to create an XML configuration file. After the configuration file is built, the claim data from the different sources can have different analysis methods applied to determine information that could be used to help with preventative actions.

Another domain where FileParser can prove useful is in mining data on purchases from a variety of stores. Since each store probably has its own Point of Sale (POS) system, the purchase data output could look different, but still contain the same key pieces of information including date purchased, price, location, other items purchased, and anything else that might help determine potential buying habits and items that tend to be purchased together. Determining the key data points provides a base for the XML configuration needed to perform any required analysis independent of the source file format.

Evaluation of Provider Processing

Provider processing has provided a strong foundation for building a system that can parse and transform data, perform data cleansing and normalization, and process provider data. The foundation solves the key issue with the previous SSIS solution – increased costs associated with a need to make significant manual changes to the DTSX files when a file format changes or needs to be added. By using Provider Processing and a corresponding file for each medical provider input file, changes can be made to match new provider formats quickly, without the need to republish or compile any new code. This is one of the biggest benefits of using the FileParser to manage and generate a provider format that can be processed by a single application/process like DelegatedProviderProcessing.

Benefits and Outcomes

Immediate Benefits

By using Provider Processing, a health insurance organization has been able to realize immediate benefits upon implementing Provider Processing

- Accuracy in provider data
- Ensuring that data is entered in a timely manner
- Reduction in overall workforce needed to process the data
- Better provider data archival processes
- Better maintenance of provider data submitted for processing

The immediate benefits can affect the health insurance organization not only by reducing costs, but also by allowing the workforce to focus on both member-centric and company-centric

tasks. This can improve the health and wellbeing of the health insurance organization's members and also the wellbeing of the company as a whole.

The benefits apply not only to the insurance company. Medical providers also receive benefit because by ensuring that the provider data is accurate, the health insurance organization can ensure that their patients have a good customer experience. An example of this is if a medical provider decides to relocate their practice to a different location. If they do not notify the health insurance organization about the move, the data for the provider is out of date and can end up causing a member to go to a location that is no longer valid. The negative outcomes from a scenario like this range from an annoyed customer to a customer who decides that they will no longer see that medical provider. If a health insurance organization has the updated information, they can relay that information to the member and ensure that the member is able to find the new location, which tends to lead to a more positive impression of not only the insurance organization, but also of the medical provider.

From a business perspective, perhaps the strongest immediate benefit is the cost savings derived from taking something that was manually performed and changing the process to make it automatic. Even though some manual intervention is still required to process a provider's file, the amount of manual intervention is reduced to validating that the output file is correct and creating new XML configuration files. As an example, it can take a single worker an average of five minutes to enter and update a provider using the current manual interface. Most of the provider network files that are received can range from updates and additions of 100 provider records for a small provider file to over 5,000 provider records. The worker will not have to update all of the records in the files, because some of the provider records will not have changed. Even if there are only 48 provider additions, updates, or terminations, almost half of an employee's day will be used to perform updates (48 providers time 5 minutes/provider). A systematic process can apply a standard set of business rules to the data and consume the data

much faster. To see an execution sample that is taken from different production runs, please refer to Table 2 in the SWOT Opportunities section.

REFERENCES

1. Paez, I. & Jain, N. & Krugman, B. (2010) GasDay FileParser. Unpublished, Marquette University, Milwauke, WI
2. Wisconsin Statewide Health Information Network (WISHIN) www.wishin.org
3. Provider Directories. (n.d.). Retrieved March 31, 2015, from http://wishin.org/Products/WISHINDirect/ProviderDirectories.aspx
4. .DTSX File Extension. (2011, May 5). Retrieved March 31, 2015, from http://fileinfo.com/extension/dtsx
5. "Data Parse - Home." *Data Parse - Home*. Web. 31 Mar. 2015. https://www.dataparse.com/
6. "Data Extraction Software | Retrieve Transform Data from Text File, Web, Email, PDF, Excel." *Template-Parser.com*. Web. 31 Mar. 2015. http://www.template-parser.com/
7. "Convert Legacy Data Files Using ParseRat Data, Database and File Parser, Converter & Restructurer from Guy Software." *Convert Legacy Data Files Using ParseRat Data, Database and File Parser, Converter & Restructurer from Guy Software*. Web. 31 Mar. 2015. http://www.guysoftware.com/parserat.htm

