Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Computer Forensics: Automatización con Autopsy

Autor: Jorge Benítez Abad

Tutor: Manuel Sánchez Rubio

2018

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Computer Forensics: Automatización con Autopsy

Autor: Jorge Benítez Abad

Tutor: Manuel Sánchez Rubio

Tribunal:

Presidente:

Vocal 1º:

Vocal 2º:

Fecha:

Calificación:.....

A mis padres, por todo lo que han hecho por mi…

AGRADECIMIENTOS

En primer lugar, le quiero agradecer a mi tutor Manuel Sánchez Rubio por mostrarme el gran mundo que es la Informática Forense. También, quisiera agradecerle a Juan Manuel Martínez Alcalá por su apoyo en la realización de este proyecto.

A mi compañero y amigo Roberto Cabrera, por su magnífica colaboración y apoyo en la realización de este proyecto y por su amistad todos estos años que hemos pasado juntos.

También, quisiera agradecer su apoyo incondicional a los amigos que he llegado a conocer a lo largo de esta etapa, en especial, a los que han estado a mi lado casi a diario, desde el principio, y a los que no pudieron estar a mi lado todo lo que nos hubiera gustado, pero que siempre estuvieron ahí, sin los cuales, no habría sido lo mismo.

Por último, agradecer a mi familia. A mis padres, Jose Luis y Sonsoles, que de no ser por ellos, nada de esto habría sido posible y no habría llegado tan lejos, y a mi hermano Jose Luis y su novia Tamara, por su ánimo y apoyo durante todos estos años.

ÍNDICE RESUMIDO

ÍNDICE DETALLADO

ÍNDICE DE FIGURAS

RESUMEN

La Informática Forense está destinada a la extracción de archivos de una imagen computacional o evidencia. El procedimiento más eficaz, hasta el momento, es el análisis de la memoria RAM para conseguir obtener la actividad reciente de un determinado usuario en un equipo que se desea analizar. Pero gracias a una nueva generación de Software y técnicas, se está consiguiendo un análisis más exhaustivo de los equipos informáticos. Además, con la inserción de nuevos complementos a dichos Software, se consigue un análisis mucho más completo.

Este tipo de técnicas están destinadas tanto para casos civiles como para casos delictivos.

SUMMARY

Computer Forensics is designed to extract files from a computer image or evidence. The most effective procedure so far, is the analysis of RAM to get the recent activity of a certain user on a computer that you want to analyze. But thanks to a new generation of Software and techniques, a more exhaustive analysis of the computer equipment is being achieved. In addition, with the insertion of new add-ons to said Software, a much more complete analysis is achieved.

These types of techniques are intended for both civil cases and criminal cases.

PALABRAS CLAVE

Autopsy

Evidencia

Plugin

Postmortem

RESUMEN EXTENDIDO

El pilar fundamental de análisis forense de equipos informáticos es la extracción de toda la información necesaria de una evidencia para la elaboración de un informe acorde a las necesidades para las que se ha realizado dicho análisis.

Existen dos casos en los que se pueden utilizar técnicas de informática forense para analizar una evidencia: casos civiles y casos delictivos.

- En los casos civiles, estas técnicas están orientadas a la recuperación de archivos, por ejemplo, un ordenador de una empresa sufre una avería y no hay manera de solucionar dicha avería sin hacer desaparecer una serie de archivos que contiene dicho equipo y que son de vital importancia para la empresa. Con estas técnicas, sería posible analizar el disco duro del equipo y recuperar todos los archivos que se deseen (fotos, documentos de texto, archivos de Microsoft Office, etc) antes de proceder a la restauración del equipo. Además, sería posible recuperar archivos eliminadospor accidente o deliberadamente.
- En los casos delictivos, se analiza el equipo adquirido mediante técnicas de informática forense para intentar conseguir una serie de pruebas que ayuden a la investigación de un determinado delito. Estas pruebas pueden tratarse de actividad reciente realizada en el equipo (historial, descargas, programas ejecutados, etc) o, tanto archivos existentes en el equipo como archivos que se han intentado ocultar o eliminar de forma deliberada (fotos incriminatorias, recibos bancarios, facturas, etc) y que, gracias a estas técnicas, es posible su recuperación.

De un tiempo a esta parte, en los casos delictivos se analizaba la memoria RAM de los equipos de los sospechosos, para intentar conseguir la actividad reciente de dicho equipo. Esto suponía una gran cantidad de complicaciones, ya que se debía apagar el equipo extrayendo directamente la fuente de alimentación, para no borrar los datos de la RAM y transportarlo rápidamente para su análisis.

En la actualidad y con la aparición de nuevo Software y nuevas técnicas, ya no se realiza el análisis solo sobre la RAM, sino que se analiza todo el equipo. Esto facilitó mucho el análisis de la RAM, ya que esta, actualmente, guarda una serie de procesos volátiles en la memoria interna del equipo, y gracias a estas nuevas técnicas, se puede reconstruir la actividad reciente de la RAM sin necesidad de analizar esta. En el caso que sólo se disponga de la memoria RAM para su análisis, estas herramientas también podrían encargarse de realizarlo.

No sólo aparecieró nuevo Software y nuevas técnicas de análisis, además, estos nuevos programas tienen la capacidad de integrar complementos externos al código fuente del mismo, facilitando y agilizando notablemente su utilización. Estos complementos pueden estar desarrollados por los propios diseñadores de la herramienta o por terceros, usuarios que gracias a este tipo de Software y a una necesidad, aumentan poco a poco las funcionalidades de dichas herramientas. Por ejemplo, gracias a estos complementos, es posible integrar bases de datos de HASH conocidas para poder realizar un barrido automático sobre una evidencia para comprobar si la HASH de algún archivo en el interior de la evidencia concuerda con alguna HASH alojada en la base de datos.

Este tipo de análisis se realizan sobre evidencias en estado “Postmortem”, denominación relacionada con imágenes adquiridad o datos en estático. El análisis de este tipo de evidencias son mucho más complejos que el análisis de evidencias “en vivo”, ya que “en vivo” sólo sería necesario navegar por el equipo, mientras que en “Postmortem” es necesario montar la evidencia en unas herramientas destinadas a este propósito (por ejemplo FTK Imager o Autopsy). La diferencia es que, gracias a lo mencionado anteriormente, herramientas como Autopsy pueden automatizar el análisis de la evidencia, ahorrando mucho tiempo al análisis y reduciendo el posible “error humano” a la hora de realizar dicho análisis.

En el presente proyecto, se profundiza significativamente en el campo de la criptografía y de la infonrmática forense, para conseguir adquirir los conocimientos necesarios para la planificación, desarrollo y comprensión de este proyecto. Además, se presentan las pautas necesarias para realizar el desarrollo de un complemento funcional destinado a Autopsy. Se proporcionarán una serie de procedimientos para conseguir la realización de un análisis forense con Autopsy.

Finalmente, se presentará un caso práctico, como ejemplo de situación en la que se puede dar la necesidad de realizar un análisis forense. Aunque, gracias a los complementos desarrollados, la necesidad de utilizar los presentes procedimientos, abarcarían una gran cantidad de tipos de casos, tanto civiles como delictivos, y una inmensa cantidad de situaciones reales.

1. Introducción

En la actualidad, se realizan estudios forenses de evidencias analizando la memoria RAM de los equipos. Para ello se deben cumplir una serie de requisitos:

- No apagar el equipo como se haría normalmente. Se debe adquirir el equipo desconectándolo directamente de la fuente de alimentación (enchufe o batería). De este modo se consigue mantener la información que se ha almacenado en la memoria RAM intacta durante cierto tiempo.
- Se debe transportar la memoria RAM de inmediato para su análisis, debido a que se trata de una memoria volátil y la información se irá borrando poco a poco. Es recomendable, que su transporte se realice a temperaturas muy bajas (-60º ó inferior), de esta forma, la información almacenada tardará más tiempo en desaparecer.

Poco a poco, van apareciendo nuevas herramientas que facilitan el análisis forense, debido a que no es necesario analizar la memoria RAM de un equipo, sino que, dichas herramientas, realizan el análisis sobre la CPU.

Además de estas nuevas herramientas, se van añadiendo nuevos complementos para mejorar y/o conseguir análisis más completos.

2. Objetivo

El objetivo de este Trabajo Fin de Grado se basa en el estudio y la implementación de Plugins que puedan ser utilizados por Autopsy para conseguir realizar un análisis forense completo sobre una evidencia.

Los principales objetivos que se pretenden alcanzar son:

- Estudio de la Criptografía y de la Informática Forense.
- Estudio de las herramientas forenses existentes, en especial, Autopsy.
- Creación de una evidencia a partir de la herramienta FTK Imager.
- Estudio y desarrollo de complementos o Plugins para su implantación en Autopsy.
- Realización de un análisis forense.

3. Alcance

Este proyecto tiene como finalidad la realización de un análisis forense sobre una evidencia, extrayendo información necesaria, que sería interesante para una auditoría forense (documentación, imágenes, vídeos, historial de búsqueda, aplicaciones instaladas, archivos, etc).

El presente proyecto está claramente dividido en dos partes:

- Teórico: estudio de la criptografía y la informática forense, estudio de herramientas destinadas a la informática forense y estudio del desarrollo de complementos.
- Práctico: creación de una evidencia con FTK Imager, implantación y ejecución de Plugins en Autopsy, y extracción de información o archivos de una evidencia utilizando los Plugins implantados en Autopsy.

Se pretendía, la implantación de un Plugin extra, destinado a la desencriptación de contraseñas cacheadas con cifrado DPAPI (Título: “DPAPI Forensics: estudio y pruebas con Autopsy”, Autor: Roberto Cabrera Díaz). Pero, debido a una serie de incompatibilidades, no ha sido posible su integracción en Autopsy.

4. Criptografía

4.1. Introducción a la criptografía

La criptografía se ha definido, tradicionalmente, como el ámbito de la criptología que se ocupa de las técnicas de cifrado o codificado destinadas a alterar las representaciones lingüísticas de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados. Estas técnicas se utilizan tanto en el arte como en la ciencia y en la tecnología. Por tanto, el único objetivo de la criptografía era conseguir la confidencialidad de los mensajes, para lo cual se diseñaban sistemas de cifrado y códigos, y la única criptografía existente era la llamada criptografía clásica.

La aparición de la informática y el uso masivo de las comunicaciones digitales, han producido un número creciente de problemas de seguridad. Las transacciones que se realizan a través de la red pueden ser interceptadas, y, por tanto, la seguridad de esta información debe garantizarse. Este desafío ha generalizado los objetivos de la criptografía para ser la parte de la criptología que se encarga del estudio de los algoritmos, protocolos (se les llama protocolos criptográficos), y sistemas que se utilizan para proteger la información y dotar de seguridad a las comunicaciones y a las entidades que se comunican.

Para ello los criptógrafos investigan, desarrollan y aprovechan técnicas matemáticas que les sirven como herramientas para conseguir sus objetivos. Los grandes avances producidos en el mundo de la criptografía han sido posibles gracias a la evolución que se ha producido en el campo de la matemática y la informática.

4.1.1. Objetivos de la criptografía

La criptografía actualmente se encarga del estudio de los algoritmos, protocolos y sistemas que se utilizan para dotar de seguridad a las comunicaciones, a la información y a las entidades que se comunican. El objetivo de la criptografía es diseñar, implementar, implantar, y hacer uso de sistemas criptográficos para dotar de alguna forma de seguridad. Por tanto, el tipo de propiedades de las que se ocupa la criptografía son, por ejemplo:

- Confidencialidad: garantiza que la información sea accesible únicamente a personal autorizado. Para conseguirlo utiliza códigos y técnicas de cifrado.
- Integridad: garantiza la corrección y completitud de la información. Para conseguirlo puede usar por ejemplo funciones hash criptográficas MDC, protocolos de compromiso de bit, o protocolos de notarización electrónica.
- Vinculación: permite vincular un documento o transacción a una persona o un sistema de gestión criptográfico automatizado. Cuando se trata de una persona, se trata de asegurar su conformidad respecto a esta vinculación (content commitment) de forma que pueda entenderse que la vinculación gestionada incluye el entendimiento de sus implicaciones por la persona. Antiguamente se utilizaba el término "No repudio" que está abandonándose, ya que implica conceptos jurídicos que la tecnología por sí sola no puede resolver. En relación con dicho término se entendía que se proporcionaba protección frente a que alguna de las entidades implicadas en la comunicación, para que no pudiera negar haber participado en toda o parte de la comunicación. Para conseguirlo se puede usar por ejemplo firma digital. En algunos contextos lo que se intenta es justo lo contrario: Poder negar que se ha intervenido en la comunicación. Por ejemplo, cuando se usa un servicio de mensajería instantánea y no queremos que se pueda demostrar esa comunicación. Para ello se usan técnicas como el cifrado negable.

- Autenticación: proporciona mecanismos que permiten verificar la identidad del comunicador. Para conseguirlo puede usar por ejemplo función hash criptográfica MAC o protocolo de conocimiento cero.
- Soluciones a problemas de la falta de simultaneidad en la telefirma digital de contratos. Para conseguirlo puede usar por ejemplo protocolos de transferencia inconsciente.

Un sistema criptográfico es seguro respecto a una tarea si un adversario con capacidades especiales no puede romper esa seguridad, es decir, el atacante no puede realizar esa tarea específica.

4.1.2. Formas de romper la seguridad criptográfica

Existen tres formas de romper la seguridad criptográfica de los equipos o evidencias:

- Atacar la criptografía subyacente. Es lo que sería un ataque teórico a los mecanismos criptográficos usados.
- Atacar la implementación concreta. La criptografía puede ser implementada en software o en hardware. Es bastante probable que las implementaciones concretas tengan vulnerabilidades que se pueden aprovechar. También las vulnerabilidades se podrían introducir de forma deliberada y de esta forma proporcionar puertas traseras disponibles para ser utilizadas.
- Atacar el lado humano. Muchas veces en los sistemas criptográficas hay personas o entidades que tienen privilegios especiales. Presionando a estas personas o entidades para que nos den acceso a recursos o a información privilegiada, podríamos vulnerar la seguridad del sistema.

El usuario interesado en romper la seguridad criptográfica no realiza únicamente una de las tres formar de romper dicha seguridad. Sopesa las tres opciones y está preparado para realizar cualquiera de ellas si bien lo cree necesario o si bien está obligado a realizar cualquier otra opción porque la opción inicial ha fallado.

4.2. Tipos de criptografía

Existen varios tipos de criptografía, que han ido evolucionando a lo largo de la historia.

4.2.1. Criptografía clásica

La criptografía clásica abarca desde tiempos inmemoriales hasta la mitad del siglo XX. El punto de inflexión en esta clasificación la marcan tres hechos relevantes:

- En el año 1948 se publica el estudio de Claude Shannon sobre la Teoría de la Información.
- En 1974 aparece el estándar de cifra DES.
- En el año 1976 se publica el estudio realizado por W. Diffie y M. Hellman sobre la aplicación de funciones matemáticas de un solo sentido a un modelo de cifra, denominado cifrado con clave pública.

Existen varios tipos de cifrado clásico:

- Transposición: consiste en crear el texto cifrado simplemente desordenando las unidades que forman el texto original; los algoritmos de transposición reordenan las letras, pero no las disfrazan. Ejemplo: TU SECRETO ES TU PRISIONERO; SI LO SUELTAS, TU ERES SU PRISIONERO.

Imagen - 1: Transposición

- Sustitución: consiste en sustituir las unidades del texto original por otras; Los algoritmos de sustitución y los códigos, preservan el orden de los símbolos en claro, pero los disfrazan. Ejemplo: ENCONTREMONOS A MEDIANOCHE.

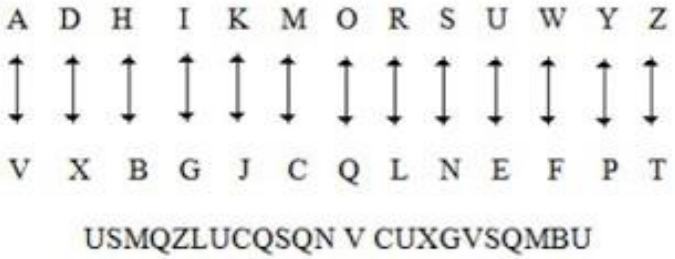

Imagen - 2: Sustitución 1

Otra práctica era la de sustituir los caracteres del mensaje por parejas de letras o parejas de números. Ejemplo: QUE BUENA IDEA LA DEL GRIEGO.

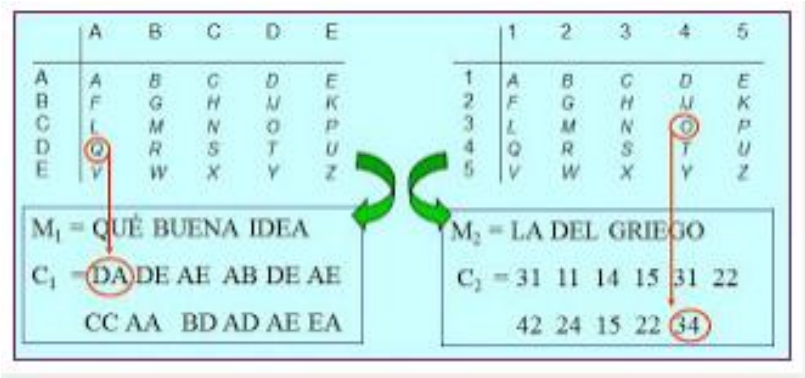

Imagen - 3: Sustitución 2

- El cifrado del César: en el siglo I a.C., Julio César usa este cifrado, cuyo algoritmo consiste en el desplazamiento de tres espacios hacia la derecha de los caracteres del texto en claro. Es un cifrado por sustitución monoalfabético en el que las operaciones se realizan módulo n, siendo n el número de elementos del alfabeto (en aquel entonces latín).

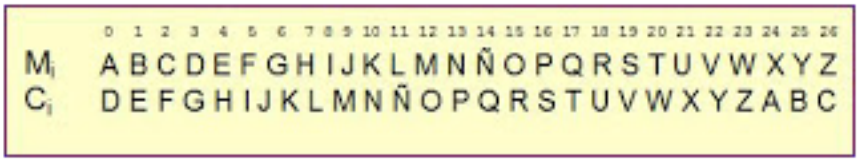

Imagen - 4: Cifrado del César

Cada letra se cifrará siempre igual. Es una gran debilidad y hace que este sistema sea muy vulnerable y fácil de atacar simplemente usando las estadísticas del lenguaje.

- El cifrado Pigpen: la cifra de sustitución monoalfabética perduró a través de los siglos en formas diversas. Por ejemplo, la cifra de los templarios: El Temple era una orden de monjes fundada en el siglo XII., cuya misión principal era asegurar la seguridad de los peregrinos en Tierra Santa. Rápidamente, los templarios se desentendieron de este objetivo, y se enriquecieron considerablemente hasta el punto de llegar a ser tesoreros del rey y del Papa. Para codificar las letras de crédito que ellos intercambiaban, ellos remplazaban cada letra por un símbolo, siguiendo la sustitución siguiente:

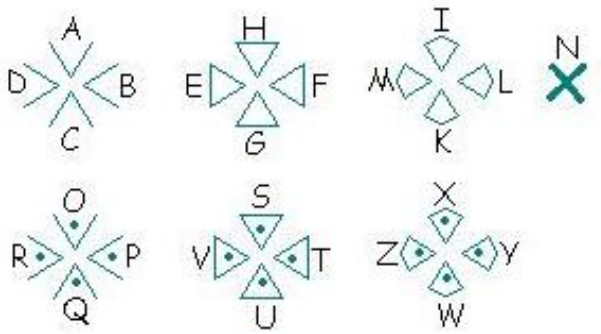

Imagen - 5: Cifrado Pigpen 1

Otro ejemplo es la cifra Pigpen fue utilizada por los masones en el siglo XVIII para preservar la privacidad de sus archivos, y todavía la usan los niños hoy en día. La cifra no sustituye una letra por otra, sino que sustituye cada letra por un símbolo de acuerdo al siguiente modelo:

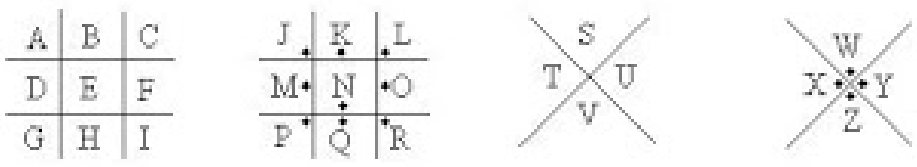

Imagen - 6: Cifrado Pigpen 2

4.2.2. Criptografía simétrica

También conocida como criptografía de clave secreta, se incluye en esta familia el conjunto de algoritmos diseñados para cifrar un mensaje utilizando una única clave conocida por los dos interlocutores, de manera que el documento cifrado sólo pueda descifrarse conociendo dicha clave secreta. Algunas de las características más destacadas de este tipo de algoritmos son las siguientes:

- A partir del mensaje cifrado no se puede obtener el mensaje original ni la clave que se ha utilizado, aunque se conozcan todos los detalles del algoritmo criptográfico utilizado.
- Se utiliza la misma clave para cifrar el mensaje original que para descifrar el mensaje codificado.
- Emisor y receptor deben haber acordado una clave común por medio de un canal de comunicación confidencial antes de poder intercambiar información confidencial por un canal de comunicación inseguro.

Los algoritmos simétricos más conocidos son: DES, 3DES, RC2, RC4, RC5, IDEA, Blowfish y AES.

El algoritmo DES, basado en Lucifer de IBM (1975), fue seleccionado como algoritmo estándar de cifrado en 1977 por NIST (National Institute of Standards and Technology, USA). Utiliza claves de cifrado bastante cortas (56 bits, de los cuales sólo se utilizan 48 bits) y hoy en día se considera poco robusto, sobre todo desde que en 1998 la Electronica Frontier Foundation hizo público un crackeador de código DES capaz de descifrar mensajes DES en menos de 3 días.

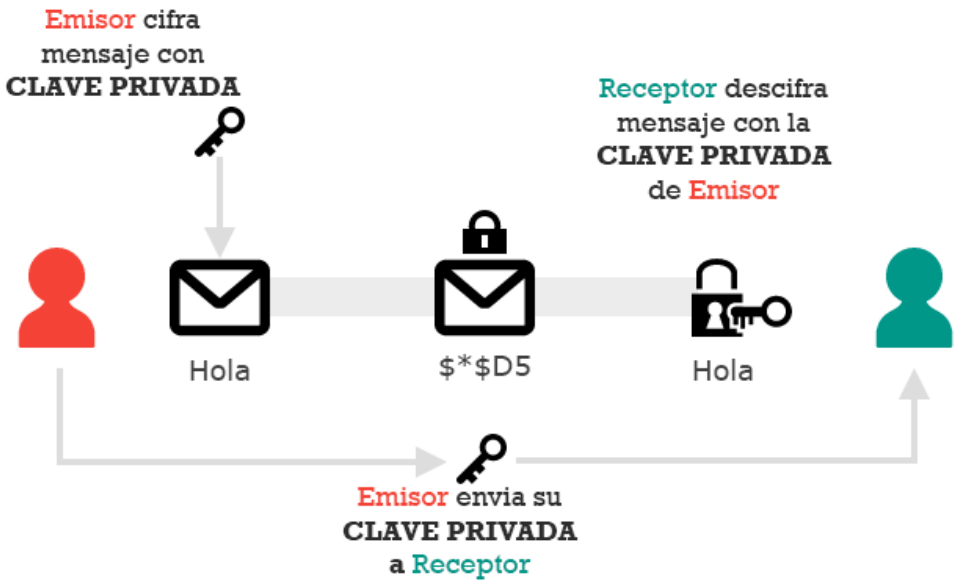

Imagen - 7: Criptografía Simétrica

4.2.3. Criptografía asimétrica

Esta categoría incluye un conjunto de algoritmos criptográficos que utilizan dos claves distintas para cifrar y para descifrar el mensaje. Ambas claves tienen una relación matemática entre sí, pero la seguridad de esta técnica se basa en que el conocimiento de una de las claves no permite descubrir cuál es la otra clave. En realidad, sería necesario conocer todos los números primos grandes para ser capaz de deducir una clave a partir de otra, pero está demostrado que en la práctica se tardarían demasiados años sólo en el proceso de obtención de los números primos grandes.

Cada usuario cuenta con una pareja de claves, una la mantiene en secreto y se denomina clave privada y otra la distribuye libremente y se denomina clave pública. Para enviar un mensaje confidencial sólo hace falta conocer la clave pública del destinatario y cifrar el mensaje utilizando dicha clave. En este caso los algoritmos asimétricos garantizan que el mensaje original sólo puede volver a recuperarse utilizando la clave privada del destinatario. Dado que la clave privada se mantiene en secreto, sólo el destinatario podrá descifrar el mensaje.

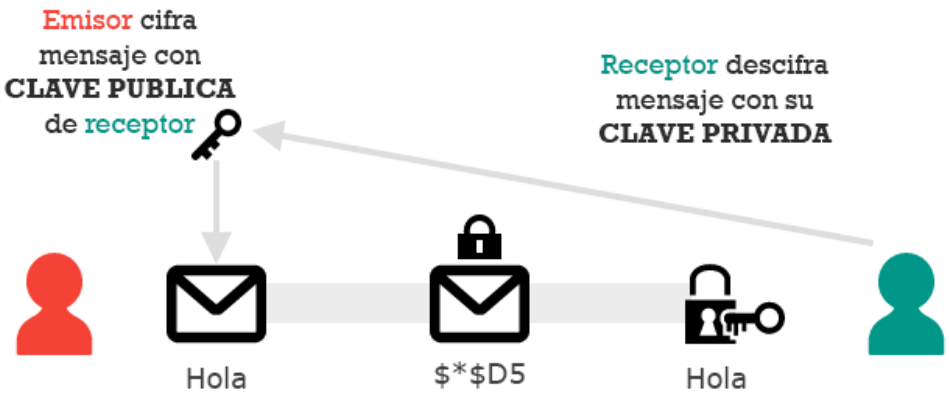

Imagen - 8: Criptografía Asimétrica

Algunas de las características más destacadas de este tipo de algoritmos son las siguientes:

- Se utilizan una pareja de claves denominadas clave pública y clave privada, pero a partir de la clave pública no es posible descubrir la clave privada.
- A partir del mensaje cifrado no se puede obtener el mensaje original, aunque se conozcan todos los detalles del algoritmo criptográfico utilizado y aunque se conozca la clave pública utilizada para cifrarlo.
- Emisor y receptor no requieren establecer ningún acuerdo sobre la clave a utilizar. El emisor se limita a obtener una copia de la clave pública del receptor, lo cual se puede realizar, en principio, por cualquier medio de comunicación, aunque sea inseguro.

A diferencia de los algoritmos de clave secreta, que existen desde los tiempos de los romanos, los métodos asimétricos son muy recientes. En 1976, Whitfield Diffie y Martin Hellman crearon un método con la ayuda de Ralph Merkle para iniciar una comunicación segura sin haber acordado previamente una clave secreta. El método se conoce como Diffie-Hellman Key Exchange [13]. Poco más tarde se publicó el primer algoritmo asimétrico completo, denominado RSA, que sigue siendo el más utilizado en la actualidad.

4.3. Algoritmos HASH

Los algoritmos HASH, parten de una información de entrada de longitud indeterminada y obtienen como salida un código, que en cierto modo se puede considerar único para cada entrada. La función de estos algoritmos es determinista, es decir, que partiendo de una misma entrada siempre se obtiene la misma salida. Sin embargo, el interés de estos algoritmos reside en que partiendo de entradas distintas se obtienen salidas distintas.

Unos ejemplos muy sencillos, aunque muy vulnerables, son los dígitos de control y los CRC (Código de Redundancia Cíclica) que se utilizan para detectar errores de trascripción o de comunicación. Estos algoritmos en particular garantizan que el código generado cambia ante una mínima modificación de la entrada y tienen aplicaciones muy concretas de control de integridad en procesos con perturbaciones fortuitas y poco probables. Sin embargo, son poco robustos y está demostrado que se pueden realizar pequeños conjuntos de modificaciones en un documento de manera que el CRC resulte inalterado. En un buen algoritmo HASH es inadmisible que un conjunto reducido de modificaciones no altere el código resultante, ya que se podrían realizar retoques en el documento sin que fuesen detectados, y por lo tanto no se garantiza la integridad.

Dado que el tamaño del código que se genera como salida es de tamaño limitado, (normalmente 128, 256 ó 512 bits) mientras que el tamaño del documento de entrada es ilimitado (normalmente un archivo), es evidente que se cumplen dos propiedades:

- El algoritmo es irreversible, es decir, no es posible obtener el documento original a partir del código generado.
- Existen varios documentos que dan lugar a un mismo código.

La segunda propiedad es debida a que el número de combinaciones de los códigos de tamaño limitado es menor al número de combinaciones de cualquier archivo grande. Sin embargo, los buenos algoritmos consiguen que los documentos que dan lugar al mismo código sean completamente diferentes y por lo tanto sólo uno de ellos será legible. Los algoritmos más utilizados son MD5 y SHA1, pero nunca se utilizan códigos CRC en aplicaciones de seguridad.

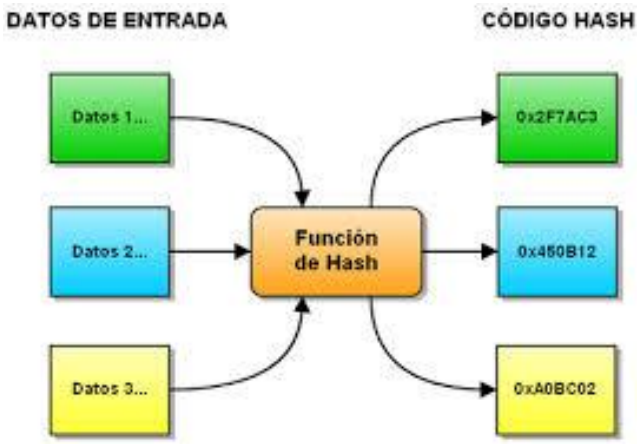

Imagen - 9: Algoritmos HASH

5. Informática forense

La informática forense es una rama de la ciencia forense digital perteneciente a la evidencia encontrada en computadores y medios de almacenamiento digital. El objetivo de la informática forense es examinar los medios digitales de manera forense con el objetivo de identificar, preservar, recuperar, analizar y presentar hechos y opiniones sobre la información digital.

Aunque a menudo se asocia con la investigación de una amplia variedad de delitos informáticos, los procedimientos forenses informáticos también pueden utilizarse en procesos civiles. La disciplina implica técnicas y principios similares a la recuperación de datos, pero con pautas y prácticas adicionales diseñadas para crear un seguimiento de auditoría legal.

La evidencia de las investigaciones forense del computador generalmente está sujeta a las mismas pautas y prácticas de otras pruebas digitales. Se ha utilizado en una serie de casos de alto perfil y está siendo ampliamente aceptado como confiable dentro de los sistemas judiciales estadounidenses y europeos.

5.1. Proceso forense

Las investigaciones forenses informáticas generalmente siguen el proceso forense digital estándar (adquisición, examen, análisis e informes). Las investigaciones se realizan en datos estáticos (es decir, imágenes adquiridas o "postmortem") en lugar de sistemas "en vivo". Este es un cambio de las primeras prácticas forenses donde la falta de herramientas especializadas llevó a los investigadores a trabajar comúnmente con datos "en vivo".

5.1.1. Técnicas

Se utilizan varias técnicas durante las investigaciones forenses informáticas y se ha escrito mucho sobre las muchas técnicas utilizadas por las fuerzas del orden público en particular.

- Análisis de transmisión cruzada: una técnica forense que correlaciona la información que se encuentra en varios discos duros. El proceso, que todavía se está investigando, se puede utilizar para identificar redes sociales y realizar la detección de anomalías.
- Análisis en vivo: el examen de las computadoras desde el sistema operativo utilizando análisis forense personalizados o herramientas de administración de sistemas existentes para extraer pruebas. La práctica es útil cuando se trata de sistemas de cifrado de archivos, por ejemplo, donde se pueden recopilar las claves de cifrado y, en algunos casos, se puede generar una imagen del volumen del disco rígido lógico (conocido como adquisición en vivo) antes de apagar la computadora.
- Archivos eliminados: una técnica común utilizada en informática forense es la recuperación de archivos eliminados. El software forense moderno tiene sus propias herramientas para recuperar o esculpir datos eliminados. La mayoría de los sistemas operativos y sistemas de archivos no siempre borran los datos físicos de los archivos, lo que permite a los investigadores reconstruirlos desde los sectores del disco físico. El tallado de archivos implica buscar encabezados de archivos conocidos dentro de la imagen del disco y reconstruir materiales eliminados.
- Forense estocástico: un método que utiliza propiedades estocásticas del sistema informático para investigar actividades que carecen de artefactos digitales. Su principal uso es investigar el robo de datos.

- Esteganografía: una de las técnicas utilizadas para ocultar datos es a través de la esteganografía, el proceso de ocultar datos dentro de una evidencia o imagen digital. Un ejemplo sería ocultar imágenes delictivas u otra información que un delincuente determinado no desea que se descubra. Los profesionales en informática forense pueden luchar contra esto mirando el hash del archivo y comparándolo con la imagen original (si está disponible). Mientras que la imagen aparece exactamente igual, el hash cambia a medida que los datos cambian.

5.1.2. Datos volátiles

Al tomar pruebas, si la máquina todavía está activa, se puede perder toda la información almacenada únicamente en la memoria RAM que no se recupera antes de apagarla. Una aplicación de "análisis en vivo" es recuperar datos RAM (por ejemplo, utilizando la herramienta COFEE de Microsoft, windd, WindowsSCOPE) antes de eliminar una exposición. CaptureGUARD Gateway evita el inicio de sesión de Windows para equipos bloqueados, lo que permite el análisis y la adquisición de memoria física en una computadora bloqueada.

La memoria RAM se puede analizar en busca de contenido previo después de la pérdida de energía, porque la carga eléctrica almacenada en las celdas de memoria lleva tiempo para disiparse, un efecto explotado por el ataque de arranque en frío. El tiempo que los datos son recuperables se ve incrementado por bajas temperaturas y mayores voltajes de celda. Mantener la RAM sin alimentación por debajo de -60°C ayuda a preservar los datos residuales en un orden de magnitud, mejorando las posibilidades de una recuperación exitosa. Sin embargo, puede ser poco práctico hacer esto durante un examen de campo.

Sin embargo, algunas de las herramientas necesarias para extraer datos volátiles requieren que una computadora esté en un laboratorio forense, tanto para mantener una cadena de pruebas legítima como para facilitar el trabajo en la máquina. Si es necesario, la policía aplica técnicas para mover una computadora de escritorio en funcionamiento. Estos incluyen un jiggler de mouse, que mueve el mouse rápidamente en pequeños movimientos e impide que la computadora se vaya a dormir accidentalmente. Por lo general, una fuente de alimentación ininterrumpida (UPS) proporciona energía durante el tránsito.

Sin embargo, una de las maneras más fáciles de capturar datos es guardar realmente los datos de RAM en el disco. Varios sistemas de archivos que tienen características de diario como NTFS y ReiserFS mantienen una gran parte de los datos de RAM en el medio de almacenamiento principal durante la operación, y estos archivos de página se pueden volver a ensamblar para reconstruir lo que estaba en la RAM en ese momento.

5.2. Herramientas de análisis

Existen varias herramientas comerciales y de código abierto para la investigación forense informática. El análisis forense típico incluye una revisión manual del material en los medios, revisar el registro de Windows en busca de información sospechosa, descubrir y descifrar contraseñas, buscar palabras clave para temas relacionados con el delito y extraer el correo electrónico y las imágenes para su revisión.

5.2.1. Informática forense

NOMBRE	*PLATAFORMA*	*LICENCIA*	*VERSIÓN*	*DESCRIPCIÓN*
Autopsy	Windows, macOS, Linux	GPL	4.5	Una plataforma forense digital y GUI para The Sleuth Kit
COFEE	Windows	Proprietary	n/a	Un conjunto de herramientas para Windows desarrollado por Microsoft
Digital Forensics Framework	Unix-like Windows	GLP	1.3	Marco e interfaces de usuario dedicadas a Digital Forensics
EPRB	Windows	Proprietary	1435	Conjunto de herramientas para sistemas cifrados y descifrado de datos y recuperación de contraseñas
EnCase	Windows	Proprietary	8.6	Suite forense digital creada por Guidance Software
FTK	FTK	Proprietary	6.0.1	Herramienta multiusos, FTK es una plataforma de investigaciones digitales citada por la corte construida para velocidad, estabilidad y facilidad de uso.
ISEEK	Windows	Proprietary	1	Herramienta híbrida-forense que se ejecuta solo en la memoria: diseñada para grandes entornos de red
IsoBuster	Windows	Proprietary	4.1	Herramienta ligera esencial para inspeccionar cualquier tipo de soporte de datos, compatible con una amplia gama de sistemas de archivos, con funcionalidad de exportación avanzada.
Xiraf	n/a	Proprietary	n/a	Servicio informático forense en línea.
Open Computer Forensics Architecture	Linux	LGPL/GPL	2.3.0	Marco forense informático para el entorno CF-Lab
OSForensics	Windows	Proprietary	3.3	Herramienta forense multiusos
PTK Forensics	LAMP	Proprietary	2.0	GUI para The Sleuth Kit
SafeBack	n/a	Proprietary	3.0	Adquisición y respaldo de medios digitales (evidencia)

SANS Investigative Forensics Toolkit – SIFT	Ubuntu		2.1	Sistema operativo forense multiusos
The Coroner´s Toolkit	Unix-like	IBM Public License	1.19	Un conjunto de programas para el análisis de Unix
The Sleuth Kit	Unix-like Windows	IPL, CPL, GPL	4.1.2	Una biblioteca de herramientas para Unix y Windows
Windows To Go	n/a	Proprietary	n/a	Sistema operativo de arranque
Wireshark	Cross-platform	GPL	n/a	La biblioteca de backend / captura de paquetes de código abierto utilizada en [win] pcap.

5.2.2. Análisis forense de la memoria

Las herramientas de memoria forense se usan para adquirir y / o analizar la memoria volátil (RAM) de un equipo. A menudo se utilizan en situaciones de respuesta a incidentes para preservar las pruebas en la memoria que se perderían cuando se apaga el sistema, y para detectar rápidamente malware sigiloso, al examinar directamente el sistema operativo y otro software en ejecución en la memoria.

NOMBRE	*VENDEDOR*	*PLATAFORMA*	*LICENCIA*
Volatililty	Volatile Systems	Windows Linux	GPL
WindowsSCOPE	BlueRISC	Windows	Proprietary

5.2.3. Análisis forense de dispositivos móviles

Las herramientas forenses móviles tienden a consistir en un componente de hardware y software. Los teléfonos móviles vienen con una amplia gama de conectores, los dispositivos de hardware admiten una serie de cables diferentes y desempeñan el mismo papel que un bloqueador de escritura en dispositivos informáticos.

NOMBRE	*PLATAFORMA*	*LICENCIA*	*VERSIÓN*	*DESCRIPCIÓN*
MicroSystemation XRY/XACT	Windows	Proprietary		Paquete de hardware/software, se especializa en datos eliminados

5.2.4. Análisis forense de Software

El análisis forense del software es la ciencia de analizar el código fuente del software o el código binario para determinar si se produjo una infracción a la propiedad intelectual o un robo. Es la pieza central de juicios y acuerdos cuando las empresas están en disputa por cuestiones relacionadas con patentes de software, derechos de autor y secretos comerciales. Las herramientas de análisis forense de software pueden comparar el código para determinar la correlación, una medida que puede usarse para guiar a un experto en análisis forense de software.

5.2.5. Otros

NOMBRE	*PLATAFORMA*	*LICENCIA*	*VERSIÓN*	*DESCRIPCIÓN*
DECAF	Windows	GPL	n/a	Herramienta que ejecuta automáticamente un conjunto de acciones definidas por el usuario para detectar la herramienta COFEE de Microsoft
Evidence Eliminator	Windows	Proprietary	6.03	Software anti-forensics, confirma la eliminación de archivos de forma segura
HashKeeper	Windows	GPL	n/a	Aplicación de base de datos para almacenar firmas HASH de archivos

6. Autopsy

Autopsy es un software que simplifica el despliegue de muchos de los programas y complementos de código abierto que se usan en "The Sleuth Kit". La interfaz gráfica de usuario muestra los resultados de la búsqueda forense del volumen subyacente, lo que facilita a los investigadores marcar las secciones de datos pertinentes. La herramienta es mantenida en gran medida por "Basis Technology Corp", con la asistencia de programadores de la comunidad. La compañía vende servicios de soporte y capacitación para usar el producto.

La herramienta está diseñada teniendo en cuenta estos principios:

- Extensible: el usuario debe poder agregar nuevas funcionalidades mediante la creación de complementos que puedan analizar todo o parte de la fuente de datos subyacente.
- Frameworks: la herramienta ofrecerá algunos enfoques estándar para la ingesta de datos, analizándolos e informando los hallazgos para que los desarrolladores puedan seguir los mismos patrones de diseño cuando sea posible.
- Facilidad de uso: el buscador de Autopsy debe ofrecer los asistentes y las herramientas históricas para que los usuarios puedan repetir sus pasos sin demasiada reconfiguración.

El navegador principal se puede ampliar agregando módulos que ayudan a escanear los archivos (llamado "ingestión"), explorar los resultados (llamados "ver") o resumir los resultados (llamados "informes"). Una colección de módulos de código abierto permite la personalización.

La versión 2 de Autopsy está escrita en Perl y se ejecuta en todas las plataformas principales, incluidas Linux, Unix, macOS y Windows. Se basa en "The Sleuth Kit" para analizar el disco. La versión 2 se lanzó bajo "GNU GPL 2.0".

Autopsy 3.0 está escrito en Java usando la plataforma NetBeans . Se ejecuta solo en Windows en este momento y se publica bajo la licencia de Apache 2.0.

Autopsy depende de varias bibliotecas con varias licencias.

6.1. Requisitos previos

Antes de comenzar el análisis forense de la evidencia, debemos tener en cuenta que Autopsy trabaja con evidencias o imágenes de sistema en formatos con extensión .dd (ó .001, .002, .003, etc).

A continuación, se muestra el procedimiento a seguir para que, a partir de una evidencia en formato .vmdk, obtener una evidencia en el formato deseado, utilizando FTK Imager:

1. Arrancamos FTK Imagen y cargamos la imagen en formato .vmdk que deseamos cambiar de formato.
2. Una vez esté cargada la imagen, vamos al árbol de nuestra evidencia. En él, podemos ver que aparecen dos particiones en la evidencia. No es estrictamente necesario elegir ambas particiones, sería suficiente con la segunda partición. Aunque sería recomendable elegir ambas particiones para conseguir realizar un análisis forense más completo.

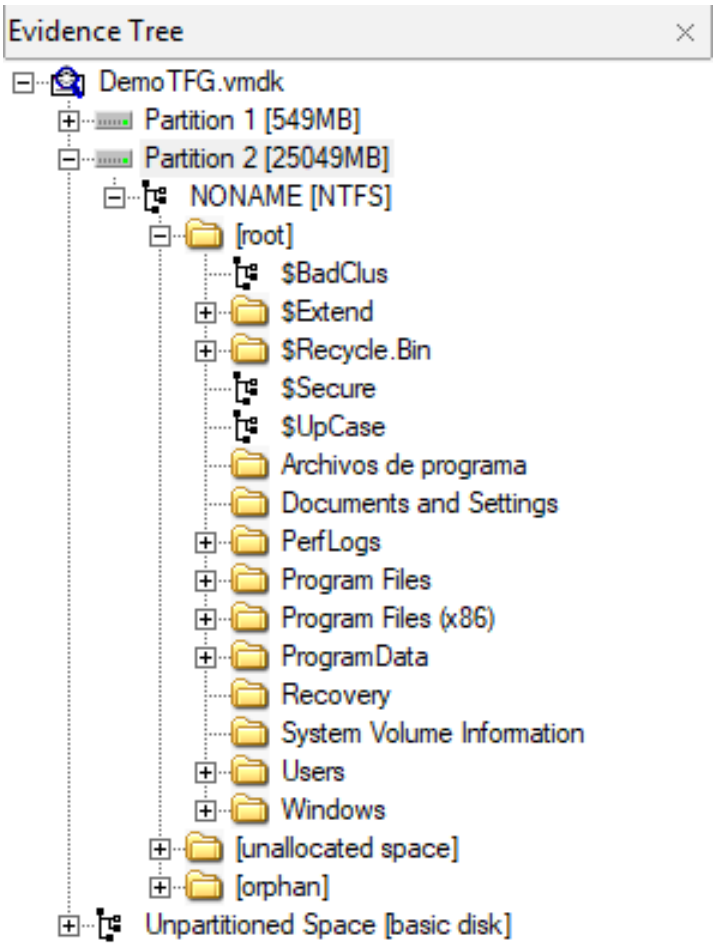

Imagen - 10: Árbol evidencia FTK Imager

3. Hacemos clic derecho sobre la segunda partición y pulsamos en “Export Disk Image…”.

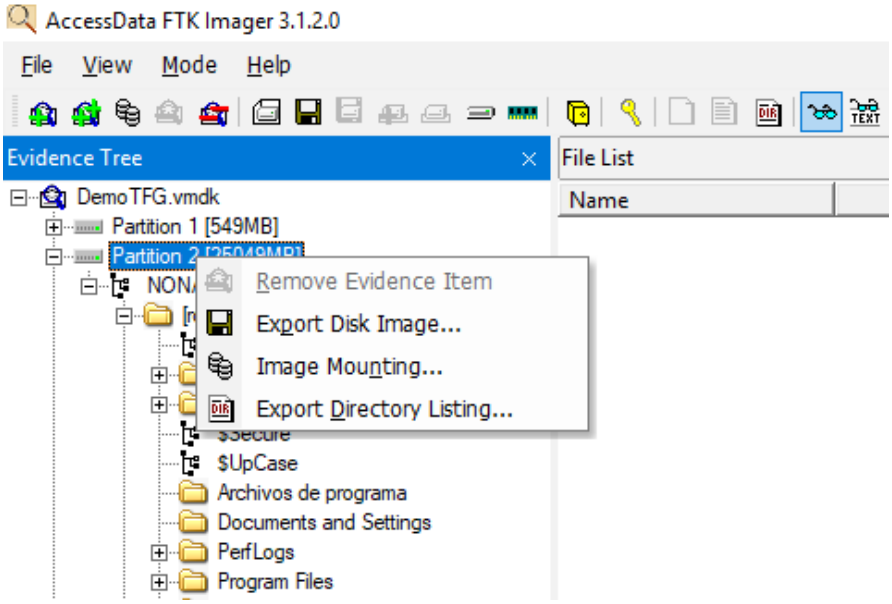

Imagen - 11: Exportar evidencia 1

4. Aparecerá una ventana como la siguiente:

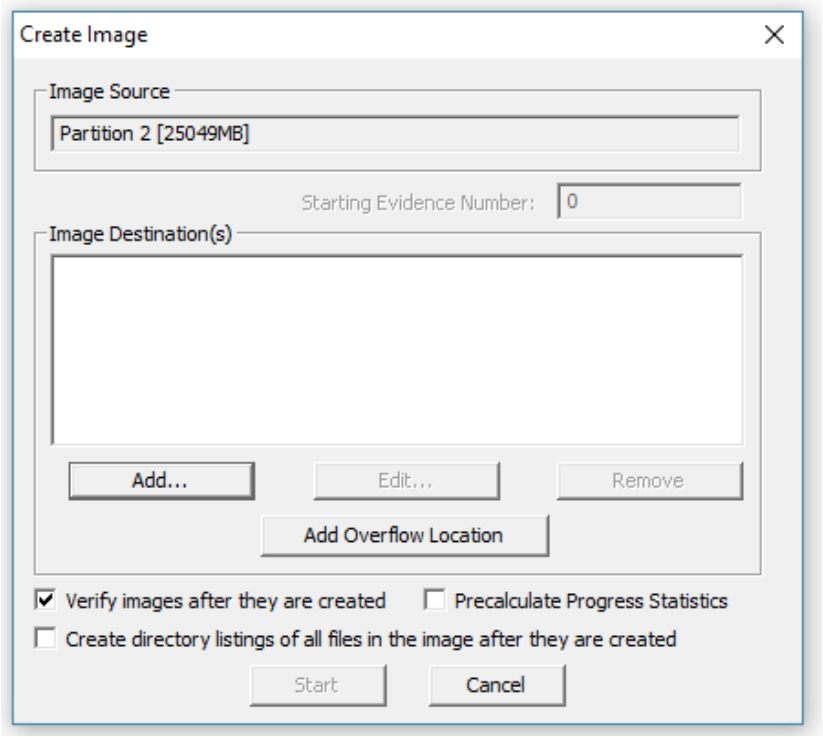

Imagen - 12: Exportar evidencia 2

5. Simplemente debemos pulsar el botón "Add…" y seguir los pasos que irán apareciendo en sucesivas ventanas emergentes.
    a. En la primera ventana emergente, elegimos el formato al que queremos exportar nuestra nueva imagen. Como se ha mencionado anteriormente, queremos que esté en el formato .dd.

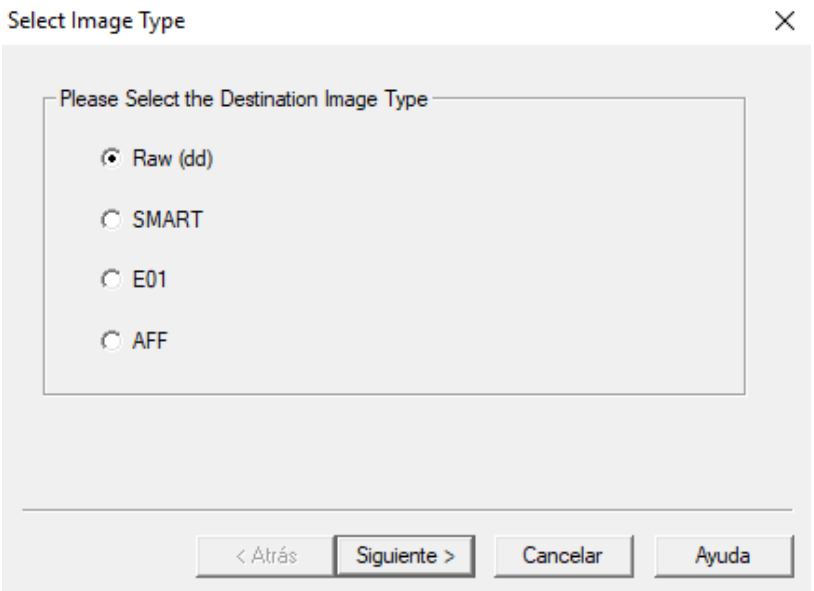

Imagen - 13: Exportar evidencia 3

b. En la siguiente ventana emergente, solo tenemos que insertar información básica de la imagen. No es necesario completar esta ventana.