APPENDIX

1.1 XML File Attribute Description

Text representation of the XML Configuration File

- **Source File Attributes**
  - Path
  - Delimiter
  - Transpose
- **Target File Attributes**
  - Path
  - Delimiter
  - Remove Text Qualifiers
  - Append File
- **Log File Attributes**
  - Path
- **Field Attributes**
  - In Name
  - Out Name
  - Out Position
  - Type
  - Length
  - Format
  - Low Bound
  - High Bound
  - Ok Null

- Default Value
- Column Optional
- Include Optional Column

1.2 Defined Provider Object Flat File Output

Text representation of the required columns generated by FileParser

- **Practitioners (Medical Personnel)**
  - Last Name
  - First Name
  - Middle Initial
  - Gender
  - Language
  - Degree
  - NPI
  - DEA
  - UPIN
  - State License
  - Medicaid License
  - Tax ID
  - Office Name
  - Office Address Line 1
  - Office Address Line 2
  - Office City
  - Office State
  - Office Zip
  - Office Phone

        - Office Fax
        - Accepting New Patients
        - Primary Location
        - Include In Directory
        - Expertise
        - Taxonomy
        - Specialty (State)
        - Billing Name
        - Billing Address Line 1
        - Billing Address Line 2
        - Billing City
        - Billing State
        - Billing Zip
        - Billing Phone
        - Billing Fax
        - Input File
        - Group NPI (For Files with Facilities and Medical Personnel combined)
    - **Facilities**
        - Full Name
        - NPI
        - DEA
        - UPIN
        - State License
        - Medicaid
        - Tax ID
        - Office Name

- Office Address Line 1
- Office Address Line 2
- Office City
- Office State
- Office Zip
- Office Phone
- Office Fax
- Accepting New Patients
- Primary Location
- Include In Directory
- Expertise
- Taxonomy
- Specialty (State)
- Billing Name
- Billing Address Line 1
- Billing Address Line 2
- Billing City
- Billing State
- Billing Zip
- Billing Phone
- Billing Fax
- Input File

1.3 XML Template

Sample of the XML Configuration file used by FileParser

```
<?xml version="1.0" encoding="utf-8" ?>
<ApplicationSettings>
  <SourceFile path="<SourcePath>" delimiter="," transpose="false"
/>
  <TargetFile path="<DestinationPath>" delimiter="\t"
removeTextQualifiers="false"/>
  <LogFile path="<LogPath>" />
  <HeaderIdentifier></HeaderIdentifier> <!--Regular Expression to
find Header-->
  <Fields fieldsIncludeHeader="true">
    <Field inName="<Column_InputName>" inPosition=""
outName="<Column_OutputName>"
outPosition="<LocationInOutputFile>"  type="<DataType>" length=""
format="" lowBound="" highBound="" okNull="true" defaultValue=""
columnOptional="" includeOptionalColumn="" />
  </Fields>
</ApplicationSettings>
```

1.4 Powershell Scripting Example

Microsoft Powershell sample code to cycle through a specified folder and perform the FileParser parsing and transformation