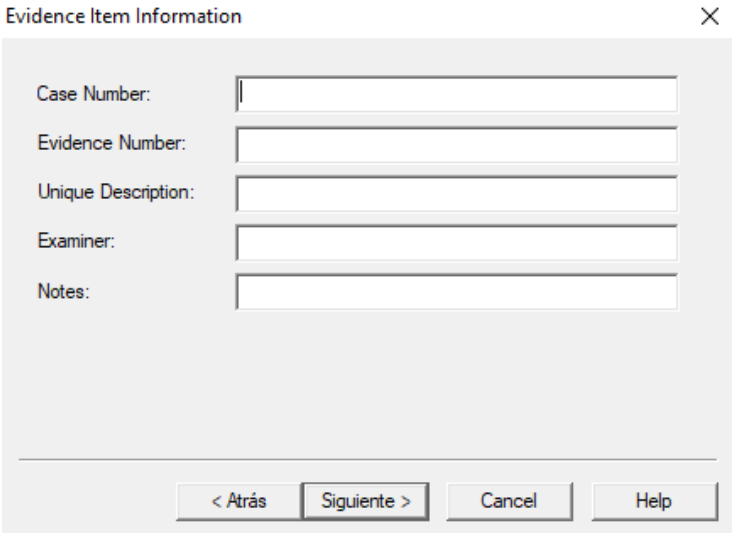

Imagen - 14: Exportar evidencia 4

c. En la última ventana, debemos seleccionar el lugar donde queremos guardar la nueva imagen que se va a crear. Además, debemos poner un nombre a dicha imagen.

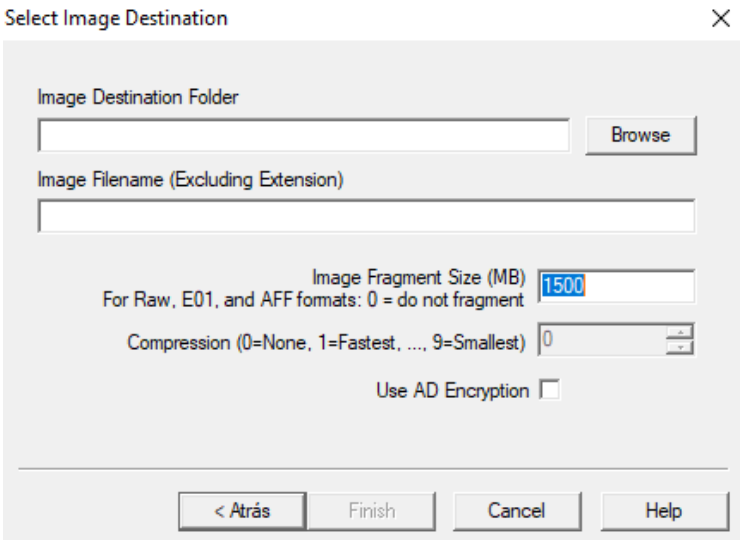

Imagen - 15: Exportar evidencia 5

No se nos puede olvidar, si lo deseamos, en el apartado “Image Fragment Size (MB)” cambiarlo de 1500 a 0. En 0, FTK Imager no generará particiones de distintos tamaños a la hora de crear la imagen.

6. Una vez completadas toda la serie de ventanas emergentes, sólo debemos pulsar “Start” para que la exportación comience.
7. Al finalizar, ya contaremos con una evidencia forense en formato .dd, totalmente acta para su utilización en Autopsy.

6.2. Creación de un caso

Para crear un “caso” en Autopsy, hay que seguir el siguiente procedimiento:

1. Al ejecutar Autopsy, aparecerá una ventana emergente en la que tenemos tres opciones a elegir: “Create New Case”, “Open Recient Case” o “Open Existing Case”.

Imagen - 16: Autopsy

Para crear un nuevo caso, se debería seleccionar la primera opción. La segunda opción permite abrir casos que se hayan creado recientemente y, la tercera opción, permite abrir casos ya existentes.

2. Al elegir la primera opción, se abrirá una serie de ventanas emergentes que se deberán completar para conseguir crear un caso.
    a. Para comenzar, se debe poner un nombre al caso y elegir el lugar donde se quiere guardar.

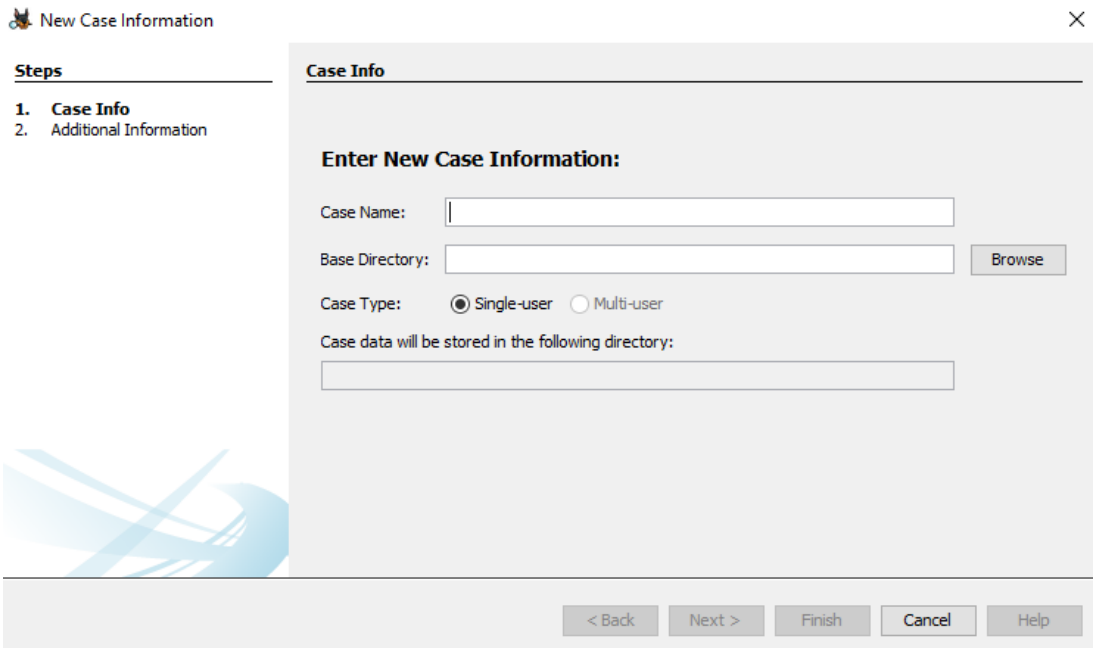

Imagen - 17: Crear caso 1

b. En la siguiente ventana se puede indicar el número del caso y el examinador del mismo.

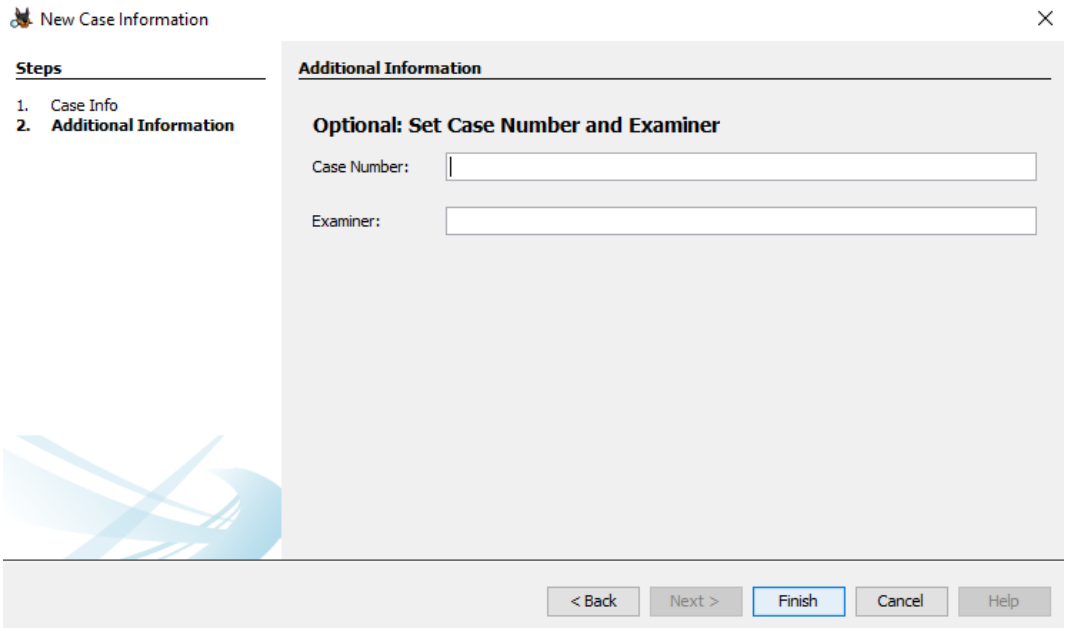

Imagen - 18: Crear caso 2

6.3. Carga de una evidencia

Una vez creado el caso, Autopsy permitirá la carga de la evidencia en dicho caso. Pulsando el botón “Add Data Source”, aparecerá una serie de ventanas emergentes para proceder a la carga de la evidencia.

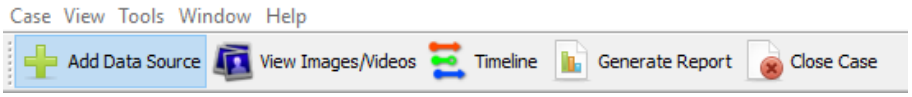

Imagen - 19: Barra herramientas Autopsy

En la primera ventana simplemente hay que elegir el tipo de fuente de datos que se va a añadir. Como la fuente de datos con la que se va a trabajar es una evidencia forense, se deberá seleccionar la primera opción (Disk Image or VM File).

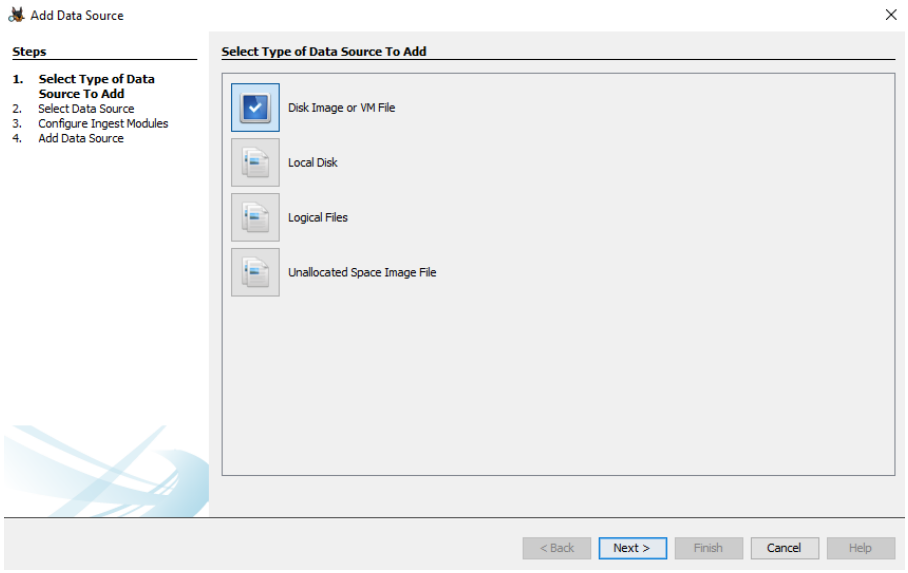

Imagen - 20: Cargar evidencia 1

En la siguiente ventana emergente, se debe localizar la evidencia en el equipo (Importante: la evidencia deberá ser de formato .dd, como se ha mencionado anteriormente). También, si se desea, se puede elegir la localización donde se está realizando este procedimiento, aunque no es necesario.

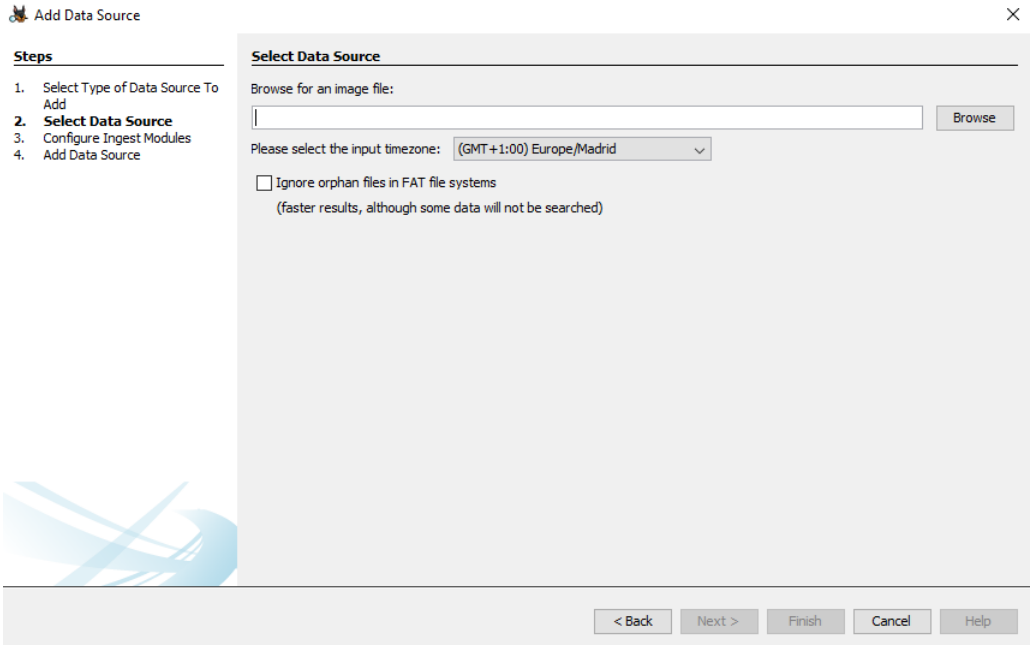

Imagen - 21: Cargar evidencia 2

A continuación, aparecerá una última ventana donde se puede elegir los módulos de ingesta que se quieren utilizar sobre esta evidencia.

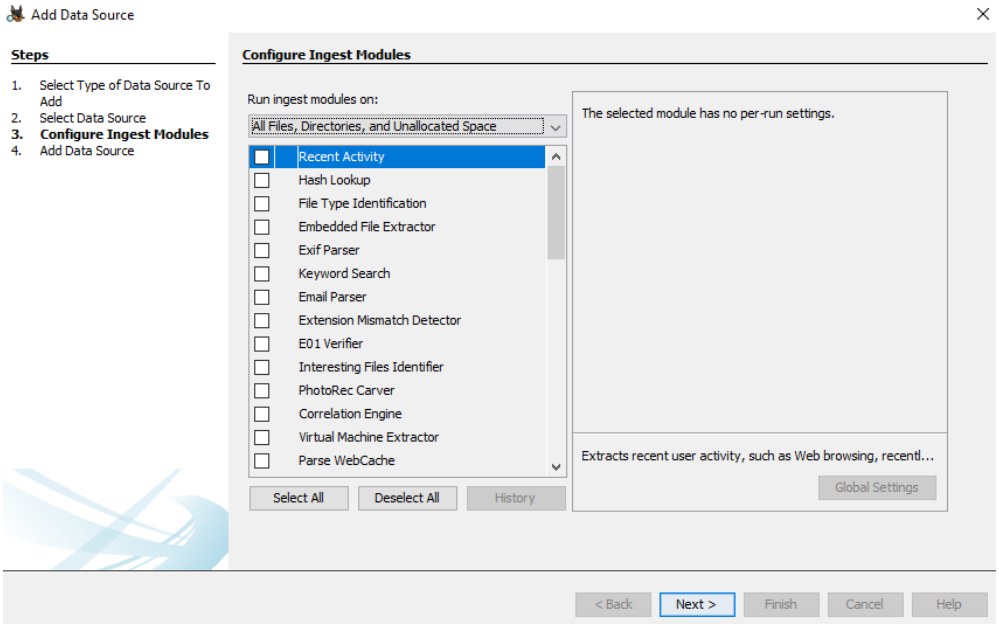

Imagen - 22: Cargar evidencia 3

En este momento, no es necesario seleccionar algún módulo de ingesta, se pueden ejecutar más adelante, como se mostrará más adelante.

Finalmente, aparecerá una ventana de progresión de carga de la evidencia.

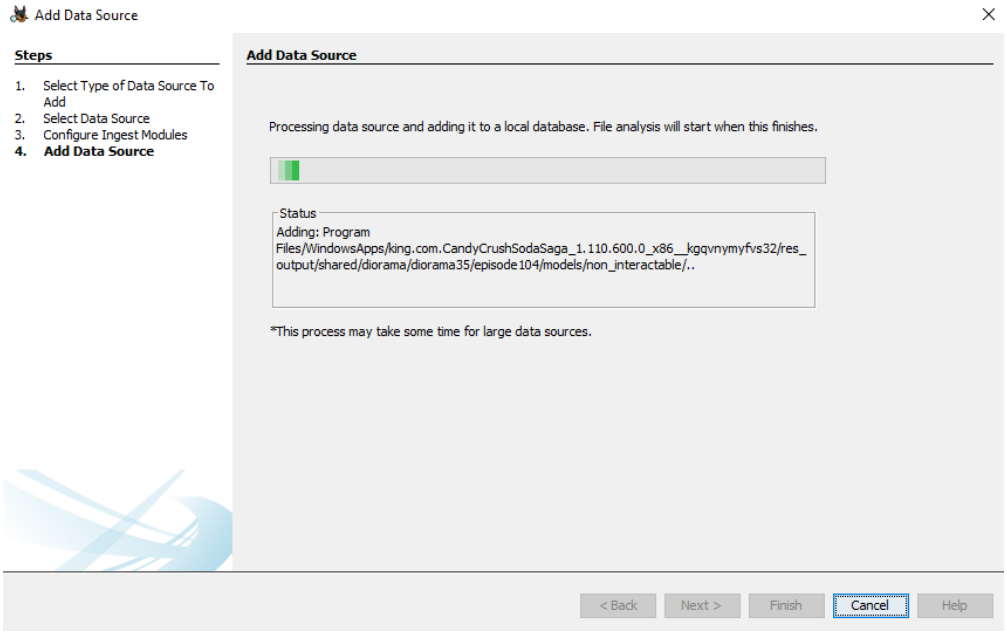

Imagen - 23: Cargar evidencia 4

Cuando finalice, emergerá una nueva ventana donde avisará que el proceso ha finalizado con éxito.

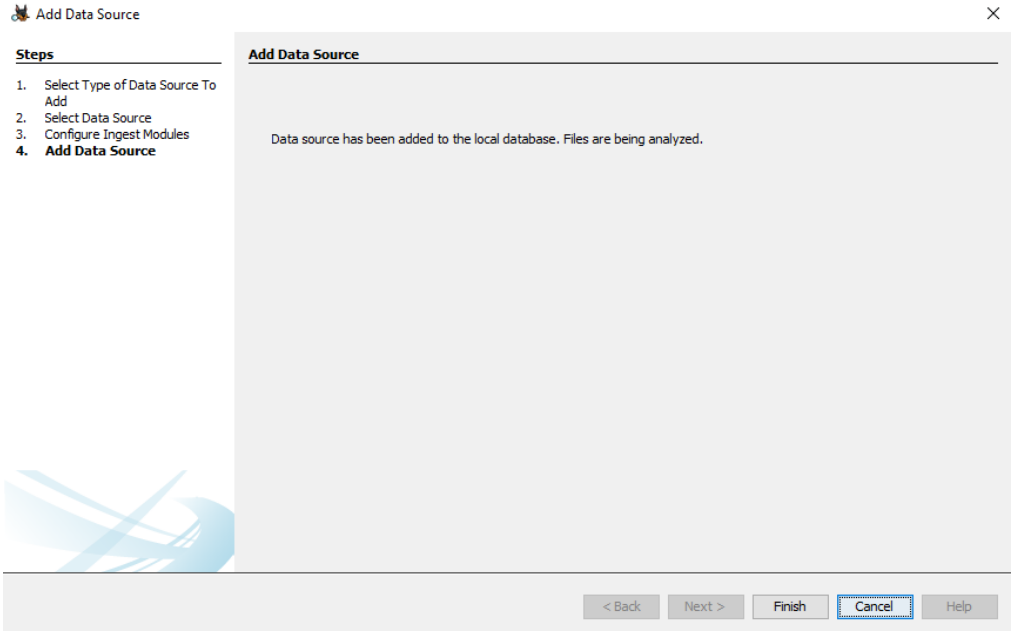

Imagen - 24: Cargar evidencia 5

6.4. Visualización de la evidencia

Una vez cargada la evidencia en el caso de Autopsy, de una manera muy similar que FTK Imager, Autopsy muestra un árbol de evidencia donde se puede navegar por los distintos archivos que componen dicha evidencia:

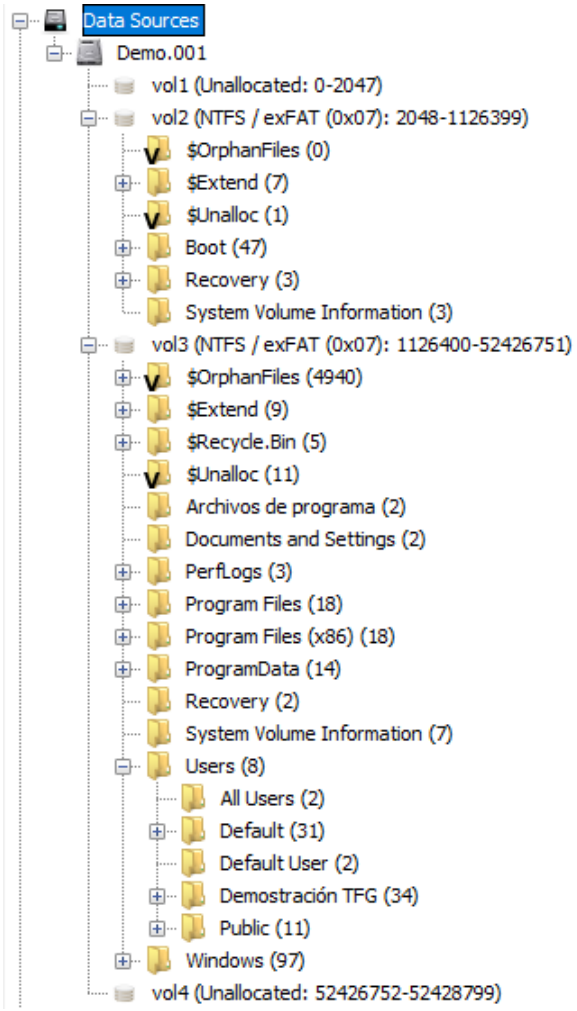

Imagen - 25: Árbol evidencia Autopsy

Además, en el mismo árbol, aparece una sección donde se guardará la información obtenida de la realización del análisis (sección de vista):

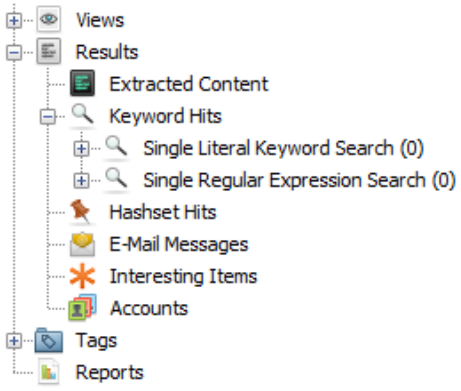

Imagen - 26: Sección de vista

6.5. Módulos de ingesta

Se trata de una de las utilidades más poderosas de Autopsy. Con los módulos de ingesta, los usuarios de Autopsy tienen la posibilidad de aumentar el número de funcionalidades gracias a la utilización de plugins creados por los desarrolladores o por los propios usuarios.

Estos plugins están desarrollados en Python y se guardan en una carpeta determinada, cuya localización la proporciona el propio Autopsy.

7. Plugin

Un Plugin, o complemento, es un componente de software que agrega una función específica a un programa informático existente. Cuando un programa admite complementos, permite su personalización. Los ejemplos comunes son los complementos que se usan en los navegadores web para agregar nuevas características como motores de búsqueda, escáneres de virus o la capacidad de usar un nuevo tipo de archivo, como un nuevo formato de video. Los complementos de navegador conocidos incluyen Adobe Flash Player, QuickTime Player, y el complemento Java, que puede iniciar un applet Java activado por el usuario en una página web para su ejecución en una máquina virtual Java local.

7.1. Propósitos y ejemplos

Las aplicaciones admiten complementos por muchas razones. Algunas de las principales son:

- Agregar nuevas características.
- Reducir el tamaño de una aplicación.
- Separar el código fuente de una aplicación debido a licencias de software incompatibles.

Tipos de aplicaciones y por qué usan complementos:

- Los editores de audio usan complementos para generar, procesar o analizar sonido. “Ardor” y “Audacity” son ejemplos de tales editores.
- Las estaciones de trabajo de audio digital (DAW) usan complementos para generar sonido o procesarlo. Los ejemplos incluyen “Logic ProX” y “ProTools”.
- El software de ingeniería, asistido por computadora, usa complementos para ampliar las funciones de otro programa de software, por ejemplo, “MBD” para “ANSYS”.
- Los clientes de correo electrónico usan complementos para descifrar y cifrar el correo electrónico. “Pretty Good Privacy” es un ejemplo de tales complementos.
- Los emuladores de videoconsolas a menudo usan complementos para modularizar los subsistemas separados de los dispositivos que intentan emular. Por ejemplo, el emulador “PCSX2” hace uso de plug-ins de video, audio, óptica, etc, para esos componentes respectivos de la PlayStation 2.
- El software de gráficos utiliza complementos para admitir formatos de archivo y procesar imágenes (*cf* plugin de “Photoshop”).
- Los reproductores de medios usan complementos para admitir formatos de archivo y aplicar filtros. “Foobar2000”, “GStreamer” , “Quintessential “, “VST” , “Winamp” , “XMMS” son ejemplos de tales reproductores multimedia.
- Los rastreadores de paquetes usan complementos para decodificar formatos de paquetes. "OmniPeek” es un ejemplo de tales sniffers de paquetes.
- Las aplicaciones de detección remota utilizan complementos para procesar datos de diferentes tipos de sensores; *por ejemplo*, “Opticks”.