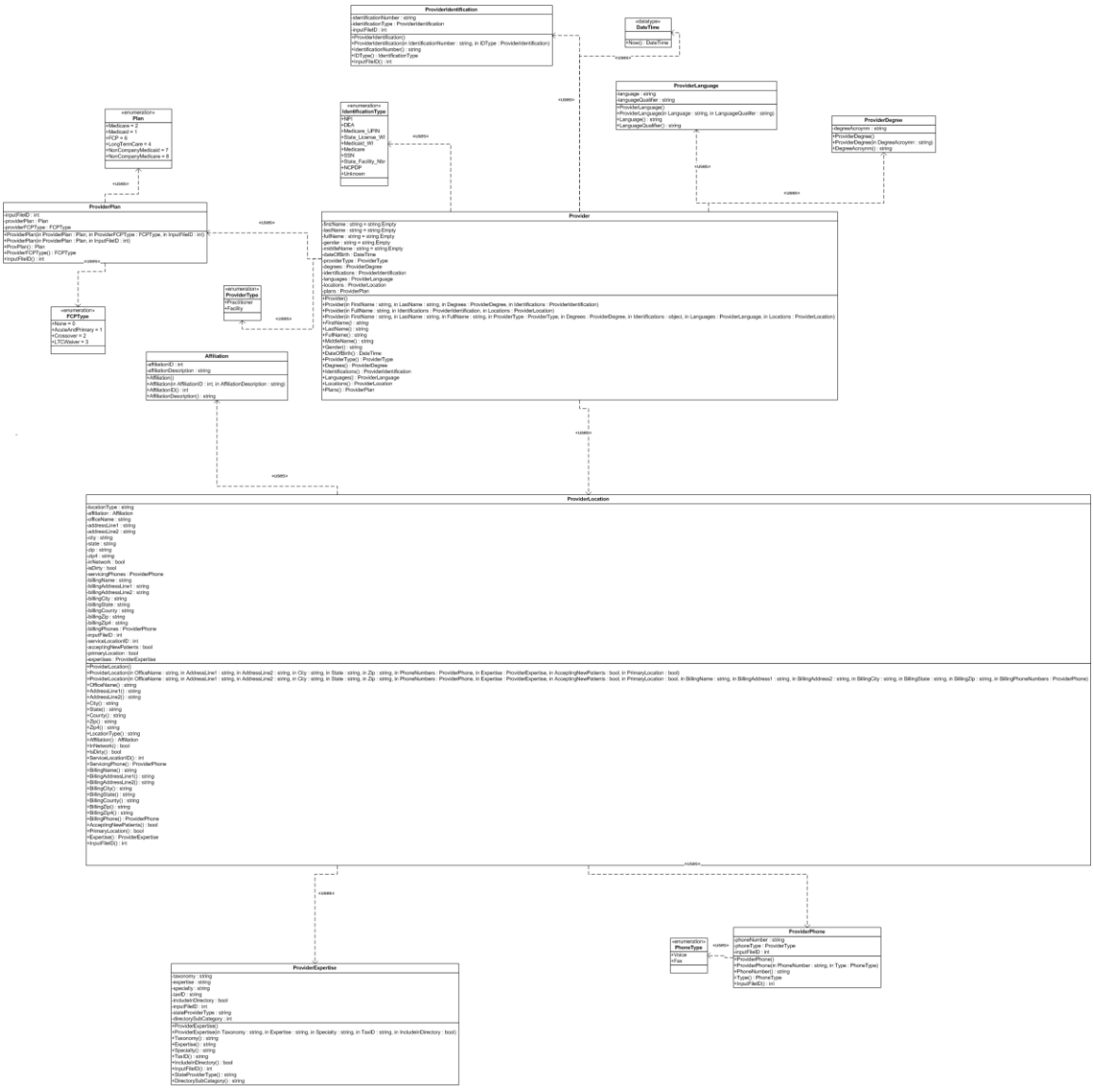
```
clear
. C:\JobLibrary\Scripts\Stub.ps1
Load-Assembly mscorlib
[string] $errorReportingAddress = "errorReporting@emaildomain"
function ProcessFiles([string] $FileLocation)
{
  Trap
  {
    [string] $exceptionMessage = $_.Exception.Message

    $emailBody = "Error Occured: " + $exceptionMessage
    Send-Mail -To $errorReportingAddress -Body $emailBody -
Subject "File Processing Error"

    break
  }

  [string] $filePath = ""
  [string] $archivePath = ""
  [bool] $validPath = $false

  [datetime] $currentDate = Get-Date

  $delegatedFiles = New-Object System.IO.DirectoryInfo
$FileLocation
```

```
  foreach($file in $delegatedFiles.GetFiles("*.xml"))
  {
    " "
    $file.Name

    $filePath = $FileLocation + "/" + $file.Name

    c:\JobLibrary\bin\DelegatedFeedParser.exe -file $filePath

  }
}
function ArchiveFiles([string] $FileLocation, [string]
$ArchiveLocation, [string] $FileType)
{
  [string] $filePath = ""
  [string] $archivePath = ""
  [bool] $validPath = $false

  [datetime] $currentDate = Get-Date


  $delegatedFiles = New-Object System.IO.DirectoryInfo
$FileLocation

  foreach($file in $delegatedFiles.GetFiles("*." + $FileType))
```

```
  {
    $file.Name

    $archivePath = $ArchiveLocation + "/" +
$file.Name.Replace("."+$FileType,"_" +
$currentDate.ToShortDateString().Replace("/","") +"." +$FileType)



    $validPath = Test-Path($archivePath)
    $validPath

    if($validPath -eq $false)
    {
      "MoveFile"
      $file.MoveTo($archivePath)
    }

  }
}


ProcessFiles -FileLocation "<XML Configuration Path>"
#If you want to archive the input files
ArchiveFiles -FileLocation "<Input File Location>" -
ArchiveLocation "<Archive File Location>"
```

1.5 Detailed Provider Class Object

Functions, attributes and other properties that make up the generic Provider Object



1.6 Detailed FileParser Class Diagram

Functions, attributes and other properties that make up the DLL used in FileParser