- Los editores de texto y los entornos de desarrollo integrados usan complementos para admitir lenguajes de programación o mejorar el proceso de desarrollo, *por ejemplo* , los complementos de soporte de “Visual Studio”, “RAD Studio”, “Eclipse”, “IntelliJ IDEA”, “jEdit” y “MonoDevelop”. “Visual Studio” se puede conectar a otras aplicaciones a través de “Visual Studio Tools for Office” y “Visual Studio Tools for Applications”.
- Los navegadores web usan extensiones de navegador para expandir su funcionalidad. Los ejemplos incluyen “Adobe Flash Player”, “Java SE”, “QuickTime”, “Microsoft Silverlight” y “Unity”.

7.2. Funcionamiento

La aplicación de host proporciona servicios que el complemento puede utilizar, incluida una forma de que los complementos se registren con la aplicación de host y un protocolo para el intercambio de datos con complementos. Los complementos dependen de los servicios proporcionados por la aplicación host y generalmente no funcionan solos. Por el contrario, la aplicación host opera independientemente de los complementos, lo que permite a los usuarios finales agregar y actualizar complementos dinámicamente sin necesidad de realizar cambios en la aplicación host.

Los programadores suelen implementar la funcionalidad de complemento utilizando bibliotecas compartidas, que se cargan dinámicamente en tiempo de ejecución, instaladas en un lugar prescrito por la aplicación host. HyperCard admite una instalación similar, pero más comúnmente incluye el código de complemento en los documentos de HyperCard (llamados *stacks*). Por lo tanto, la pila HyperCard se convirtió en una aplicación autónoma por derecho propio, que se distribuía como una sola entidad que los usuarios finales podían ejecutar sin la necesidad de pasos de instalación adicionales. Los programas también pueden implementar complementos al cargar un directorio de archivos de script simples escritos en un lenguaje de scripts como Python o Lua.

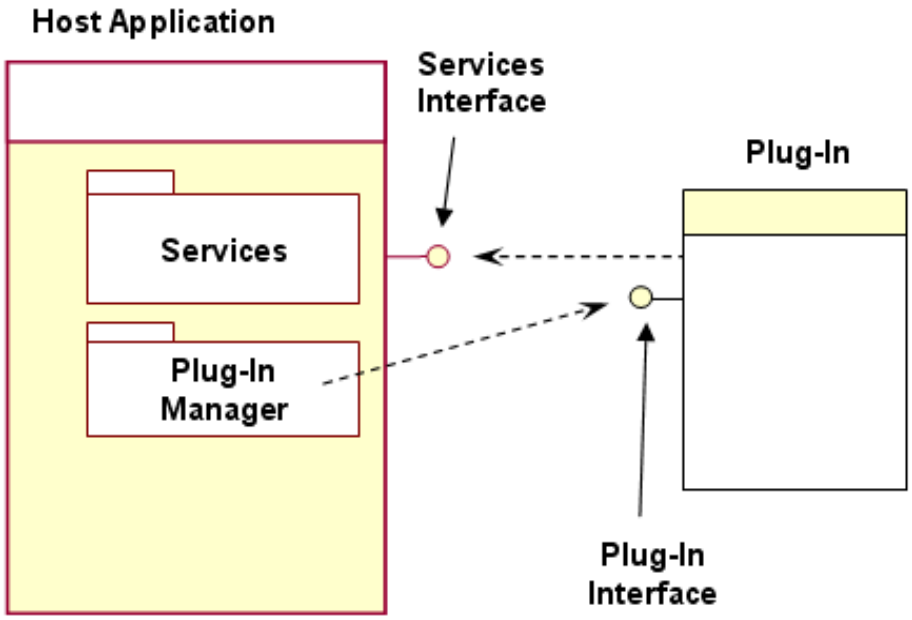

Imagen - 27: Plugin

7.3. Implementación. Construir un Plugin para Autopsy

Los plugins están desarrollados en Python 2.7. Se trata de un lenguaje muy versátil, utilizado tanto para programación ordinaria como para programación orientada a objetos o visual.

En este apartado se explica cómo debe estar desarrollado un plugin para que pueda ser visualizado y ejecutado como un complemento de Autopsy.

Primero, se debe tener en cuenta las librerías que necesita el plugin para que Autopsy reconozca el plugin en cuanto se encuentre en su correspondiente localización.

- Las siguientes librerías corresponden a la parte de visualización del plugin, es decir, que los parámetros que deseemos que aparezcan legibles en Autopsy. Al margen de estas librerías, podrían ser necesarias otras librerías desarrolladas por terceros:

```
from java.lang import Class
from java.lang import System
from java.sql  import DriverManager, SQLException
from java.util.logging import Level
from java.io import File
from org.sleuthkit.datamodel import SleuthkitCase
from org.sleuthkit.datamodel import AbstractFile
from org.sleuthkit.datamodel import ReadContentInputStream
from org.sleuthkit.datamodel import BlackboardArtifact
from org.sleuthkit.datamodel import BlackboardAttribute
```

- A continuación, se encuentran las librerías necesarias para el tratamiento del plugin mediante la herramienta Autopsy.

```
from org.sleuthkit.autopsy.ingest import IngestModule
from org.sleuthkit.autopsy.ingest.IngestModule import IngestModuleException
from org.sleuthkit.autopsy.ingest import DataSourceIngestModule
from org.sleuthkit.autopsy.ingest import IngestModuleFactoryAdapter
from org.sleuthkit.autopsy.ingest import IngestModuleIngestJobSettings
from org.sleuthkit.autopsy.ingest import IngestModuleIngestJobSettingsPanel
from org.sleuthkit.autopsy.ingest import IngestMessage
from org.sleuthkit.autopsy.ingest import IngestServices
from org.sleuthkit.autopsy.ingest import ModuleDataEvent
from org.sleuthkit.autopsy.coreutils import Logger
from org.sleuthkit.autopsy.coreutils import PlatformUtil
from org.sleuthkit.autopsy.casemodule import Case
from org.sleuthkit.autopsy.casemodule.services import Services
from org.sleuthkit.autopsy.casemodule.services import FileManager
from org.sleuthkit.autopsy.datamodel import ContentUtils
```

Para continuar con la implementación de un plugin para Autopsy, son necesarias dos clases, con las que se consigue el correcto funcionamiento con dicha herramienta (sin estas dos clases, Autopsy no reconocería la existencia de los Plugins):

```
"""   Sustituir XXXXXXXXX por el nombre o la información relacionada con el Plugin general
      Sustituir YYYYYYYYY por el nombre o la información relacionada con los Plugins a combinar  """
```

- Primera clase: se consigue realizar una serie de funciones que contendrán la información de los metadatos del plugin (nombre, descripción, versión…).

```
class XXXXXXXXXIngestModuleFactory(IngestModuleFactoryAdapter):

    def __init__(self):

        self.settings = None

    moduleName = "XXXXXXXXX"

    def getModuleDisplayName(self):

        return self.moduleName

    def getModuleDescription(self):

        return "Descripcion de XXXXXXXXX"

    def getModuleVersionNumber(self):

        return "Número de versión (1.0)"

    def isDataSourceIngestModuleFactory(self):

        return True

    def createDataSourceIngestModule(self, ingestOptions):

        return XXXXXXXXXIngestModule(self.settings)
```

- Segunda clase: en esta clase se van a crear las funciones que va a desplegar el plugin.

```
class XXXXXXXXXIngestModule(DataSourceIngestModule):

        _logger = Logger.getLogger(XXXXXXXXXIngestModuleFactory.moduleName)

        def log(self, level, msg):

            self._logger.logp(level, self.__class__.__name__, inspect.stack()[1][3], msg)

        def __init__(self, settings):

                    self.context = None

                    self.local_settings = settings

                    self.List_Of_Events = []
```

  - Aparecen funciones iniciales como inicializadores de variables, funciones de log, etc.

- Además de las funciones que aparecen en esta clase, se puede añadir una serie de funciones adiccionales. La siguiente clase, permite la ejecución de ejecutables externos para la realización de distintas funcionalidades necesarias en el Plugin (por ejemplo, exportar archivos, exportar bases de datos, etc):

```
def startUp(self, context):

self.context = context

if self.local_settings.getXXXXXXXXX_Flag():

self.log(Level.INFO, "XXXXXXXXX ==> " + str(self.local_settings.getXXXXXXXXX_Flag()))

self.path_to_XXXXXXXXX_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "XXXXXXXXX.exe")

if not os.path.exists(self.path_to_XXXXXXXXX_file):
   raise IngestModuleException("XXXXXXXXX Executable does not exist")
```

- Finalmente, se pueden añadir una serie de clases para mejorar la visualización de la selección de los Plugins, si se cree necesario, como, por ejemplo, menús de selección CheckBox (como en el caso de Windows_Internals):

```
class XXXXXXXXXWithUISettings(IngestModuleIngestJobSettings):
   serialVersionUID = 1L

   def __init__(self):     #aparecen los ckeckbox desmarcados (uno por cada Plugin que se desee combinar)
      self.YYYYYYYYY_Flag = False

   def getVersionNumber(self):
      return serialVersionUID

   #definir métodos get y set (para cada plugin)
   def getYYYYYYYYY_Flag(self):
      return self.Blobdec_Flag

   def setYYYYYYYYY_Flag(self, flag):
      self.Blobdec_Flag = flag
.
.
.
```

```
class XXXXXXXXXWithUISettingsPanel(IngestModuleIngestJobSettingsPanel):
   # We get passed in a previous version of the settings so that we can
   # prepopulate the UI
   def __init__(self, settings):
      self.local_settings = settings
      self.initComponents()
      self.customizeComponents()

   def checkBoxEvent(self, event):

      if self.YYYYYYYYY_CB.isSelected():
         self.local_settings.setYYYYYYYYY_Flag(True)
      else:
         self.local_settings.setYYYYYYYYY_Flag(False)
.
.
.
```

```
    def initComponents(self):
        #aparencia visible en Autopsy
        self.panel0 = JPanel()

        self.gbPanel0 = GridBagLayout()
        self.gbcPanel0 = GridBagConstraints()
        self.panel0.setLayout( self.gbPanel0 )

        self.Recentlyused_CB = JCheckBox( "Blobdec", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2 #valor de colocación respecto al eje X
        self.gbcPanel0.gridy = 5 #valor de colocación respecto al eje Y
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Recentlyused_CB, self.gbcPanel0 )
        self.panel0.add( self.Recentlyused_CB )

        self.Jumplist_CB = JCheckBox( "YYYYYYYYY", actionPerformed=self.checkBoxEvent)
.
.
.


    def customizeComponents(self):

        self.YYYYYYYYY_CB.setSelected(self.local_settings.getYYYYYYYYY_Flag())
.
.
.


    # retornar la configuración utilizada
    def getSettings(self):
        return self.local_settings
```

7.4. ***Funcionalidad de los Plugins***

A continuación, se mostrará la funcionalidad de una serie de Plugins desarrollados para aumentar las funcionalidades de Autopsy y mejorar el análisis forense:

- Amazon_Echosystem_Parser: analiza las bases de datos de una imagen de Amazon Alexa.
- CCM_RecentlyUsedApps: analiza la base de datos WMI () para aplicaciones usadas recientemente.
- FileHistory: se crea una base de datos SQLite que contiene la información del historial de archivos y, después, se importa a la sección de vista extraída de Autopsy.
- Jump_List_AD: se crea una base de datos SQLite que contiene la información JumpList y luego se importa a la sección de vista extraída de Autopsy.
- MacFSEvents: exporta el directorio .fsevents y ejecuta el programa FSEParser_v2.1.exe con los datos exportados. A continuación, importará la base de datos SQLite que se creó a partir del programa.
- MacOSX_Recent: exporta/analiza Mac´s recientes.
- Parse_Plist: analiza cualquier Plist y la convierte en una base de datos SQLite e importa la información en el contenedor extraído.
- Parse_SAM: crea una base de datos SQLite que contiene información SAM, que posteriormente importará a la sección de vista extraída de Autopsy.

- Parse_Shellbags: crea una base de datos SQLite que contiene la información del shellbag y luego se importa a la sección de vista extraída de Autopsy.
- Parse_SQLite_Databases: analiza cualquier archivo SQLite y los importa a la sección de contenido extraído de Autopsy.
- Parse_SQLite_Del_Records: analiza cualquier base de datos SQLite y busca registros eliminados. Luego creará una base de datos SQLite con los registros eliminados y luego se importará a la sección de contenido extraído de Autopsy.
- Parse_USNJ: se crea una base de datos SQLite que contiene la información NTFS UsrJrnl y se importa en la sección de vista extraída de Autopsy.
- Process_Amcache: una base de datos SQLite contiene la información de Amcache que se crea y luego se importa en la vista de contenido extraído de Autopsy.
- Process_EVTX: crea una base de datos SQLite que contiene la información del registro de eventos y se importa en la sección de vista extraída de Autopsy.
- Process_EVTX_By_EventID: se crea una base de datos SQLite que contiene la información del registro de eventos y luego se importa en la sección de vista extraída de Autopsy como una tabla basada en Event_Log_Id. El usuario puede ejecutar el módulo nuevamente y extraer los eventos proporcionados por el usuario desde la base de datos Evtx SQLite.
- Process_Prefetch_Files_V41: se crea una base de datos SQLite que contiene la información del registro de eventos y luego se importa en la sección de vista extraída de Autopsy como una tabla basada en Event_Log_Id. El usuario puede ejecutar el módulo nuevamente y extraer los eventos proporcionados por el usuario desde la base de datos Evtx SQLite.
- Process_SRUDB: se crea una base de datos SQLite que contiene la información de Uso de recursos y luego se importa en la vista extraída de Autopsy.
- Recient_Activity: entre otras funcionalidades, recopila información sobre la actividad del ususario en los navegadores web y en el sistema operativo.
- Shimcache_parser: Se crea una base de datos SQLite que contiene la información shimcache y luego se importa en la sección de vista extraída de Autopsy.
- Thumbcache_parser: exporte todos los archivos Thumbcache _ *. db en la imagen y luego ejecute el programa thumbcache_viewer_cmd contra ellos y exporte los archivos incrustados al directorio ModuleOutput para que los archivos puedan volverse a agregar a Autopsy.
- Thumbs_parser: exporte todos los archivos thumbs.db en la imagen y luego ejecute el programa thumbs_viewer contra ellos y exporte los archivos incrustados al directorio ModuleOutput para que los archivos puedan volver a agregarse a Autopsy.
- Volatility: ejecute Volatility contra una imagen de memoria. Le pedirá al usuario el directorio donde reside el ejecutable de Volatility y luego ejecutará la volatilidad contra la imagen de memoria utilizando las opciones que el usuario especifique.
- Webcache: el módulo exportará el archivo WebcacheV01 y luego llamará a la versión de línea de comando de Export_Esedb. Se crea una base de datos SQLite que contiene la información de caché Web y luego se importa a la sección de vista extraída de Autopsy.
- Windows_Internals: varios plugins de Windows combinados en un plugin. Integra las funcionalidades de algunos Plugins ya mencionados (CCM_RecentlyUsedApps, Jump_List_AD, FileHistory, Parse_SAM, Parse_Shellbags, Shimcache_parser y Webcache).

8. Integración del Plugin en Autopsy. Módulos de ingesta

A continuación, se muestra un ejemplo de como se realizaría el análisis forense de una evidencia de forma automatizada, integrando y ejecutando uno de los Plugins mencionados anteriormente en Autopsy.

8.1. Localización

Autopsy proporciona de forma automática el directorio donde se deben guardar los plugins o complementos desarrollados por terceros.

En la ventana principal, en la parte superior, aparece una serie de menús despegables.

Pulsando en el menú “Tools”, aparecerá la opción “Python Plugins”.

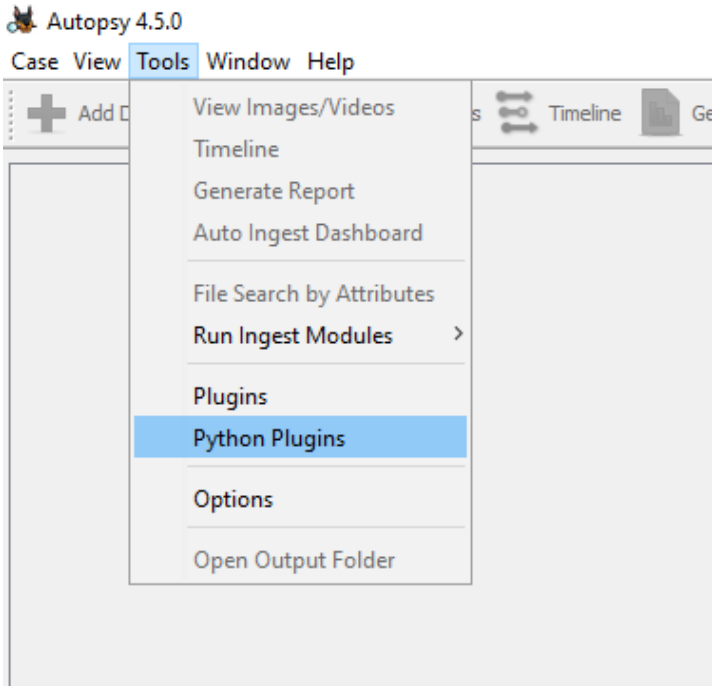

Imagen - 28: Localización Plugins

Seleccionando dicha opción, Autopsy abrirá una carpeta creada en el equipo en el que se aloja, donde se deben situar todos los complementos desarrollados por terceros. La dirección de dicha carpeta es la siguiente:

C:\Users\%USERNAME%\AppData\Roaming\autopsy\python_modules

8.2. Ejecución

Una vez que el Plugin y todos los complementos que necesita (ejecutables, clases, etc) se encuentren en la localización mencionada anteriormente y se haya creado el caso e introducido la evidencia, la ejecución de un Plugin es muy sencilla, sólo hay que seguir los siguientes pasos:

1. Haciendo clic derecho sobre la evidencia, aparecerá un menú donde se puede ver la opción “Run Ingest Modules”, de la que se ha hablado anteriormente.

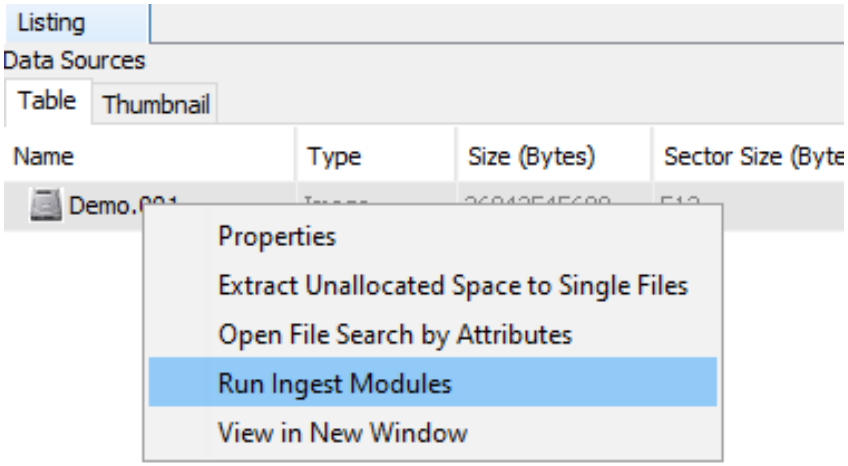

Imagen - 29: Ejecución Módulos de Ingesta 1

2. Pulsando en dicha opción, emergerá una ventana donde aparecen listados los Plugins que se han guardado anteriormente.

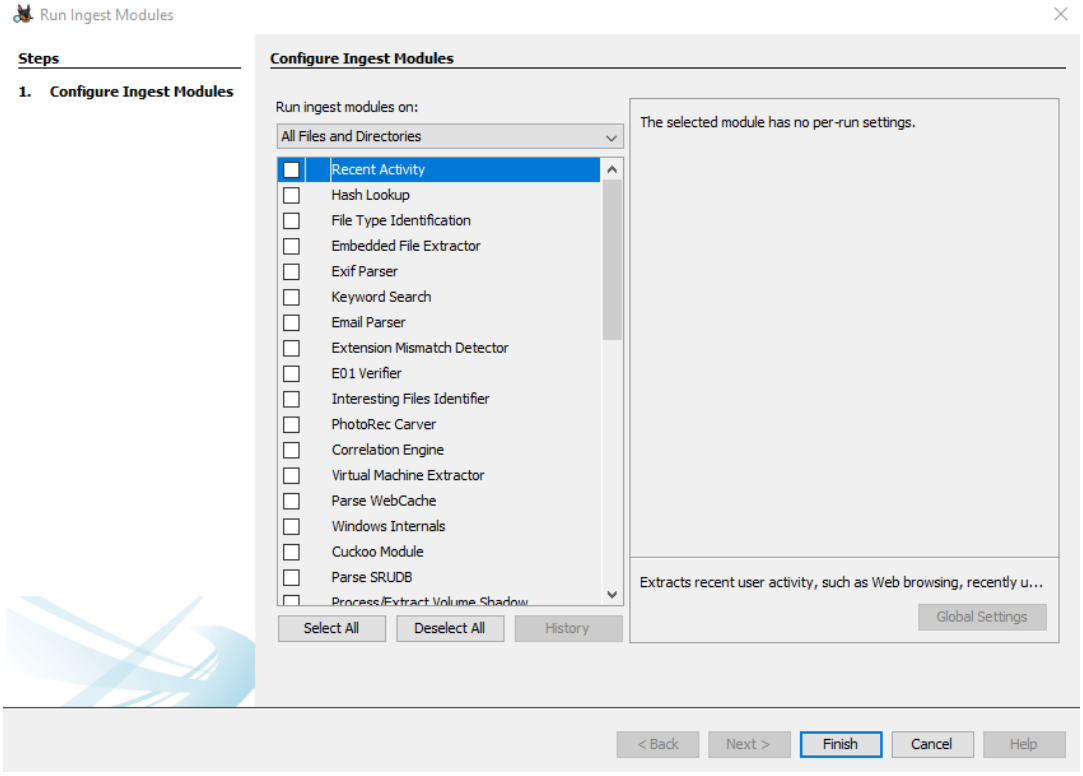

Imagen - 30: Ejecución Módulos de Ingesta 2

3. En esta ventana, se deben elegir los Plugins que se deseen, dependiendo del análisis forense que se quiera realizar. En este caso se van a elegir los Plugins “Windows Internals” y “Extension Mismatch Detector”:
    a. Windows Internals: permite la ejecución simultánea de varios Plugins que permiten obtener, entre otras cosas, el historial de nanavegación. Se marcan las casillas de los Plugins internos que se desea que se ejecuten, en este caso, “File History”.

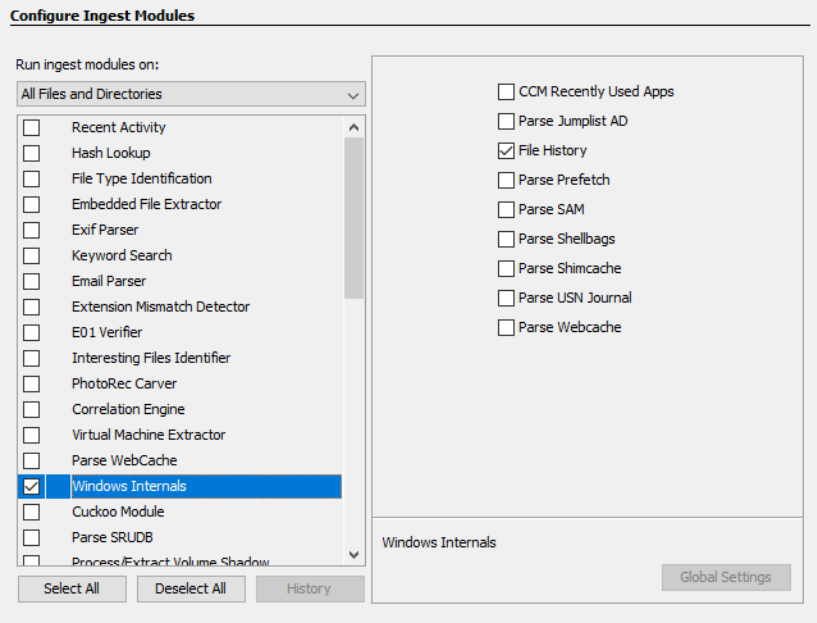

Imagen - 31: Ejecución Módulos de Ingesta 3

    b. Extension Mismatch Detector: en el caso de este Plugin, permite la selección de los archivos que se desea obtener, analizando las extensiones de los archivos. Se puede elegir entre tres opciones:
        - Comprobar todos los archivos.
        - Comprobar todos los archivos excepto los archivos de texto.
        - Comprobar sólo los archivos multimedia y ejecutables.

    Además, permite filtrar la búsqueda evitando los siguientes casos

        - Omitir los archivos sin extensiones.
        - Omitir los archivos conocidos.

    :

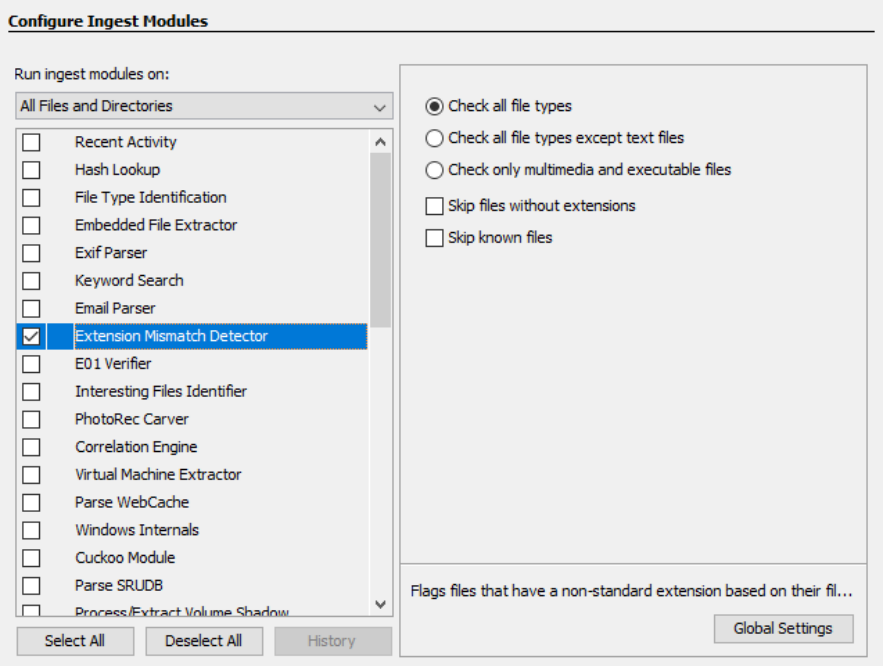

Imagen - 32: Ejecución Módulos de Ingesta 4

4. Se puede elegir tantos Plugins como se deseen. Una vez seleccionados los Plugins, se presiona el botón "Finish" y Autopsy se pondrá a trabajar. Se puede ver el progreso del análisis de Autopsy en la parte inferior de la ventana principal.

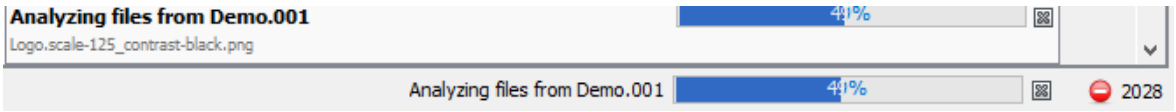

Imagen - 33: Estado del análisis

5. Una vez que Autopsy haya finalizado el análisis, se podrán visualizar los resultados obtenidos.

8.3. Visualización de resultados

En el árbol de la evidencia, existe una parte que se denomina "sección de vista", donde se muestran los resultados que se han obtenido del análisis que ha realizado Autopsy.

Se pueden distinguir dos secciones independientes:

- Views: resultados obtenidos de un análisis realizado sobre la evidencia para la obtención de archivos. Se pueden diferenciar tres apartados distintos:
    - File types: todos aquellos archivos que se han encontrado, dependiendo de su tipo. Se puede visualizar estos resultados de dos formas distintas:
        - Por su extensión: imágenes, vídeos, audios, archivos, documentos (HTML, Office, PDF, Plain Text o Rich Text) o ejecutable (.exe, .dll, .bat, .cmd o .com).
        - Por su naturaleza: aplicaciones, audios, imágenes, textos o vídeos.

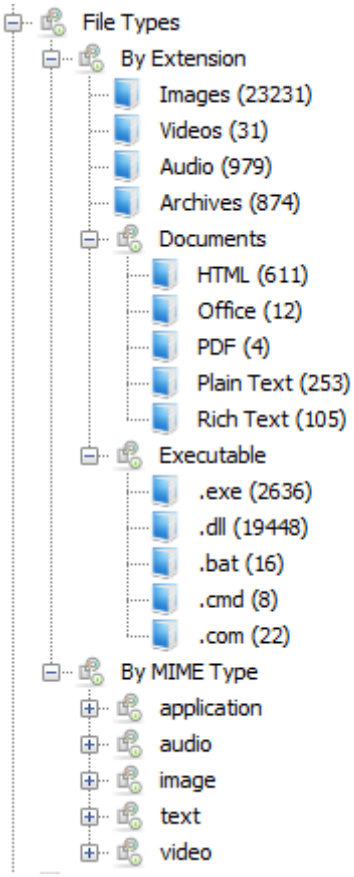

Imagen - 34: File Types

- Deleted files: obtiene todos aquellos archivos que han sido eliminados. Se clasifican entre "File System" y "All".

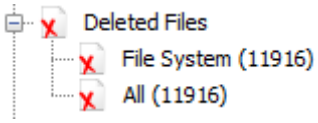

Imagen - 35: Deleted Files

- File size:

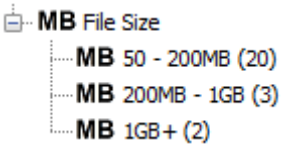

Imagen - 36: File Size

- Results: esta sección, muestra los resultados obtenidos mediante el uso de complementos externos a Autopsy.

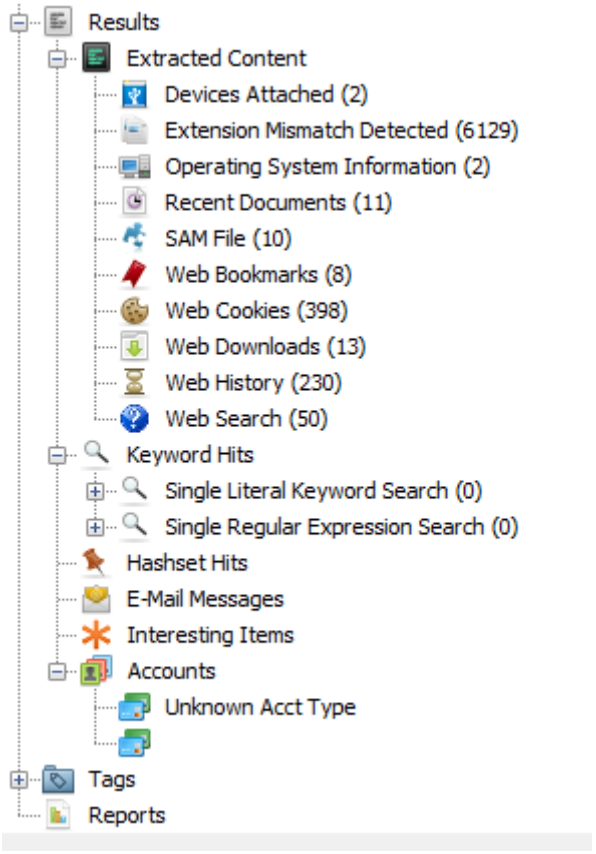

Imagen - 37: Resultados análisis

Además de poder visualizar los resultados obtenidos en el árbol de evidencia, en la ventana principal de Autopsy aparecen dos secciones para la visualización de los resultados, cada una de ellas destinada a una finalidad:

- Listing: en esta sección se listan los archivos que se consideran relevantes en el momento que se ejecuta el Plugin. Consta de dos pestañas:
    - Table: muestra las características de los datos encontrados, en este caso, de las imágenes.

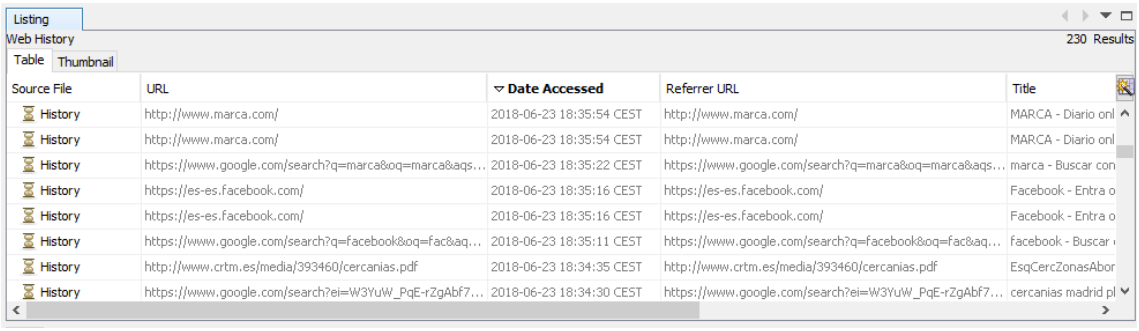

Imagen - 38: Vista de historial

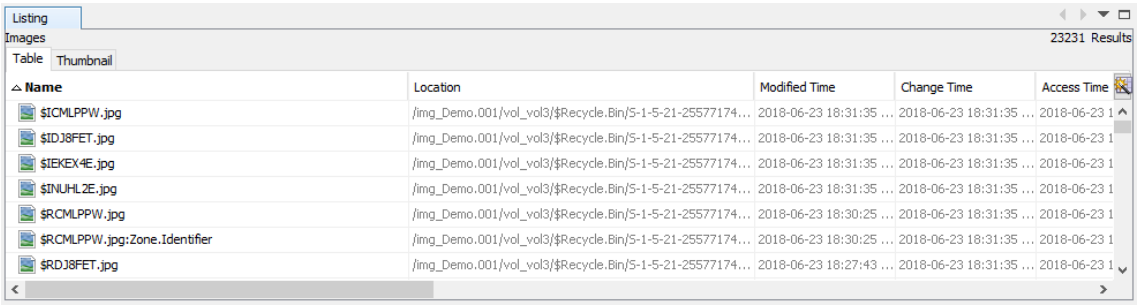

Imagen - 39: Vista de archivos 1

- Thumbnail: esta pestaña lista los resultados de igual modo que la anterior, pero en lugar de mostrar características de los archivos, muestra los archivos de una forma más legible, ideal en casos como el de las imágenes.

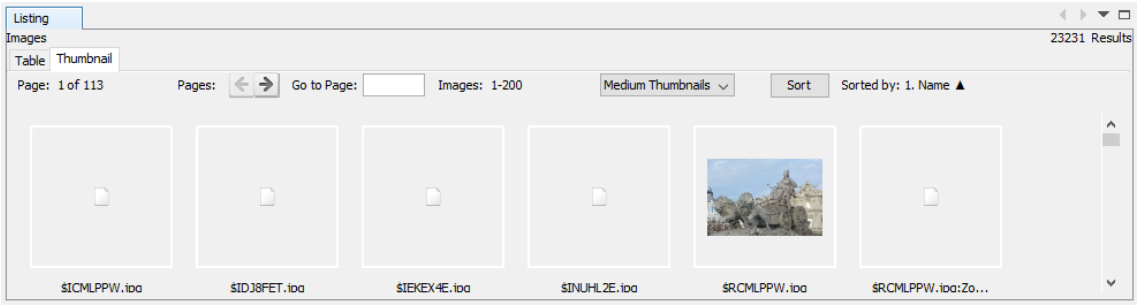

Imagen - 40: Vista de archivos 2

- Justo debajo, aparece una sección que te muestra de una forma más completa los resultados del archivo seleccionado en Listing (Hex, Strings, File Metadata, Results, Indexed Text, Media, Other Occurrences), como, por ejemplo:
  - Hex:

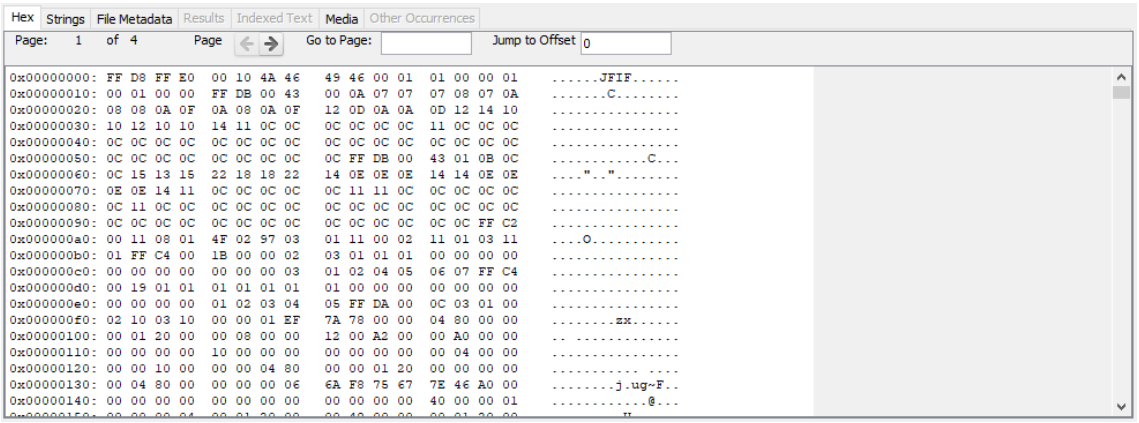

Imagen - 41: Información del archivo 1

- Strings:

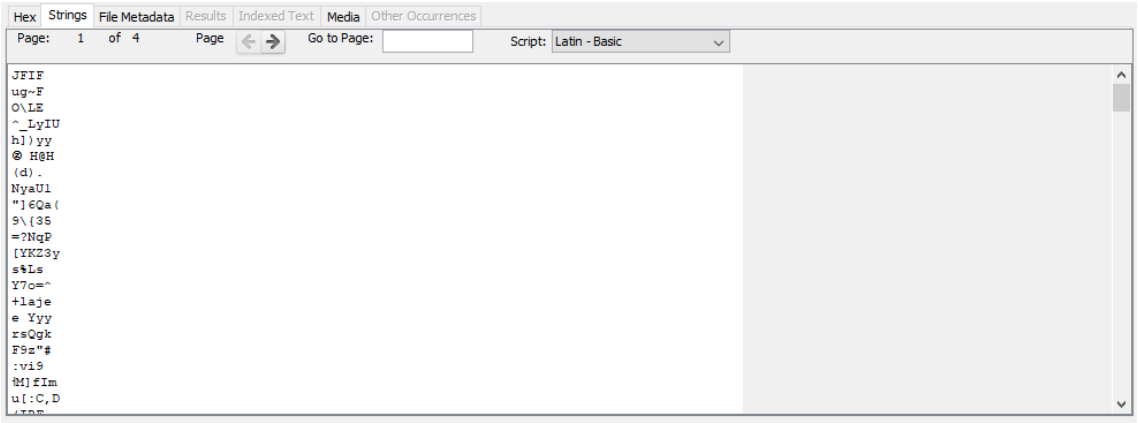

Imagen - 42: Información del archivo 2

- File Metadata: muestra el numbre y tipo del archivo, fecha de última modificación, fecha de último acceso…, incluso hasta su HASH.

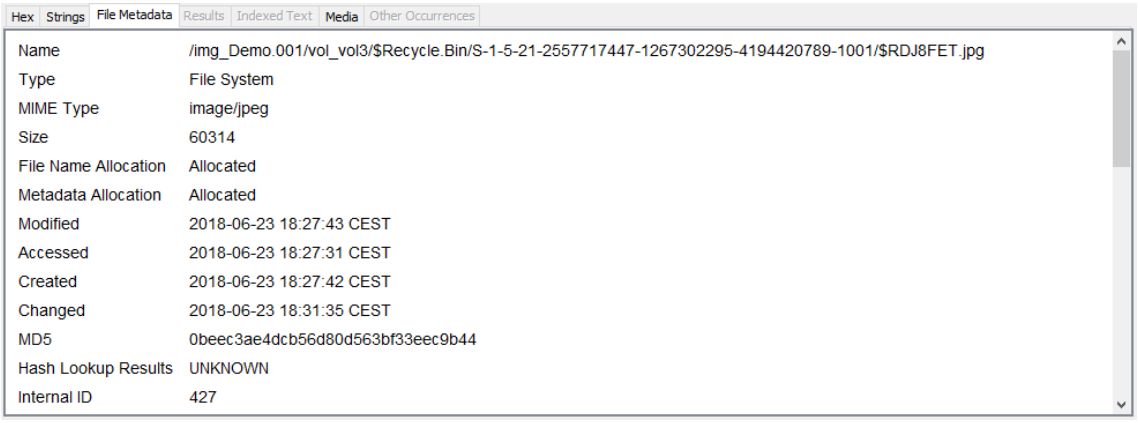

Imagen - 43: Información del archivo 3

- Media: vista previa del archivo, si se trata de un archivo multimedia.

Imagen - 44: Información del archivo 4

Ademas, Autopsy permite la extracción y la visualización de archivos obtenidos mediante la utilización de la misma herramienta. Simplemente, habría que hacer clic derecho sobre el archivo, en la sección de Listing, sobre el archivo que se desea extraer o visualizar. Aparecerá una serie de opciones:

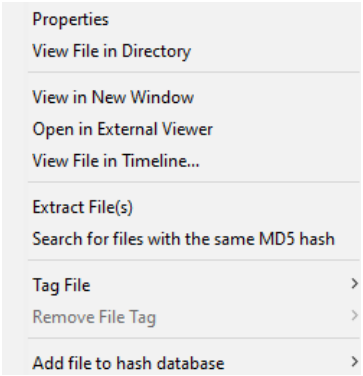

Imagen - 45: Utilidades Autopsy

Para extraer el archivo, bastaría con elegir la opción “Extract File(s)” y elegir la localización en el equipo donde se quiera exportar. Por otro lado, para visualizar el archivo con un programa independiente a Autopsy, bastaría con elegir la opción “Open in External Viewer”.

Hay otras opciones muy interesantes que pueden ayudar al análisis forense, como, por ejemplo, “View File in Directory” para encontrar la localización exacta del archivo (Equipo, Documentos, Carpetas o Subcarpetas, Papelera…) o “Add file to hash database” para posteriormente poder utilizar la base de datos que se va creando con otros Plugins y localizar archivos en una evidencia utilizando directamente la HASH del mismo.

9. Conclusión

Mediante la realización de este proyecto, se ha aumentado los conocimientos adquiridos durante estos años de formación, tanto los conocimiento teóricos como los prácticos. Además, de la aplicación de los mismos en el desarrollo del presente proyecto. Gracias a este proyecto, he descubierto un campo apasionante para poder aplicar mis conocimientos de una forma más profesional.

El pilar principal de la informática forense es la estracción de información de equipos y dispositivos informáticos, pero las aplicaciones de estas técnicas son innumerables y tan amplias como la propia informática, gracias a herramientas y complementos creados para esta finalidad, como se muestra en este documento.

Para finalizar, exponer que los conocimientos adquiridos durante este periodo y expuestos en este proyecto, han alcanzado los objetivos plasmados al inicio de este documento, intentando conseguir una visualización clara y sencilla de los mismos.

10. Bibliografía y enlaces

10.1. Bibliografía

Andrés Marzal Varó, Isabel Gracia Luengo y Pedro García Sevilla, *Introducción a la programación con Python 3*, Universitat Jaume I, ISBN: 978-84-697-1178-1

Juan Manuel Martínez Alcalá, *Extracción de contraseñas de Microsoft Windows*, Universidad de Alcalá, TFC

Francisco Lázaro Domínguez, *Introducción a la Informática Forense*, Ra-Ma, ISBN: 978-84-996-4209-3

10.2. Enlaces

https://es.wikipedia.org/wiki/Criptograf%C3%ADa

https://es.wikipedia.org/wiki/Criptograf%C3%ADa_sim%C3%A9trica

https://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica

https://salmocorpblog.wordpress.com/2017/01/27/criptografia-simetrica-y-asimetrica/

https://en.wikipedia.org/wiki/Hash_function

https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/

https://es.wikipedia.org/wiki/C%C3%B3mputo_forense

http://www.criptored.upm.es/descarga/ConferenciaJavierPagesTASSI2013.pdf

https://www.autopsy.com/

https://en.wikipedia.org/wiki/Autopsy

https://en.wikipedia.org/wiki/Plug-in_(computing)

https://www.flu-project.com/2016/09/los-modulos-de-ingesta-de-autopsy-parte.html

ANEXOS

ANEXO I: Caso práctico

A continuación, se pasa a aplicar las técnicas y procedimientos vistos a lo largo de este proyecto a un caso práctico que se podría dar en la vida real.

Como se ha mencionado anteriormente, este tipo de técnicas están orientadas tanto a casos civiles como casos delictivos. Un posible caso civil que se podría dar en la vida real sería la de extracción de archivos (documentos, fotos, etc) importantes para una empresa que se encuentran en el disco duro de un equipo que se ha averiado y que es imposible su reparación sin influir en los archivos que contiene.

Pero, en este apartado se va a realizar un caso práctico orientado a los posibles casos delictivos que se pueden dar en la vida real. Para ello, contamos con el siguiente supuesto:

*En **la madrugada del 19 de mayo de 2018 al 20 de mayo de 2018** se produjo un robo en diversas tiendas del **Centro Comercial La Dehesa** (Alcalá de Henares / Madrid).*
*La única pista que se ha obtenido es una cinta de seguridad en el que se puede ver una furgoneta con **el logotipo de una empresa de alquiler de vehículos** llegando a dicho centro comercial a las 02:31 y saliendo a las 03:12. En las imágenes se intuye que la furgoneta es una **Citroen gris metalizado** y se ha conseguido obtener parte de la matrícula **(- - 1 1 A - -)**.*

Se ha adquirido el ordenador de un sospechoso y se procederá a su análisis para determinar si hay pruebas suficientes como para considerarlo autor del robo.

Para comenzar el análisis forense, se ha realizado una imagen forense del equipo del sospechoso utilizando FTK Imager. Una vez creada la evidencia, se procede a crear un caso y cargar la imagen forense en Autopsy, para proceder a realizar un análisis en estado Postmortem de la evidencia.

Lo primero que se realiza es la ejecución del Plugin *Recient_Activity* (integrado en Autopsy de forma predeterminada) para obtener el historial web completo de la evidencia. Una vez ejecutado, se podrá visualizar los datos obtenidos:

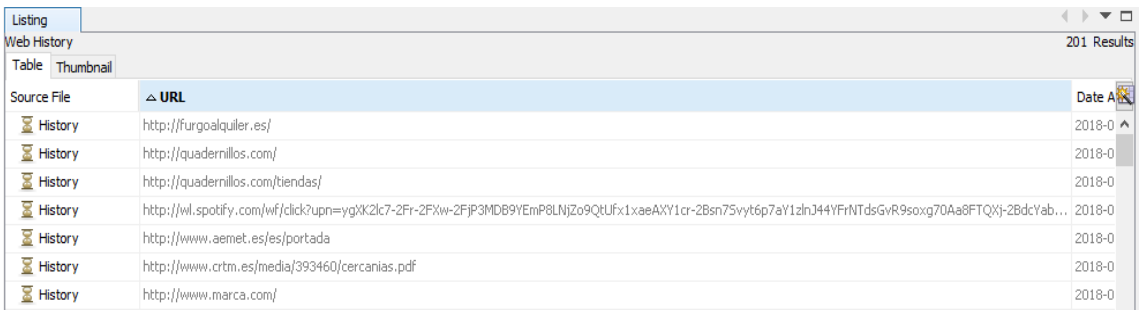

Imagen - 46: Prueba 1

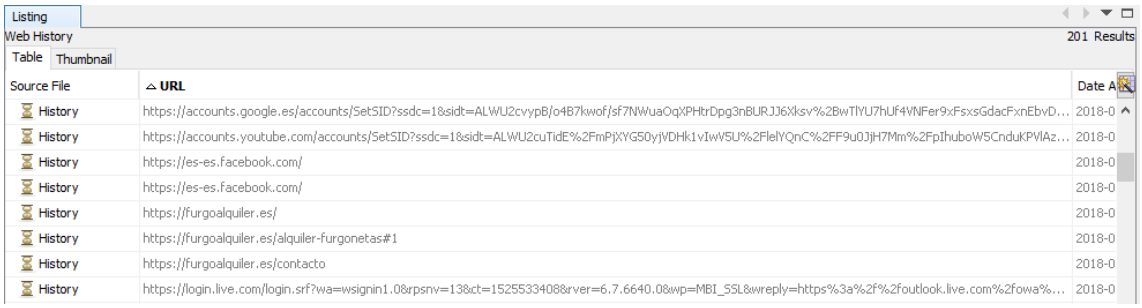

Imagen - 47: Prueba 2

Se puede observar, que el usuario del equipo ha realizado una serie de búsquedas. Entre ellas, se puede ver que se han realizado búsquedas relacionadas con empresas de alquiler de vehículos y una serie de búsquedas relacionadas con el Centro Comercial donde se ha producido el robo.

A continuación, se pasa a buscar archivos relacionados con el crimen, que puedan ser útiles para la investigación y, si es necesario, que demuestren si el sospechoso participó en este delito. Para se analizan los archivos que contiene la evidencia.

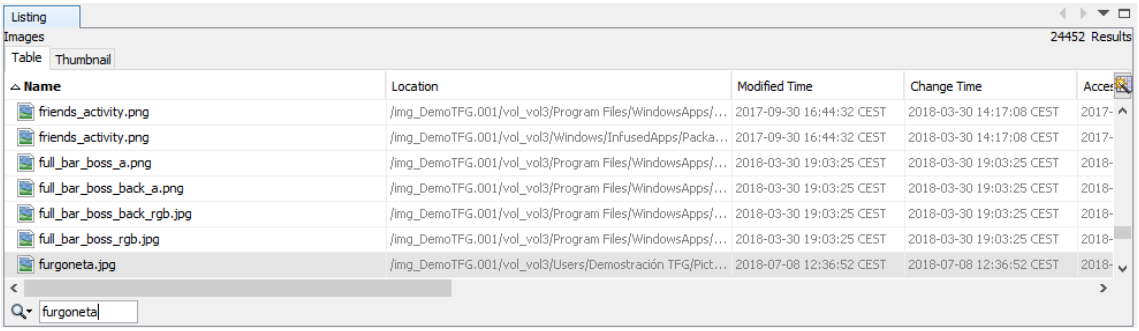

Imagen - 48: Prueba 3

Al realizar una búsqueda con la palabra clave “furgoneta” en los archivos de imágenes, se encuentra una coincidencia. Se extrae esa imagen y se prosigue con la búsqueda de más pruebas. En este caso, se pasa a buscar archivos de texto.

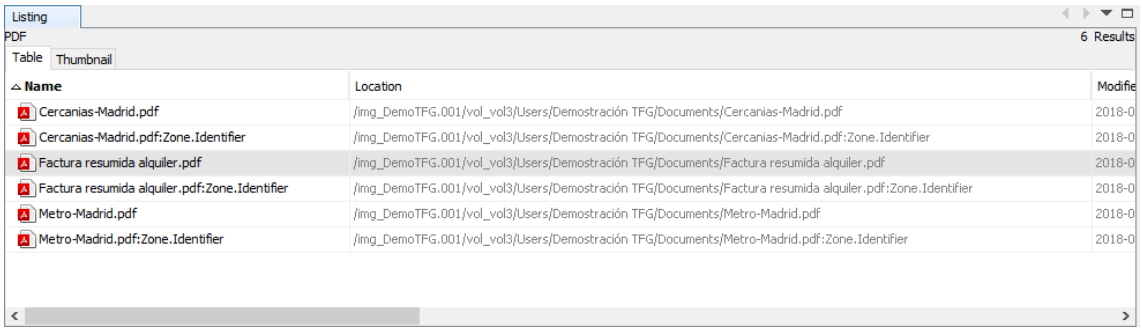

Imagen - 49: Prueba 4

En este punto, se localiza un archivo que llama la atención (Factura resumida alquiler.pdf). Al igual que con la imagen obtenida anteriormente, se procede a su extracción.

Una vez obtenidos los dos archivos, se procede a abrir los archivos con un visualizador externo a Autopsy.

Imagen - 50: Prueba 5

La imagen de la furgoneta coincide con la marca y el color del vehículo que se observa en las imágenes de la cámara de seguridad.

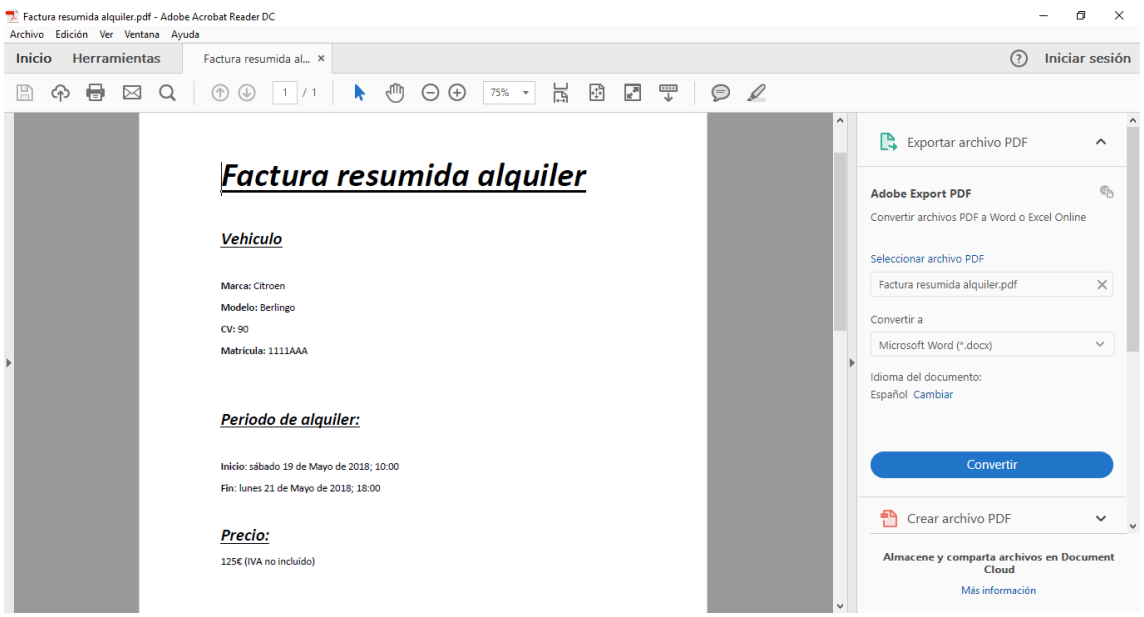

Imagen - 51: Prueba 6

El archivo de texto, muestra los datos generales del vehículo (marca, modelo, cv, matrícula), además del periodo en el que se ha alquilado y el precio total.

La matrícula coincide con la matrícula parcial obtenida por la cámara de seguridad. Además, la hora en la que se produjo el robo coincide con el periodo de alquiler de dicho vehículo.

ANEXO II: Plugins

- ***Amazon_Echosystem_Parser***

```
  def process(self, dataSource, progressBar):

    self.log(Level.INFO, "Starting to process, Just before call to parse_safari_history")

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    self.log(Level.INFO, "Starting 2 to process, Just before call to parse_safari_history")

    skCase = Case.getCurrentCase().getSleuthkitCase();

    head, tail = os.path.split(os.path.abspath(__file__))
    settings_db = head + "\\alexa_db.db3"

    #Start to process based on version of OS
    try:
      Class.forName("org.sqlite.JDBC").newInstance()
      dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % settings_db)
    except SQLException as e:
      self.log(Level.INFO, "Could not open database file (not SQLite) macos_recents.db3 (" + e.getMessage() + ")")
      return IngestModule.ProcessResult.OK

    # Query the database table for unique file names
    try:
      stmt = dbConn.createStatement()
      process_data_sql = "Select distinct file_name from alexa_databases"
      self.log(Level.INFO, process_data_sql)
      resultSet = stmt.executeQuery(process_data_sql)
      self.log(Level.INFO, "Query Database table for unique file names")
    except SQLException as e:
      self.log(Level.INFO, "Error querying database for unique file names")
      return IngestModule.ProcessResult.OK

    # Process all the artifacts based on version of the OS
    while resultSet.next():
       fileManager = Case.getCurrentCase().getServices().getFileManager()
       files = fileManager.findFiles(dataSource, resultSet.getString("file_name"))
       numFiles = len(files)
       self.log(Level.INFO, "found " + str(numFiles) + " files for file_name ==> " + resultSet.getString("file_name"))
       progressBar.switchToDeterminate(numFiles)
       fileCount = 0;

       for file in files:
         # Open the DB using JDBC
         #lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), SQLite_DB)
         lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), file.getName() + "-" + str(file.getId()))
         ContentUtils.writeToFile(file, File(lclDbPath))

         #self.log(Level.INFO, "Path the prefetch database file created ==> " + lclDbPath)
         try:
           Class.forName("org.sqlite.JDBC").newInstance()
           dbConn_x = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
           self.log(Level.INFO, "Database ==> " + file.getName())
         except SQLException as e:
           self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + "-" + str(file.getId()) + " (" +
e.getMessage() + ")")
           #return IngestModule.ProcessResult.OK

         try:
           stmt_sql = dbConn.createStatement()
           process_stmt_sql = "select artifact_name, artifact_description, sql_to_run from alexa_databases where file_name = '" +
resultSet.getString("file_name") + "';"
           self.log(Level.INFO, process_stmt_sql)
           resultSet_sql = stmt_sql.executeQuery(process_stmt_sql)
           self.log(Level.INFO, "Query Database table for sql statements")
         except SQLException as e:
           self.log(Level.INFO, "Error querying database for sql_statements for file " + resultSet.getString("file_name"))
           #return IngestModule.ProcessResult.OK
```

```
        # Process all the artifacts based on version of the OS
        while resultSet_sql.next():

            try:
              stmt_1 = dbConn_x.createStatement()
              sql_to_run = resultSet_sql.getString("sql_to_run")
              self.log(Level.INFO, sql_to_run)
              resultSet_3 = stmt_1.executeQuery(sql_to_run)
              self.log(Level.INFO, "query " + sql_to_run)
            except SQLException as e:
              self.log(Level.INFO, "Error querying database for " + resultSet.getString("file_name"))
              continue
          #return IngestModule.ProcessResult.OK

            try:
              #self.log(Level.INFO, "Begin Create New Artifacts")
              artID_sql = skCase.addArtifactType(resultSet_sql.getString("artifact_name"),
resultSet_sql.getString("artifact_description"))
            except:
              self.log(Level.INFO, "Artifacts Creation Error, for artifact. ==> " + resultSet_sql.getString("artifact_name"))

            artID_hst = skCase.getArtifactTypeID(resultSet_sql.getString("artifact_name"))
            artID_hst_evt = skCase.getArtifactType(resultSet_sql.getString("artifact_name"))

            meta = resultSet_3.getMetaData()
            columncount = meta.getColumnCount()
            column_names = []
            self.log(Level.INFO, "Number of Columns in the table ==> " + str(columncount))
            for x in range (1, columncount + 1):
              self.log(Level.INFO, "Column Name ==> " + meta.getColumnLabel(x))
              try:
                attID_ex1 = skCase.addArtifactAttributeType("TSK_ALEXA_" + meta.getColumnLabel(x).upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, meta.getColumnLabel(x))
              except:
                self.log(Level.INFO, "Attributes Creation Error, " + "TSK_ALEXA_" + meta.getColumnLabel(x) + " ==> ")
              column_names.append(meta.getColumnLabel(x))

            self.log(Level.INFO, "All Columns ==> " + str(column_names))
            # Cycle through each row and create artifacts
            while resultSet_3.next():
              try:
                #self.log(Level.INFO, SQL_String_1)
                self.log(Level.INFO, "Artifact Is ==> " + str(artID_hst))

                art = file.newArtifact(artID_hst)
                self.log(Level.INFO, "Inserting attribute URL")
                for col_name in column_names:
                  attID_ex1 = skCase.getAttributeType("TSK_ALEXA_" + col_name.upper())
                  self.log(Level.INFO, "Inserting attribute ==> " + str(attID_ex1))
                  self.log(Level.INFO, "Attribute Type ==> " + str(attID_ex1.getValueType()))
                  if attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Alexa_DB_ParseIngestModuleFactory.moduleName,
resultSet_3.getString(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes String Creation Error, " + col_name + " ==> ")
                  elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.INTEGER:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Alexa_DB_ParseIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes Integer Creation Error, " + col_name + " ==> ")
                  elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Alexa_DB_ParseIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes Long Creation Error, " + col_name + " ==> ")
                  elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DOUBLE:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Alexa_DB_ParseIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
```

```
                        except:
                            self.log(Level.INFO, "Attributes Double Creation Error, " + col_name + " ==> ")
                    elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.BYTE:
                        try:
                            art.addAttribute(BlackboardAttribute(attID_ex1, Alexa_DB_ParseIngestModuleFactory.moduleName,
resultSet_3.getString(col_name)))
                        except:
                            self.log(Level.INFO, "Attributes Byte Creation Error, " + col_name + " ==> ")
                    else:
                        try:
                            art.addAttribute(BlackboardAttribute(attID_ex1, Alexa_DB_ParseIngestModuleFactory.moduleName,
resultSet_3.getReal(col_name)))
                        except:
                            self.log(Level.INFO, "Attributes Datatime Creation Error, " + col_name + " ==> ")

                except SQLException as e:
                    self.log(Level.INFO, "Error getting values from sql statement ==> " + resultSet_sql.getString("artifact_name"))

                IngestServices.getInstance().fireModuleDataEvent(
                    ModuleDataEvent(Alexa_DB_ParseIngestModuleFactory.moduleName, artID_hst_evt, None))


                stmt_1.close()
            stmt_sql.close()
            dbConn_x.close()


        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "Mac OS Recent Artifacts", " Mac OS Recents Artifacts Have Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK
```

- ***MacFSEvents***

```
  def process(self, dataSource, progressBar):

    self.log(Level.INFO, "Starting to process, Just before call to parse_safari_history")

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    # Get the temp directory and create the sub directory
    Temp_Dir = Case.getCurrentCase().getTempDirectory()
    try:
      os.mkdir(Temp_Dir + "\\MacFSEvents")
    except:
      self.log(Level.INFO, "FSEvents Directory already exists " + Temp_Dir)

    # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "%", ".fseventsd")
    numFiles = len(files)

    for file in files:
      #self.log(Level.INFO, "Files ==> " + file.getName())
      if (file.getName() == "..") or (file.getName() == '.') or (file.getName() == 'fseventsd-uuid'):
        pass
        #self.log(Level.INFO, "Files ==> " + str(file))
      else:
        # Check if the user pressed cancel while we were busy
        if self.context.isJobCancelled():
          return IngestModule.ProcessResult.OK

        # Save the DB locally in the temp folder. use file id as name to reduce collisions
        filePath = os.path.join(Temp_Dir + "\\MacFSEvents", file.getName())
        ContentUtils.writeToFile(file, File(filePath))


    self.log(Level.INFO, "Number of files to process ==> " + str(numFiles))
    self.log(Level.INFO, "Running program ==> " + self.MacFSEvents_Executable + " -c Autopsy " + "-o " + Temp_Dir + \
                " -s " + Temp_Dir + "\\MacFSEvents")
    pipe = Popen([self.MacFSEvents_Executable, "-c", "Autopsy", "-o", Temp_Dir, "-s", Temp_Dir + "\\MacFSEvents"],
stdout=PIPE, stderr=PIPE)
    out_text = pipe.communicate()[0]
    self.log(Level.INFO, "Output from run is ==> " + out_text)

    database_file = Temp_Dir + "\\autopsy_FSEvents-Parsed_Records_DB.sqlite"

    #open the database to get the SQL and artifact info out of
    try:
      head, tail = os.path.split(os.path.abspath(__file__))
      settings_db = head + "\\fsevents_sql.db3"
      Class.forName("org.sqlite.JDBC").newInstance()
      dbConn1 = DriverManager.getConnection("jdbc:sqlite:%s"  % settings_db)
    except SQLException as e:
      self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")
      return IngestModule.ProcessResult.OK

    try:
      stmt1 = dbConn1.createStatement()
      sql_statement1 = "select distinct artifact_name, artifact_title from extracted_content_sql;"
      #self.log(Level.INFO, "SQL Statement ==> " + sql_statement)
      resultSet1 = stmt1.executeQuery(sql_statement1)
      while resultSet1.next():
        try:
          self.log(Level.INFO, "Begin Create New Artifacts")
          artID_fse = skCase.addArtifactType( resultSet1.getString("artifact_name"), resultSet1.getString("artifact_title"))
        except:
          self.log(Level.INFO, "Artifacts Creation Error, " + resultSet1.getString("artifact_name") + " some artifacts may not
exist now. ==> ")

    except SQLException as e:
      self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")
      #return IngestModule.ProcessResult.OK
```

```
        # Create the attribute type, if it exists then catch the error
        try:
            attID_fse_fn = skCase.addArtifactAttributeType("TSK_FSEVENTS_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Name")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Name. ==> ")

        try:
            attID_fse_msk = skCase.addArtifactAttributeType("TSK_FSEVENTS_FILE_MASK",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Mask")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Mask. ==> ")

        try:
            attID_fse_src = skCase.addArtifactAttributeType("TSK_FSEVENTS_SOURCE",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Source File")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Mask. ==> ")

        try:
            attID_fse_dte = skCase.addArtifactAttributeType("TSK_FSEVENTS_DATES",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Date(s)")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Mask. ==> ")

        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % Temp_Dir + "\\autopsy_FSEvents-
Parsed_Records_DB.sqlite")
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        #artID_fse = skCase.getArtifactTypeID("TSK_MACOS_FSEVENTS")
        #artID_fse_evt = skCase.getArtifactType("TSK_MACOS_FSEVENTS")
        artID_fse = skCase.getArtifactTypeID("TSK_MACOS_ALL_FSEVENTS")
        artID_fse_evt = skCase.getArtifactType("TSK_MACOS_ALL_FSEVENTS")
        attID_fse_fn = skCase.getAttributeType("TSK_FSEVENTS_FILE_NAME")
        attID_fse_msk = skCase.getAttributeType("TSK_FSEVENTS_FILE_MASK")
        attID_fse_src = skCase.getAttributeType("TSK_FSEVENTS_SOURCE")
        attID_fse_dte = skCase.getAttributeType("TSK_FSEVENTS_DATES")

        # Query the database
        for file in files:
            if ('slack' in file.getName()):
                pass
            elif (file.getName() == '..') or (file.getName() == '.'):
                pass
            else:
                stmt1 = dbConn1.createStatement()
                sql_statement1 = "select sql_statement, artifact_name, artifact_title from extracted_content_sql;"
                #self.log(Level.INFO, "SQL Statement ==> " + sql_statement)
                resultSet1 = stmt1.executeQuery(sql_statement1)
                while resultSet1.next():
                    try:
                        artID_fse = skCase.getArtifactTypeID(resultSet1.getString("artifact_name"))
                        artID_fse_evt = skCase.getArtifactType(resultSet1.getString("artifact_name"))

                        try:
                            stmt = dbConn.createStatement()
                            sql_statement = resultSet1.getString("sql_statement") + " and source like '%" + file.getName() + "';"
                            #self.log(Level.INFO, "SQL Statement ==> " + sql_statement)
                            resultSet = stmt.executeQuery(sql_statement)
                            #self.log(Level.INFO, "query SQLite Master table ==> " )
                            #self.log(Level.INFO, "query " + str(resultSet))
                            # Cycle through each row and create artifact
                            while resultSet.next():
                            # Add the attributes to the artifact.
                                art = file.newArtifact(artID_fse)
                                #self.log(Level.INFO, "Result ==> " + resultSet.getString("mask") + ' <==> ' + resultSet.getString("source"))
                                art.addAttributes(((BlackboardAttribute(attID_fse_fn, MacFSEventsIngestModuleFactory.moduleName,
resultSet.getString("filename"))), \
                                                   (BlackboardAttribute(attID_fse_msk, MacFSEventsIngestModuleFactory.moduleName,
resultSet.getString("mask"))), \
```

```
                                (BlackboardAttribute(attID_fse_src, MacFSEventsIngestModuleFactory.moduleName,
resultSet.getString("source"))), \
                                (BlackboardAttribute(attID_fse_dte, MacFSEventsIngestModuleFactory.moduleName,
resultSet.getString("OTHER_DATES")))))

                        #try:
                        # index the artifact for keyword search
                          #blackboard.indexArtifact(art)
                        #except:
                          #self.log(Level.INFO, "Error indexing artifact " + art.getDisplayName())

                    except SQLException as e:
                      self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")
                      return IngestModule.ProcessResult.OK
                  except SQLException as e:
                    self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")

            try:
              stmt.close()
            except:
              self.log(Level.INFO, "Error closing statement for " + file.getName())

            # Fire an event to notify the UI and others that there are new artifacts
            IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(MacFSEventsIngestModuleFactory.moduleName,
artID_fse_evt, None))

        try:
            stmt.close()
            dbConn.close()
            stmt1.close()
            dbConn1.close()
            os.remove(Temp_Dir + "\Autopsy_FSEvents-EXCEPTIONS_LOG.txt")
            os.remove(Temp_Dir + "\Autopsy_FSEvents-Parsed_Records.tsv")
            os.remove(Temp_Dir + "\Autopsy_FSEvents-Parsed_Records_DB.sqlite")
            shutil.rmtree(Temp_Dir + "\MacFSEvents")
        except:
            self.log(Level.INFO, "removal of MacFSEvents imageinfo database failed " + Temp_Dir)

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "MacFSEventsSettings", " MacFSEventsSettings Has Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK
```

- ***MacOSX_Recent***

```
def process(self, dataSource, progressBar):

    self.log(Level.INFO, "Starting to process, Just before call to parse_safari_history")

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    self.log(Level.INFO, "Starting 2 to process, Just before call to parse_safari_history")

    head, tail = os.path.split(os.path.abspath(__file__))
    settings_db = head + "\\Macos_recents.db3"

    # Run this first to get the version of the OS to pass to the rest of the program
    self.parse_plist_data(dataSource, progressBar, 'All', 1, settings_db)
    self.log(Level.INFO, "MacOS Version is ===> " + self.os_version + " < == ")

    # get rid of minor revision number
    if self.os_version.count('.') > 1:
      position = 0
      count = 0
      for c in self.os_version:
        position = position + 1
        if c == '.':
          count = count + 1
        if count > 1:
          break
      self.os_version = self.os_version[:position - 1]

    #Start to process based on version of OS
    try:
      Class.forName("org.sqlite.JDBC").newInstance()
      dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % settings_db)
    except SQLException as e:
      self.log(Level.INFO, "Could not open database file (not SQLite) macos_recents.db3 (" + e.getMessage() + ")")
      return IngestModule.ProcessResult.OK

    # Query the artifact table in the database and get all columns.
    try:
      stmt = dbConn.createStatement()
      process_data_sql = "select mac_osx_art_id, mac_osx_art_type, os_version from mac_artifact a, os_version b " + \
                  " where a.os_id = b.os_id and b.os_version = '10.12' and mac_osx_art_id > 1;"
      self.log(Level.INFO, process_data_sql)
      resultSet = stmt.executeQuery(process_data_sql)
      self.log(Level.INFO, "query mac_artifact table")
    except SQLException as e:
      self.log(Level.INFO, "Error querying database for mac_artifact (" + e.getMessage() + ")")
      return IngestModule.ProcessResult.OK

    # Process all the artifacts based on version of the OS
    while resultSet.next():
      if resultSet.getString("mac_osx_art_type") == "Plist":
        self.parse_plist_data(dataSource, progressBar, resultSet.getString("os_version"), resultSet.getString("mac_osx_art_id"), \
                    settings_db)
      else:
        self.parse_sqlite_data(dataSource, progressBar, resultSet.getString("os_version"), resultSet.getString("mac_osx_art_id"), \
                    settings_db)

    self.log(Level.INFO, "MacOS Version is ===> " + self.os_version + " < == ")
    self.log(Level.INFO, "ending process, Just before call to parse_safari_history")

    # After all databases, post a message to the ingest messages in box.
    message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
      "Mac OS Recent Artifacts", " Mac OS Recents Artifacts Have Been Analyzed " )
    IngestServices.getInstance().postMessage(message)

    return IngestModule.ProcessResult.OK


  def parse_plist_data(self, dataSource, progressBar, os_version, mac_os_art_id, settings_db):

    # we don't know how much work there is yet
```

```
        progressBar.switchToIndeterminate()

        skCase = Case.getCurrentCase().getSleuthkitCase();

        try:
          Class.forName("org.sqlite.JDBC").newInstance()
          dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % settings_db)
        except SQLException as e:
          self.log(Level.INFO, "Could not open database file (not SQLite) macos_recents.db3 (" + e.getMessage() + ")")
          return IngestModule.ProcessResult.OK

        # Query the Artifact table in the database and get all columns.
        try:
          stmt = dbConn.createStatement()
          macos_version_sql = "select mac_osx_art_id, mac_osx_art_type, mac_osx_art_File_Name, mac_osx_art_dir_name,
mac_osx_art_exec_file, " + \
                              " mac_osx_art_database_name, mac_osx_art_table_name,  mac_osx_art_sql_statement, os_version, " + \
                              " os_name from mac_artifact a, os_version b where a.os_id = b.os_id and b.os_version = '" + os_version + "'" +
\
                              " and mac_osx_art_id = " + str(mac_os_art_id) + ";"
          self.log(Level.INFO, macos_version_sql)
          resultSet = stmt.executeQuery(macos_version_sql)
          self.log(Level.INFO, "query recent version table")
        except SQLException as e:
          self.log(Level.INFO, "Error querying database for recent version (" + e.getMessage() + ")")
          return IngestModule.ProcessResult.OK

        # get the artifacts to see if they need to be created
        try:
          stmt_2 = dbConn.createStatement()
          artifact_sql = "select distinct autopsy_art_type, autopsy_art_name, autopsy_art_description " + \
                         " from autopsy_artifact a, Art_att_mac_xref b where a.autopsy_art_id = b.autopsy_art_id " + \
                         " and b.mac_osx_art_id = " + resultSet.getString("mac_osx_art_id") + ";"
          resultSet_art = stmt_2.executeQuery(artifact_sql)

          self.log(Level.INFO, "Artifact Type (" + resultSet_art.getString("autopsy_art_type") + ")")

          if resultSet_art.getString("autopsy_art_type") != 'AUTOPSY':
            try:
              self.log(Level.INFO, "Begin Create New Artifacts ==> " + resultSet_art.getString("autopsy_art_name"))
              self.artifact_name = resultSet_art.getString("autopsy_art_name")
              artID_art = skCase.addArtifactType( resultSet_art.getString("autopsy_art_name"), \
                                        resultSet_art.getString("autopsy_art_description"))
            except TskCoreException as ex:
              self.log(Level.INFO, "Artifacts Creation Error, artifact " + resultSet_art.getString("autopsy_art_name") + " exists. ==> "
+ str(ex), ex)
          else:
            self.artifact_name = resultSet_art.getString("autopsy_art_name")

          # Get all the attributes to see if they need to be created
          stmt_3 = dbConn.createStatement()
          attribute_sql = "select distinct autopsy_attrib_type, autopsy_attrib_name, autopsy_attrib_desc,
autopsy_attrib_value_type_desc " + \
                         " from autopsy_attribute a, Art_att_mac_xref b, autopsy_value_type c " + \
                         " where a.autopsy_attrib_id = b.autopsy_attrib_id and a.autopsy_attrib_value_type = c.autopsy_attrib_value_type "
+ \
                         " and b.mac_osx_art_id =" + resultSet.getString("mac_osx_art_id") + ";"
          self.log(Level.INFO, "Attribute SQL ==> " + attribute_sql)
          resultSet_att = stmt_3.executeQuery(attribute_sql)

          while resultSet_att.next():
            if resultSet_att.getString("autopsy_attrib_type")  == 'CUSTOM':
              if resultSet_att.getString("autopsy_attrib_value_type_desc") == 'String':
                try:
                  attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING,
resultSet_att.getString("autopsy_attrib_desc"))
                except:
                  self.log(Level.INFO, "Attributes Creation Error for ," +  resultSet_att.getString("autopsy_attrib_name") + " ==> ")
              elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Integer':
                try:
                  attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.INTEGER,
resultSet_att.getString("autopsy_attrib_desc"))
                except:
                  self.log(Level.INFO, "Attributes Creation Error for ," +  resultSet_att.getString("autopsy_attrib_name") + " ==> ")
```

```
            elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Long':
                try:
                    attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet_att.getString("autopsy_attrib_desc"))
                except:
                    self.log(Level.INFO, "Attributes Creation Error for ," + resultSet_att.getString("autopsy_attrib_name") + " ==> ")
            elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Double':
                try:
                    attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DOUBLE,
resultSet_att.getString("autopsy_attrib_desc"))
                except:
                    self.log(Level.INFO, "Attributes Creation Error for ," + resultSet_att.getString("autopsy_attrib_name") + " ==> ")
            elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Byte':
                try:
                    attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.BYTE, resultSet_att.getString("autopsy_attrib_desc"))
                except:
                    self.log(Level.INFO, "Attributes Creation Error for ," + resultSet_att.getString("autopsy_attrib_name") + " ==> ")
            else:
                try:
                    attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME,
resultSet_att.getString("autopsy_attrib_desc"))
                except:
                    self.log(Level.INFO, "Attributes Creation Error for ," + resultSet_att.getString("autopsy_attrib_name") + " ==> ")
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for artifacts/attributes (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Cycle through each row and create artifacts
        while resultSet.next():

        # Set the database to be read to the once created by the prefetch parser program
            macos_file_name = resultSet.getString("mac_osx_art_File_Name")
            macos_dir_name = resultSet.getString("mac_osx_art_dir_name")
            macos_database_name = resultSet.getString("mac_osx_art_database_name")
            macos_table_name = resultSet.getString("mac_osx_art_table_name")
            self.path_to_plist_exe = os.path.join(os.path.dirname(os.path.abspath(__file__)),
resultSet.getString("mac_osx_art_exec_file"))
            fileManager = Case.getCurrentCase().getServices().getFileManager()
            files = fileManager.findFiles(dataSource, macos_file_name, macos_dir_name)
            numFiles = len(files)
            self.log(Level.INFO, "found " + str(numFiles) + " files")
            progressBar.switchToDeterminate(numFiles)
            fileCount = 0;
            all_files = []

            # for file in files:
                # if file.getParentPath() == macos_dir_name + "/":
                    # self.log(Level.INFO, file.getParentPath())
                    # all_files.append(file)

            # files = all_files

            # Create Event Log directory in temp directory, if it exists then continue on processing
            Temp_Dir = Case.getCurrentCase().getTempDirectory()
            self.log(Level.INFO, "create Directory " + Temp_Dir)
            try:
                os.mkdir(Temp_Dir + "\\macos_recent")
            except:
                self.log(Level.INFO, "macos_recent Directory already exists " + Temp_Dir)

            # Write out each Event Log file to the temp directory
            for file in files:

                self.log(Level.INFO, str(file))

                # Check if the user pressed cancel while we were busy
                if self.context.isJobCancelled():
                    return IngestModule.ProcessResult.OK

                #self.log(Level.INFO, "Processing file: " + file.getName())
                fileCount += 1

                # Save the DB locally in the temp folder. use file id as name to reduce collisions
```

```
            lclDbPath = os.path.join(Temp_Dir + "\\macos_recent", file.getName())
            ContentUtils.writeToFile(file, File(lclDbPath))

            lclDbPath = os.path.join(Temp_Dir + "\\macos_recent", macos_database_name)
            lclFilePath = os.path.join(Temp_Dir + "\\macos_recent", macos_file_name)

            self.log(Level.INFO, "Running prog ==> " + self.path_to_plist_exe + " " + lclFilePath + " " + \
                        lclDbPath + " " + macos_table_name)
            pipe = Popen([self.path_to_plist_exe, lclFilePath, lclDbPath, macos_table_name], stdout=PIPE, stderr=PIPE)

            out_text = pipe.communicate()[0]
            self.log(Level.INFO, "Output from run is ==> " + out_text)

        for file in files:

            # Example has only a Windows EXE, so bail if we aren't on Windows
            if not PlatformUtil.isWindowsOS():
                self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
                return IngestModule.ProcessResult.OK

            # Open the DB using JDBC
            lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory() + "\\macos_recent", macos_database_name)
            self.log(Level.INFO, "Path the Safari History.db database file created ==> " + lclDbPath)
            try:
              Class.forName("org.sqlite.JDBC").newInstance()
              dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
            except SQLException as e:
              self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
              return IngestModule.ProcessResult.OK

            # Query the history_visits table in the database and get all columns.
            try:
              stmt_1 = dbConn.createStatement()
              macos_recent_sql = resultSet.getString("mac_osx_art_sql_statement")
              self.log(Level.INFO, macos_recent_sql)
              resultSet_3 = stmt_1.executeQuery(macos_recent_sql)
              self.log(Level.INFO, "query " + macos_database_name + " table")
            except SQLException as e:
              self.log(Level.INFO, "Error querying database for history table (" + e.getMessage() + ")")
              return IngestModule.ProcessResult.OK

            artID_hst = skCase.getArtifactTypeID(self.artifact_name)
            artID_hst_evt = skCase.getArtifactType(self.artifact_name)

            meta = resultSet_3.getMetaData()
            columncount = meta.getColumnCount()
            column_names = []
            self.log(Level.INFO, "Number of Columns in the table ==> " + str(columncount))
            for x in range (1, columncount + 1):
               self.log(Level.INFO, "Column Name ==> " + meta.getColumnLabel(x))
               column_names.append(meta.getColumnLabel(x))

            self.log(Level.INFO, "All Columns ==> " + str(column_names))
            # Cycle through each row and create artifacts
            while resultSet_3.next():
              try:
                #self.log(Level.INFO, SQL_String_1)
                self.log(Level.INFO, "Artifact Is ==> " + str(artID_hst))

                art = file.newArtifact(artID_hst)
                self.log(Level.INFO, "Inserting attribute URL")
                for col_name in column_names:
                   if ((col_name == "TSK_VERSION") and (mac_os_art_id == 1)):
                      self.os_version = resultSet_3.getString(col_name)
                   attID_ex1 = skCase.getAttributeType(col_name)
                   self.log(Level.INFO, "Inserting attribute ==> " + str(attID_ex1))
                   self.log(Level.INFO, "Attribute Type ==> " + str(attID_ex1.getValueType()))
                   if attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING:
                      try:
                         art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getString(col_name)))
                      except:
                         self.log(Level.INFO, "Attributes String Creation Error, " + col_name + " ==> ")
                   elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.INTEGER:
```

```
                        try:
                            art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                        except:
                            self.log(Level.INFO, "Attributes Integer Creation Error, " + col_name + " ==> ")
                    elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG:
                        try:
                            art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                        except:
                            self.log(Level.INFO, "Attributes Long Creation Error, " + col_name + " ==> ")
                    elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DOUBLE:
                        try:
                            art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                        except:
                            self.log(Level.INFO, "Attributes Double Creation Error, " + col_name + " ==> ")
                    elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.BYTE:
                        try:
                            art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getString(col_name)))
                        except:
                            self.log(Level.INFO, "Attributes Byte Creation Error, " + col_name + " ==> ")
                    else:
                        try:
                            art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getReal(col_name)))
                        except:
                            self.log(Level.INFO, "Attributes Datatime Creation Error, " + col_name + " ==> ")

                except SQLException as e:
                    self.log(Level.INFO, "Error getting values from web_history table (" + e.getMessage() + ")")

                IngestServices.getInstance().fireModuleDataEvent(
                    ModuleDataEvent(ParseMacOS_RecentIngestModuleFactory.moduleName, artID_hst_evt, None))

                stmt_3.close()
                stmt_2.close()
                stmt_1.close()
                stmt.close()
                dbConn.close()

                # Clean up
                os.remove(lclDbPath)
                os.remove(lclFilePath)

            #Clean up EventLog directory and files
            for file in files:
              try:
                os.remove(Temp_Dir + "\\macos_recent\\" + file.getName())
              except:
                self.log(Level.INFO, "removal of Safari History file failed " + Temp_Dir + "\\macos_recent" + file.getName())
            try:
              os.rmdir(Temp_Dir + "\\macos_recent")
            except:
              self.log(Level.INFO, "removal of Safari History directory failed " + Temp_Dir)

        def parse_sqlite_data(self, dataSource, progressBar, os_version, mac_os_art_id, settings_db):

            # we don't know how much work there is yet
            progressBar.switchToIndeterminate()

            skCase = Case.getCurrentCase().getSleuthkitCase();

            try:
              Class.forName("org.sqlite.JDBC").newInstance()
              dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % settings_db)
            except SQLException as e:
              self.log(Level.INFO, "Could not open database file (not SQLite) macos_recents.db3 (" + e.getMessage() + ")")
              return IngestModule.ProcessResult.OK

            # Query the history_visits table in the database and get all columns.
            try:
```

```
        stmt = dbConn.createStatement()
        macos_version_sql = "select mac_osx_art_id, mac_osx_art_type, mac_osx_art_File_Name, mac_osx_art_dir_name, " + \
                            " mac_osx_art_database_name, mac_osx_art_sql_statement, os_version, " + \
                            " os_name from mac_artifact a, os_version b where a.os_id = b.os_id and b.os_version = '" + os_version + "'" + \
                            " and mac_osx_art_id = " + str(mac_os_art_id) + ";"
        self.log(Level.INFO, macos_version_sql)
        resultSet = stmt.executeQuery(macos_version_sql)
        self.log(Level.INFO, "query recent version table")
      except SQLException as e:
        self.log(Level.INFO, "Error querying database for recent version (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

      # Get the artifact name and create it.
      try:
        stmt_2 = dbConn.createStatement()
        artifact_sql = "select distinct autopsy_art_type, autopsy_art_name, autopsy_art_description " + \
                       " from autopsy_artifact a, Art_att_mac_xref b where a.autopsy_art_id = b.autopsy_art_id " + \
                       " and b.mac_osx_art_id = " + resultSet.getString("mac_osx_art_id") + ";"
        resultSet_art = stmt_2.executeQuery(artifact_sql)

        self.log(Level.INFO, "Artifact Type (" + resultSet_art.getString("autopsy_art_type") + ")")

        if resultSet_art.getString("autopsy_art_type") != 'AUTOPSY':
          try:
            self.log(Level.INFO, "Begin Create New Artifacts ==> " + resultSet_art.getString("autopsy_art_name"))
            artID_art = skCase.addArtifactType( resultSet_art.getString("autopsy_art_name"), \
                                    resultSet_art.getString("autopsy_art_desctiption"))
            self.artifact_name = resultSet_art.getString("autopsy_art_name")
          except:
            self.log(Level.INFO, "Artifacts Creation Error, artifact " + resultSet_art.getString("autopsy_art_name") + " exists. ==>
")
        else:
          self.artifact_name = resultSet_art.getString("autopsy_art_name")

        # Get the attribute types and create them
        stmt_3 = dbConn.createStatement()
        attribute_sql = "select distinct autopsy_attrib_type, autopsy_attrib_name, autopsy_attrib_desc,
autopsy_attrib_value_type_desc " + \
                        " from autopsy_attribute a, Art_att_mac_xref b, autopsy_value_type c " + \
                        " where a.autopsy_attrib_id = b.autopsy_attrib_id and a.autopsy_attrib_value_type = c.autopsy_attrib_value_type "
+ \
                        " and b.mac_osx_art_id =" + resultSet.getString("mac_osx_art_id") + ";"
        self.log(Level.INFO, "Attribute SQL ==> " + attribute_sql)
        resultSet_att = stmt_3.executeQuery(attribute_sql)

        while resultSet_att.next():
          if resultSet_att.getString("autopsy_attrib_type")  == 'CUSTOM':
            if resultSet_att.getString("autopsy_attrib_value_type_desc") == 'String':
              try:
                attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING,
resultSet_att.getString("autopsy_attrib_desc"))
              except:
                self.log(Level.INFO, "Attributes Creation Error for ," +  resultSet_att.getString("autopsy_attrib_name") + " ==> ")
            elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Integer':
              try:
                attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.INTEGER,
resultSet_att.getString("autopsy_attrib_desc"))
              except:
                self.log(Level.INFO, "Attributes Creation Error for ," +  resultSet_att.getString("autopsy_attrib_name") + " ==> ")
            elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Long':
              try:
                attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet_att.getString("autopsy_attrib_desc"))
              except:
                self.log(Level.INFO, "Attributes Creation Error for ," +  resultSet_att.getString("autopsy_attrib_name") + " ==> ")
            elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Double':
              try:
                attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DOUBLE,
resultSet_att.getString("autopsy_attrib_desc"))
              except:
                self.log(Level.INFO, "Attributes Creation Error for ," +  resultSet_att.getString("autopsy_attrib_name") + " ==> ")
            elif resultSet_att.getString("autopsy_attrib_value_type_desc") == 'Byte':
```

```
                try:
                  attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.BYTE, resultSet_att.getString("autopsy_attrib_desc"))
                except:
                  self.log(Level.INFO, "Attributes Creation Error for ," + resultSet_att.getString("autopsy_attrib_name") + " ==> ")
              else:
                try:
                  attID_vss_num = skCase.addArtifactAttributeType(resultSet_att.getString("autopsy_attrib_name"),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME,
resultSet_att.getString("autopsy_attrib_desc"))
                except:
                  self.log(Level.INFO, "Attributes Creation Error for ," + resultSet_att.getString("autopsy_attrib_name") + " ==> ")
      except SQLException as e:
        self.log(Level.INFO, "Error querying database for artifacts/attributes (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

      # Cycle through each row and create artifacts
      while resultSet.next():

      # Set the database to be read to the once created by the prefetch parser program
        macos_file_name = resultSet.getString("mac_osx_art_File_Name")
        macos_dir_name = resultSet.getString("mac_osx_art_dir_name")
        macos_database_name = resultSet.getString("mac_osx_art_database_name")
        #macos_table_name = resultSet.getString("mac_osx_art_table_name")
        #self.path_to_plist_exe = os.path.join(os.path.dirname(os.path.abspath(__file__)),
resultSet.getString("mac_osx_art_exec_file"))
        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, macos_file_name + "%", macos_dir_name)
        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;
        all_files = []

        # do this since we want to get the wal or journal files associated with the SQLite database but we want to
        # make sure we have them to use
        if numFiles > 1:
          for file in files:
            if file.getName() == macos_file_name:
              self.log(Level.INFO, file.getParentPath())
              all_files.append(file)

        files_to_process = all_files

        # Create Event Log directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir)
        try:
          os.mkdir(Temp_Dir + "\\macos_recent")
        except:
          self.log(Level.INFO, "macos_recent Directory already exists " + Temp_Dir)

        # Write out each Event Log file to the temp directory
        file_id = 0
        for file in files:

          #self.log(Level.INFO, str(file))

          # Check if the user pressed cancel while we were busy
          if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK

          #self.log(Level.INFO, "Processing file: " + file.getName())
          fileCount += 1

          # Save the DB locally in the temp folder. use file id as name to reduce collisions, also add file id to wal and journal files
          # if needed so that it can use the journals.
          self.log(Level.INFO, "File Name ==> " + file.getName() + " <==> " + macos_database_name)
          if file.getName().upper() == macos_database_name.upper():
            file_id = file.getId()
            self.log(Level.INFO, "File Name ==> " + file.getName() + " <==> " + macos_database_name + " <++> " +
str(file.getId()))
            lclDbPath = os.path.join(Temp_Dir + "\\macos_recent", str(file_id) + "-" + file.getName())
            self.log(Level.INFO, " Database name ==> " + lclDbPath)
            ContentUtils.writeToFile(file, File(lclDbPath))
          else:
```

```
                lclDbPath = os.path.join(Temp_Dir + "\\macos_recent", str(file_id) + "-" + file.getName())
                self.log(Level.INFO, " Database name ==> " + lclDbPath)
                ContentUtils.writeToFile(file, File(lclDbPath))


            lclDbPath = os.path.join(Temp_Dir + "\\macos_recent", str(file_id) + "-" + macos_database_name)
            lclFilePath = os.path.join(Temp_Dir + "\\macos_recent", macos_file_name)
            self.log(Level.INFO, " Database name ==> " + lclDbPath + " File Path ==> " + lclFilePath)

        for file in files_to_process:

            # Example has only a Windows EXE, so bail if we aren't on Windows
            if not PlatformUtil.isWindowsOS():
                self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
                return IngestModule.ProcessResult.OK

            # Open the DB using JDBC
            lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory() + "\\macos_recent", str(file.getId()) + "-" +
macos_database_name)
            self.log(Level.INFO, "Path the Safari History.db database file created ==> " + lclDbPath)
            try:
                Class.forName("org.sqlite.JDBC").newInstance()
                dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
            except SQLException as e:
                self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
                return IngestModule.ProcessResult.OK

            # Query the history_visits table in the database and get all columns.
            try:
                stmt_1 = dbConn.createStatement()
                macos_recent_sql = resultSet.getString("mac_osx_art_sql_statement")
                self.log(Level.INFO, macos_recent_sql)
                resultSet_3 = stmt_1.executeQuery(macos_recent_sql)
                self.log(Level.INFO, "query " + macos_database_name + " table")
            except SQLException as e:
                self.log(Level.INFO, "Error querying database for history table (" + e.getMessage() + ")")
                return IngestModule.ProcessResult.OK

            artID_hst = skCase.getArtifactTypeID(self.artifact_name)
            artID_hst_evt = skCase.getArtifactType(self.artifact_name)

            meta = resultSet_3.getMetaData()
            columncount = meta.getColumnCount()
            column_names = []
            self.log(Level.INFO, "Number of Columns in the table ==> " + str(columncount))
            for x in range (1, columncount + 1):
                self.log(Level.INFO, "Column Name ==> " + meta.getColumnLabel(x))
                column_names.append(meta.getColumnLabel(x))

            self.log(Level.INFO, "All Columns ==> " + str(column_names))
            # Cycle through each row and create artifacts
            while resultSet_3.next():
                try:
                    #self.log(Level.INFO, SQL_String_1)
                    self.log(Level.INFO, "Artifact Is ==> " + str(artID_hst))

                    art = file.newArtifact(artID_hst)
                    self.log(Level.INFO, "Inserting attribute URL")
                    for col_name in column_names:
                        attID_ex1 = skCase.getAttributeType(col_name)
                        self.log(Level.INFO, "Inserting attribute ==> " + str(attID_ex1))
                        self.log(Level.INFO, "Attribute Type ==> " + str(attID_ex1.getValueType()))
                        if attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING:
                            try:
                                art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getString(col_name)))
                            except:
                                self.log(Level.INFO, "Attributes String Creation Error, " + col_name + " ==> ")
                        elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.INTEGER:
                            try:
                                art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                            except:
                                self.log(Level.INFO, "Attributes Integer Creation Error, " + col_name + " ==> ")
```

```
                elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG:
                    try:
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                    except:
                        self.log(Level.INFO, "Attributes Long Creation Error, " + col_name + " ==> ")
                elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DOUBLE:
                    try:
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getInt(col_name)))
                    except:
                        self.log(Level.INFO, "Attributes Double Creation Error, " + col_name + " ==> ")
                elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.BYTE:
                    try:
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getString(col_name)))
                    except:
                        self.log(Level.INFO, "Attributes Byte Creation Error, " + col_name + " ==> ")
                else:
                    try:
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseMacOS_RecentIngestModuleFactory.moduleName,
resultSet_3.getReal(col_name)))
                    except:
                        self.log(Level.INFO, "Attributes Datatime Creation Error, " + col_name + " ==> ")

            except SQLException as e:
                self.log(Level.INFO, "Error getting values from web_history table (" + e.getMessage() + ")")

            IngestServices.getInstance().fireModuleDataEvent(
                ModuleDataEvent(ParseMacOS_RecentIngestModuleFactory.moduleName, artID_hst_evt, None))

            stmt_3.close()
            stmt_2.close()
            stmt_1.close()
            stmt.close()
            dbConn.close()

            # Clean up
            os.remove(lclDbPath)

        #Clean up EventLog directory and files
        for file in files:
          try:
            os.remove(Temp_Dir + "\\macos_recent\\" + file.getName())
          except:
            self.log(Level.INFO, "removal of Safari History file failed " + Temp_Dir + "\\macos_recent" + file.getName())
        try:
          os.rmdir(Temp_Dir + "\\macos_recent")
        except:
          self.log(Level.INFO, "removal of Safari History directory failed " + Temp_Dir)
```

- ***Parse_Plist***

```
def process(self, dataSource, progressBar):

  # we don't know how much work there is yet
  progressBar.switchToIndeterminate()

 # Set the database to be read to the once created by the prefetch parser program
  skCase = Case.getCurrentCase().getSleuthkitCase();
  Temp_Dir = Case.getCurrentCase().getTempDirectory()
  fileManager = Case.getCurrentCase().getServices().getFileManager()
  message_desc = ''
  for Plist_Files in self.List_Of_DBs:
     files = fileManager.findFiles(dataSource, Plist_Files)
     numFiles = len(files)
     self.log(Level.INFO, "found " + str(numFiles) + " files")
     progressBar.switchToDeterminate(numFiles)
     fileCount = 0;

     for file in files:
       # Open the DB using JDBC
       #lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), SQLite_DB)
       lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), file.getName() + "-" + str(file.getId()))
       ContentUtils.writeToFile(file, File(lclDbPath))

       # Run the EXE, saving output to a sqlite database
       self.log(Level.INFO, "Running program ==> " + self.path_to_exe + " " + Temp_Dir + "\\" + \
              file.getName() + "-" + str(file.getId()) + " " + Temp_Dir + "\\Plist_File-" + str(file.getId()) + ".db3 ")
       pipe = Popen([self.path_to_exe, Temp_Dir + "\\" + file.getName() + "-" + str(file.getId()), Temp_Dir + \
               "\\Plist_File-" + str(file.getId()) + ".db3"], stdout=PIPE, stderr=PIPE)
       out_text = pipe.communicate()[0]
       self.log(Level.INFO, "Output from run is ==> " + out_text)

       if 'not a valid Plist' in out_text:
          message_desc = message_desc + "Error Parsing plist file " + file.getName() + ". File not parsed \n"
       else:
          extDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "Plist_File-" + str(file.getId()) + ".db3")

          #self.log(Level.INFO, "Path the sqlite database file created ==> " + lclDbPath)
          try:
             Class.forName("org.sqlite.JDBC").newInstance()
             dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % extDbPath)
             self.log(Level.INFO, "Database ==> " + file.getName())
          except SQLException as e:
             self.log(Level.INFO, "Could not open database file (not SQLite) " + extDbPath + " (" + e.getMessage() + ")")
             #return IngestModule.ProcessResult.OK

          # Query the contacts table in the database and get all columns.
          try:
             stmt = dbConn.createStatement()
             stmt2 = dbConn.createStatement()
             stmt3 = dbConn.createStatement()
             stmt4 = dbConn.createStatement()
             resultSet = stmt.executeQuery("Select tbl_name, type from SQLITE_MASTER where type in ('table','view');")
             #self.log(Level.INFO, "query SQLite Master table")
             #self.log(Level.INFO, "query " + str(resultSet))

             # Cycle through each row and create artifacts
             while resultSet.next():
                try:
                   self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
                   table_name = resultSet.getString("tbl_name")
                   resultSet4  = stmt4.executeQuery("Select count(*) 'NumRows' from " + resultSet.getString("tbl_name") + " ")
#                    while resultSet4.next():
                   row_count = resultSet4.getInt("NumRows")
                   self.log(Level.INFO, " Number of Rows is " + str(row_count) + " ")
                   if row_count >= 1:
                      #self.log(Level.INFO, "Result get information from table " + resultSet.getString("tbl_name") + " ")
                      SQL_String_1 = "Select * from " + table_name + ";"
                      SQL_String_2 = "PRAGMA table_info('" + table_name + "')"
                      #self.log(Level.INFO, SQL_String_1)
                      #self.log(Level.INFO, SQL_String_2)
                      artifact_name = "TSK_" + file.getName()
```

```
                    artifact_desc = "Plist " + file.getName()
                    #self.log(Level.INFO, "Artifact Name ==> " + artifact_name + "  Artifact Desc ==> " + artifact_desc)
                    try:
                        #self.log(Level.INFO, "Begin Create New Artifacts")
                        artID_plist = skCase.addArtifactType( artifact_name, artifact_desc)
                    except:
                        self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

                    artID_plist = skCase.getArtifactTypeID(artifact_name)
                    artID_plist_evt = skCase.getArtifactType(artifact_name)

                    Column_Names = []
                    Column_Types = []
                    resultSet2  = stmt2.executeQuery(SQL_String_2)
                    while resultSet2.next():
                      Column_Names.append(resultSet2.getString("name").upper())
                      Column_Types.append(resultSet2.getString("type").upper())
                      attribute_name = "TSK_PLIST_" + resultSet2.getString("name").upper()
                      #self.log(Level.INFO, "attribure id for " + attribute_name + " == " + resultSet2.getString("type").upper())
                      if resultSet2.getString("type").upper() == "TEXT":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "LONGVARCHAR":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "BLOB":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "REAL":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      else:
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")


                    resultSet3 = stmt3.executeQuery(SQL_String_1)
                    while resultSet3.next():
                      art = file.newArtifact(artID_plist)
                      Column_Number = 1
                      for col_name in Column_Names:
                        #self.log(Level.INFO, "Result get information for column " + Column_Names[Column_Number - 1] + "
")
                        #self.log(Level.INFO, "Result get information for column_number " + str(Column_Number) + " ")
                        #self.log(Level.INFO, "Result get information for column type " + Column_Types[Column_Number - 1]
+ " <== ")
                        c_name = "TSK_PLIST_" + Column_Names[Column_Number - 1]
                        #self.log(Level.INFO, "Attribute Name is " + c_name + " ")
                        attID_ex1 = skCase.getAttributeType(c_name)
                        if Column_Types[Column_Number - 1] == "TEXT":
                          art.addAttribute(BlackboardAttribute(attID_ex1,
ParsePlists2DBDelRecIngestModuleFactory.moduleName, resultSet3.getString(Column_Number)))
                        elif Column_Types[Column_Number - 1] == "":
                          art.addAttribute(BlackboardAttribute(attID_ex1,
ParsePlists2DBDelRecIngestModuleFactory.moduleName, resultSet3.getString(Column_Number)))
```

```
                        elif Column_Types[Column_Number - 1] == "LONGVARCHAR":
                            art.addAttribute(BlackboardAttribute(attID_ex1,
ParsePlists2DBDelRecIngestModuleFactory.moduleName, "BLOBS Not Supported - Look at actual file"))
                        elif Column_Types[Column_Number - 1] == "BLOB":
                            art.addAttribute(BlackboardAttribute(attID_ex1,
ParsePlists2DBDelRecIngestModuleFactory.moduleName, "BLOBS Not Supported - Look at actual file"))
                        elif Column_Types[Column_Number - 1] == "REAL":
                            art.addAttribute(BlackboardAttribute(attID_ex1,
ParsePlists2DBDelRecIngestModuleFactory.moduleName, long(resultSet3.getFloat(Column_Number))))
                        else:
                            art.addAttribute(BlackboardAttribute(attID_ex1,
ParsePlists2DBDelRecIngestModuleFactory.moduleName, long(resultSet3.getInt(Column_Number))))
                        Column_Number = Column_Number + 1


IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(ParsePlists2DBDelRecIngestModuleFactory.moduleName, \
                            artID_plist_evt, None))

                    except SQLException as e:
                        self.log(Level.INFO, "Error getting values from table " +  resultSet.getString("tbl_name") + " (" +
e.getMessage() + ")")
                except SQLException as e:
                    self.log(Level.INFO, "Error querying database " + file.getName() + " (" + e.getMessage() + ")")
                #return IngestModule.ProcessResult.OK
            # Clean up
                stmt.close()
                dbConn.close()
                os.remove(Temp_Dir + "\\Plist_File-" + str(file.getId()) + ".db3")
            os.remove(Temp_Dir + "\\" + file.getName() + "-" + str(file.getId()))


        # After all databases, post a message to the ingest messages in box.
        if len(message_desc) == 0:
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Plist Parser", " Plist files have been parsed  " )
            IngestServices.getInstance().postMessage(message)
        else:
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Plist Parser", message_desc + " Plist files have been parsed with the above files failing  " )
            IngestServices.getInstance().postMessage(message)



        return IngestModule.ProcessResult.OK

class GUI_PSQLiteUISettings(IngestModuleIngestJobSettings):
    serialVersionUID = 1L

    def __init__(self):
        self.flag = False
        self.area = ""

    def getVersionNumber(self):
        return serialVersionUID

    # TODO: Define getters and settings for data you want to store from UI
    def getFlag(self):
        return self.flag

    def setFlag(self, flag):
        self.flag = flag

    def getArea(self):
        return self.area

    def setArea(self, area):
        self.area = area

class GUI_PSQLiteUISettingsPanel(IngestModuleIngestJobSettingsPanel):
    # Note, we can't use a self.settings instance variable.
    # Rather, self.local_settings is used.
    # https://wiki.python.org/jython/UserGuide#javabean-properties
    # Jython Introspector generates a property - 'settings' on the basis
    # of getSettings() defined in this class. Since only getter function
    # is present, it creates a read-only 'settings' property. This auto-
    # generated read-only property overshadows the instance-variable -
```

```
    # 'settings'

    # We get passed in a previous version of the settings so that we can
    # prepopulate the UI
    # TODO: Update this for your UI
    def __init__(self, settings):
        self.local_settings = settings
        self.initComponents()
        self.customizeComponents()

    # TODO: Update this for your UI
    def checkBoxEvent(self, event):
        if self.checkbox.isSelected():
            self.local_settings.setFlag(True)
            self.local_settings.setArea(self.area.getText());
        else:
            self.local_settings.setFlag(False)

    # TODO: Update this for your UI
    def initComponents(self):
        self.setLayout(BoxLayout(self, BoxLayout.Y_AXIS))
        #self.setLayout(GridLayout(0,1))
        self.setAlignmentX(JComponent.LEFT_ALIGNMENT)
        self.panel1 = JPanel()
        self.panel1.setLayout(BoxLayout(self.panel1, BoxLayout.Y_AXIS))
        self.panel1.setAlignmentY(JComponent.LEFT_ALIGNMENT)
        self.checkbox = JCheckBox("Check to activate/deactivate TextArea", actionPerformed=self.checkBoxEvent)
        self.label0 = JLabel(" ")
        self.label1 = JLabel("Input in SQLite DB's in area below,")
        self.label2 = JLabel("seperate values by commas.")
        self.label3 = JLabel("then check the box above.")
        self.label4 = JLabel(" ")
        self.panel1.add(self.checkbox)
        self.panel1.add(self.label0)
        self.panel1.add(self.label1)
        self.panel1.add(self.label2)
        self.panel1.add(self.label3)
        self.panel1.add(self.label4)
        self.add(self.panel1)

        self.area = JTextArea(5,25)
        #self.area.getDocument().addDocumentListener(self.area)
        #self.area.addKeyListener(listener)
        self.area.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0))
        self.pane = JScrollPane()
        self.pane.getViewport().add(self.area)
        #self.pane.addKeyListener(self.area)
        #self.add(self.area)
        self.add(self.pane)



    # TODO: Update this for your UI
    def customizeComponents(self):
        self.checkbox.setSelected(self.local_settings.getFlag())

    # Return the settings used
    def getSettings(self):
        return self.local_settings
```

- ***Parse_SQLite_Databases***

```
  def process(self, dataSource, progressBar):

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

   # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    for SQLite_DB in self.List_Of_DBs:
      files = fileManager.findFiles(dataSource, SQLite_DB)
      numFiles = len(files)
      self.log(Level.INFO, "found " + str(numFiles) + " files")
      progressBar.switchToDeterminate(numFiles)
      fileCount = 0;

      for file in files:
        # Open the DB using JDBC
        #lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), SQLite_DB)
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), file.getName() + "-" + str(file.getId()))
        ContentUtils.writeToFile(file, File(lclDbPath))

        #self.log(Level.INFO, "Path the prefetch database file created ==> " + lclDbPath)
        try:
          Class.forName("org.sqlite.JDBC").newInstance()
          dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
          self.log(Level.INFO, "Database ==> " + file.getName())
        except SQLException as e:
          self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + "-" + str(file.getId()) + " (" +
e.getMessage() + ")")
          #return IngestModule.ProcessResult.OK

        # Query the contacts table in the database and get all columns.
        try:
          stmt = dbConn.createStatement()
          stmt2 = dbConn.createStatement()
          stmt3 = dbConn.createStatement()
          stmt4 = dbConn.createStatement()
          resultSet = stmt.executeQuery("Select tbl_name, type from SQLITE_MASTER where type in ('table','view');")
          #self.log(Level.INFO, "query SQLite Master table")
          #self.log(Level.INFO, "query " + str(resultSet))

          # Cycle through each row and create artifacts
          while resultSet.next():
            try:
              self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
              table_name = resultSet.getString("tbl_name")
              object_type = resultSet.getString("type")
              resultSet4  = stmt4.executeQuery("Select count(*) 'NumRows' from " + resultSet.getString("tbl_name") + " ")
             #while resultSet4.next():
              row_count = resultSet4.getInt("NumRows")
              self.log(Level.INFO, " Number of Rows is " + str(row_count) + " ")
              if row_count >= 1:
                #self.log(Level.INFO, "Result get information from table " + resultSet.getString("tbl_name") + " ")
                SQL_String_1 = "Select * from " + table_name + ";"
                SQL_String_2 = "PRAGMA table_info('" + table_name + "')"
                #self.log(Level.INFO, SQL_String_1)
                #self.log(Level.INFO, SQL_String_2)
                artifact_name = "TSK_" + SQLite_DB.upper() + "_" + table_name.upper()
                artifact_desc = "SQLite Database  " + SQLite_DB.upper() + "  " + object_type.title()  + "  "+ table_name.upper()
                #self.log(Level.INFO, "Artifact Name ==> " + artifact_name + "  Artifact Desc ==> " + artifact_desc)
                try:
                  #self.log(Level.INFO, "Begin Create New Artifacts")
                  artID_sql = skCase.addArtifactType( artifact_name, artifact_desc)
                except:
                  self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

                artID_sql = skCase.getArtifactTypeID(artifact_name)
                artID_sql_evt = skCase.getArtifactType(artifact_name)

                Column_Names = []
                Column_Types = []
```

```
                    resultSet2  = stmt2.executeQuery(SQL_String_2)
                    while resultSet2.next():
                      Column_Names.append(resultSet2.getString("name").upper())
                      Column_Types.append(resultSet2.getString("type").upper())
                      #self.log(Level.INFO, "Add Attribute TSK_" + resultSet2.getString("name").upper() + " ==> " +
resultSet2.getString("name"))
                      ##attID_ex1 = skCase.addAttrType("TSK_" + resultSet2.getString("name").upper(),
resultSet2.getString("name"))
                      ##self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
                      attribute_name = "TSK_" + SQLite_DB + "_" + table_name.upper() + "_" +
resultSet2.getString("name").upper()
                      #self.log(Level.INFO, "attribure id for " + attribute_name + " == " + resultSet2.getString("type").upper())
                      if resultSet2.getString("type").upper() == "TEXT":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "LONGVARCHAR":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "BLOB":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "REAL":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      else:
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")


                    resultSet3 = stmt3.executeQuery(SQL_String_1)
                    while resultSet3.next():
                      art = file.newArtifact(artID_sql)
                      Column_Number = 1
                      for col_name in Column_Names:
                        c_name = "TSK_" + SQLite_DB + "_" + table_name.upper() + "_" + Column_Names[Column_Number - 1]
                        attID_ex1 = skCase.getAttributeType(c_name)
                        if Column_Types[Column_Number - 1] == "TEXT":
                          art.addAttribute(BlackboardAttribute(attID_ex1, ParseSQLiteDBIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                        elif Column_Types[Column_Number - 1] == "":
                          art.addAttribute(BlackboardAttribute(attID_ex1, ParseSQLiteDBIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                        elif Column_Types[Column_Number - 1] == "LONGVARCHAR":
                          art.addAttribute(BlackboardAttribute(attID_ex1, ParseSQLiteDBIngestModuleFactory.moduleName,
"BLOBS Not Supported - Look at actual file"))
                        elif Column_Types[Column_Number - 1] == "BLOB":
                          art.addAttribute(BlackboardAttribute(attID_ex1, ParseSQLiteDBIngestModuleFactory.moduleName,
"BLOBS Not Supported - Look at actual file"))
                        elif Column_Types[Column_Number - 1] == "REAL":
                          art.addAttribute(BlackboardAttribute(attID_ex1, ParseSQLiteDBIngestModuleFactory.moduleName,
long(resultSet3.getFloat(Column_Number))))
                        else:
                          art.addAttribute(BlackboardAttribute(attID_ex1, ParseSQLiteDBIngestModuleFactory.moduleName,
long(resultSet3.getInt(Column_Number))))
                        Column_Number = Column_Number + 1
```

```
IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(ParseSQLiteDBIngestModuleFactory.moduleName, \
                        artID_sql_evt, None))

                except SQLException as e:
                    self.log(Level.INFO, "Error getting values from table " +  resultSet.getString("tbl_name") + " (" + e.getMessage()
+ ")")
            except SQLException as e:
                self.log(Level.INFO, "Error querying database " + file.getName() + " (" + e.getMessage() + ")")

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "SQLite Database Parser", " SQLite Databases have been parsed  " )
        IngestServices.getInstance().postMessage(message)


        return IngestModule.ProcessResult.OK

class GUI_PSQLiteUISettings(IngestModuleIngestJobSettings):
    serialVersionUID = 1L

    def __init__(self):
        self.flag = False
        self.area = ""

    def getVersionNumber(self):
        return serialVersionUID

    # TODO: Define getters and settings for data you want to store from UI
    def getFlag(self):
        return self.flag

    def setFlag(self, flag):
        self.flag = flag

    def getArea(self):
        return self.area

    def setArea(self, area):
        self.area = area

class GUI_PSQLiteUISettingsPanel(IngestModuleIngestJobSettingsPanel):
    # Note, we can't use a self.settings instance variable.
    # Rather, self.local_settings is used.
    # https://wiki.python.org/jython/UserGuide#javabean-properties
    # Jython Introspector generates a property - 'settings' on the basis
    # of getSettings() defined in this class. Since only getter function
    # is present, it creates a read-only 'settings' property. This auto-
    # generated read-only property overshadows the instance-variable -
    # 'settings'

    # We get passed in a previous version of the settings so that we can
    # prepopulate the UI
    # TODO: Update this for your UI
    def __init__(self, settings):
        self.local_settings = settings
        self.initComponents()
        self.customizeComponents()

    # TODO: Update this for your UI
    def checkBoxEvent(self, event):
        if self.checkbox.isSelected():
            self.local_settings.setFlag(True)
            self.local_settings.setArea(self.area.getText());
        else:
            self.local_settings.setFlag(False)

    # TODO: Update this for your UI
    def initComponents(self):
        self.setLayout(BoxLayout(self, BoxLayout.Y_AXIS))
        #self.setLayout(GridLayout(0,1))
        self.setAlignmentX(JComponent.LEFT_ALIGNMENT)
        self.panel1 = JPanel()
        self.panel1.setLayout(BoxLayout(self.panel1, BoxLayout.Y_AXIS))
        self.panel1.setAlignmentY(JComponent.LEFT_ALIGNMENT)
```

```
        self.checkbox = JCheckBox("Check to activate/deactivate TextArea", actionPerformed=self.checkBoxEvent)
        self.label0 = JLabel(" ")
        self.label1 = JLabel("Input in SQLite DB's in area below,")
        self.label2 = JLabel("seperate values by commas.")
        self.label3 = JLabel("then check the box above.")
        self.label4 = JLabel(" ")
        self.panel1.add(self.checkbox)
        self.panel1.add(self.label0)
        self.panel1.add(self.label1)
        self.panel1.add(self.label2)
        self.panel1.add(self.label3)
        self.panel1.add(self.label4)
        self.add(self.panel1)

        self.area = JTextArea(5,25)
        #self.area.getDocument().addDocumentListener(self.area)
        #self.area.addKeyListener(listener)
        self.area.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0))
        self.pane = JScrollPane()
        self.pane.getViewport().add(self.area)
        #self.pane.addKeyListener(self.area)
        #self.add(self.area)
        self.add(self.pane)


    # TODO: Update this for your UI
    def customizeComponents(self):
        self.checkbox.setSelected(self.local_settings.getFlag())

    # Return the settings used
    def getSettings(self):
        return self.local_settings
```

- ***Parse_SQLite_Del_Records***

```
def process(self, dataSource, progressBar):

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

   # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();
    Temp_Dir = Case.getCurrentCase().getTempDirectory()
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    for SQLite_DB in self.List_Of_DBs:
        files = fileManager.findFiles(dataSource, SQLite_DB)
        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        for file in files:
          # Open the DB using JDBC
          #lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), SQLite_DB)
          lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), file.getName() + "-" + str(file.getId()))
          ContentUtils.writeToFile(file, File(lclDbPath))

          # Run the EXE, saving output to a sqlite database
          self.log(Level.INFO, "Running program ==> " + self.path_to_exe + " " + Temp_Dir + "\\" + \
                file.getName() + "-" + str(file.getId()) + " " + Temp_Dir + "\\SQLite_Del_Records-" + str(file.getId()) + ".db3 ")
          pipe = Popen([self.path_to_exe, Temp_Dir + "\\" + file.getName() + "-" + str(file.getId()), Temp_Dir + \
                 "\\SQLite_Del_Records-" + str(file.getId()) + ".db3"], stdout=PIPE, stderr=PIPE)
          out_text = pipe.communicate()[0]
          self.log(Level.INFO, "Output from run is ==> " + out_text)

          extDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "SQLite_Del_Records-" + str(file.getId()) + ".db3")

          #self.log(Level.INFO, "Path the sqlite database file created ==> " + lclDbPath)
          try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % extDbPath)
            self.log(Level.INFO, "Database ==> " + file.getName())
          except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + extDbPath + " (" + e.getMessage() + ")")
            #return IngestModule.ProcessResult.OK

          # Query the contacts table in the database and get all columns.
          try:
            stmt = dbConn.createStatement()
            stmt2 = dbConn.createStatement()
            stmt3 = dbConn.createStatement()
            stmt4 = dbConn.createStatement()
            resultSet = stmt.executeQuery("Select tbl_name, type from SQLITE_MASTER where type in ('table','view');")
            #self.log(Level.INFO, "query SQLite Master table")
            #self.log(Level.INFO, "query " + str(resultSet))

            # Cycle through each row and create artifacts
            while resultSet.next():
              try:
                self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
                table_name = resultSet.getString("tbl_name")
                object_type = resultSet.getString("type")
                resultSet4  = stmt4.executeQuery("Select count(*) 'NumRows' from " + resultSet.getString("tbl_name") + " ")
#                 while resultSet4.next():
                row_count = resultSet4.getInt("NumRows")
                self.log(Level.INFO, " Number of Rows is " + str(row_count) + " ")
                if row_count >= 1:
                  #self.log(Level.INFO, "Result get information from table " + resultSet.getString("tbl_name") + " ")
                  SQL_String_1 = "Select * from " + table_name + ";"
                  SQL_String_2 = "PRAGMA table_info('" + table_name + "')"
                  #self.log(Level.INFO, SQL_String_1)
                  #self.log(Level.INFO, SQL_String_2)
                  artifact_name = "TSK_" + SQLite_DB.upper() + "_" + table_name.upper()
                  artifact_desc = "SQLite Database  " + SQLite_DB.upper() + "  " + object_type.title()  + "  "+ table_name.upper()
                  #self.log(Level.INFO, "Artifact Name ==> " + artifact_name + "  Artifact Desc ==> " + artifact_desc)
                  try:
```

```
                        #self.log(Level.INFO, "Begin Create New Artifacts")
                        artID_sql = skCase.addArtifactType( artifact_name, artifact_desc)
                    except:
                        self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

                    artID_sql = skCase.getArtifactTypeID(artifact_name)
                    artID_sql_evt = skCase.getArtifactType(artifact_name)

                    Column_Names = []
                    Column_Types = []
                    resultSet2  = stmt2.executeQuery(SQL_String_2)
                    while resultSet2.next():
                      Column_Names.append(resultSet2.getString("name").upper())
                      Column_Types.append(resultSet2.getString("type").upper())
                      #self.log(Level.INFO, "Add Attribute TSK_" + resultSet2.getString("name").upper() + " ==> " +
resultSet2.getString("name"))
                      ##attID_ex1 = skCase.addAttrType("TSK_" + resultSet2.getString("name").upper(),
resultSet2.getString("name"))
                      ##self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
                      attribute_name = "TSK_" + SQLite_DB + "_" + table_name.upper() + "_" +
resultSet2.getString("name").upper()
                      #self.log(Level.INFO, "attribure id for " + attribute_name + " == " + resultSet2.getString("type").upper())
                      if resultSet2.getString("type").upper() == "TEXT":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "LONGVARCHAR":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "BLOB":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      elif resultSet2.getString("type").upper() == "REAL":
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                      else:
                        try:
                          attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                          self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")


                    resultSet3 = stmt3.executeQuery(SQL_String_1)
                    while resultSet3.next():
                      art = file.newArtifact(artID_sql)
                      Column_Number = 1
                      for col_name in Column_Names:
                        c_name = "TSK_" + SQLite_DB + "_" + table_name.upper() + "_" + Column_Names[Column_Number - 1]
                        #self.log(Level.INFO, "Attribute Name is " + c_name + " ")
                        attID_ex1 = skCase.getAttributeType(c_name)
                        if Column_Types[Column_Number - 1] == "TEXT":
                          art.addAttribute(BlackboardAttribute(attID_ex1,
ParseSQLiteDBDelRecIngestModuleFactory.moduleName, resultSet3.getString(Column_Number)))
                        elif Column_Types[Column_Number - 1] == "":
                          art.addAttribute(BlackboardAttribute(attID_ex1,
ParseSQLiteDBDelRecIngestModuleFactory.moduleName, resultSet3.getString(Column_Number)))
                        elif Column_Types[Column_Number - 1] == "LONGVARCHAR":
```

```
                        art.addAttribute(BlackboardAttribute(attID_ex1,
ParseSQLiteDBDelRecIngestModuleFactory.moduleName, "BLOBS Not Supported - Look at actual file"))
                    elif Column_Types[Column_Number - 1] == "BLOB":
                        art.addAttribute(BlackboardAttribute(attID_ex1,
ParseSQLiteDBDelRecIngestModuleFactory.moduleName, "BLOBS Not Supported - Look at actual file"))
                    elif Column_Types[Column_Number - 1] == "REAL":
                        art.addAttribute(BlackboardAttribute(attID_ex1,
ParseSQLiteDBDelRecIngestModuleFactory.moduleName, long(resultSet3.getFloat(Column_Number))))
                    else:
                        art.addAttribute(BlackboardAttribute(attID_ex1,
ParseSQLiteDBDelRecIngestModuleFactory.moduleName, long(resultSet3.getInt(Column_Number))))
                    Column_Number = Column_Number + 1


IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(ParseSQLiteDBDelRecIngestModuleFactory.moduleName, \
                    artID_sql_evt, None))

                except SQLException as e:
                    self.log(Level.INFO, "Error getting values from table " +  resultSet.getString("tbl_name") + " (" + e.getMessage()
+ ")")
            except SQLException as e:
                self.log(Level.INFO, "Error querying database " + file.getName() + " (" + e.getMessage() + ")")
                #return IngestModule.ProcessResult.OK
            # Clean up
            stmt.close()
            dbConn.close()
            os.remove(Temp_Dir + "\\" + file.getName() + "-" + str(file.getId()))
            os.remove(Temp_Dir + "\\SQLite_Del_Records-" + str(file.getId()) + ".db3")


        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "SQLite Database Parser", " SQLite Databases have been parsed  " )
        IngestServices.getInstance().postMessage(message)


        return IngestModule.ProcessResult.OK

class GUI_PSQLiteUISettings(IngestModuleIngestJobSettings):
    serialVersionUID = 1L

    def __init__(self):
        self.flag = False
        self.area = ""

    def getVersionNumber(self):
        return serialVersionUID

    # TODO: Define getters and settings for data you want to store from UI
    def getFlag(self):
        return self.flag

    def setFlag(self, flag):
        self.flag = flag

    def getArea(self):
        return self.area

    def setArea(self, area):
        self.area = area

class GUI_PSQLiteUISettingsPanel(IngestModuleIngestJobSettingsPanel):
    # Note, we can't use a self.settings instance variable.
    # Rather, self.local_settings is used.
    # https://wiki.python.org/jython/UserGuide#javabean-properties
    # Jython Introspector generates a property - 'settings' on the basis
    # of getSettings() defined in this class. Since only getter function
    # is present, it creates a read-only 'settings' property. This auto-
    # generated read-only property overshadows the instance-variable -
    # 'settings'

    # We get passed in a previous version of the settings so that we can
    # prepopulate the UI
    # TODO: Update this for your UI
    def __init__(self, settings):
        self.local_settings = settings
```

```
        self.initComponents()
        self.customizeComponents()

    # TODO: Update this for your UI
    def checkBoxEvent(self, event):
        if self.checkbox.isSelected():
            self.local_settings.setFlag(True)
            self.local_settings.setArea(self.area.getText());
        else:
            self.local_settings.setFlag(False)

    # TODO: Update this for your UI
    def initComponents(self):
        self.setLayout(BoxLayout(self, BoxLayout.Y_AXIS))
        #self.setLayout(GridLayout(0,1))
        self.setAlignmentX(JComponent.LEFT_ALIGNMENT)
        self.panel1 = JPanel()
        self.panel1.setLayout(BoxLayout(self.panel1, BoxLayout.Y_AXIS))
        self.panel1.setAlignmentY(JComponent.LEFT_ALIGNMENT)
        self.checkbox = JCheckBox("Check to activate/deactivate TextArea", actionPerformed=self.checkBoxEvent)
        self.label0 = JLabel(" ")
        self.label1 = JLabel("Input in SQLite DB's in area below,")
        self.label2 = JLabel("seperate values by commas.")
        self.label3 = JLabel("then check the box above.")
        self.label4 = JLabel(" ")
        self.panel1.add(self.checkbox)
        self.panel1.add(self.label0)
        self.panel1.add(self.label1)
        self.panel1.add(self.label2)
        self.panel1.add(self.label3)
        self.panel1.add(self.label4)
        self.add(self.panel1)

        self.area = JTextArea(5,25)
        #self.area.getDocument().addDocumentListener(self.area)
        #self.area.addKeyListener(listener)
        self.area.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0))
        self.pane = JScrollPane()
        self.pane.getViewport().add(self.area)
        #self.pane.addKeyListener(self.area)
        #self.add(self.area)
        self.add(self.pane)



    # TODO: Update this for your UI
    def customizeComponents(self):
        self.checkbox.setSelected(self.local_settings.getFlag())

    # Return the settings used
    def getSettings(self):
        return self.local_settings
```

- ***Parse_USNJ***

```
def process(self, dataSource, progressBar):

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

   # Set the database to be read to the once created by the SAM parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "$UsnJrnl:$J", "$Extend")
    numFiles = len(files)
    self.log(Level.INFO, "found " + str(numFiles) + " files")
    progressBar.switchToDeterminate(numFiles)
    fileCount = 0;

    # Create Event Log directory in temp directory, if it exists then continue on processing
    Temp_Dir = Case.getCurrentCase().getTempDirectory()
    self.log(Level.INFO, "create Directory " + Temp_Dir)
    try:
       os.mkdir(Temp_Dir + "/usnj")
    except:
       self.log(Level.INFO, "Usnj Directory already exists " + Temp_Dir)

    for file in files:
      # Check if the user pressed cancel while we were busy
      if self.context.isJobCancelled():
         return IngestModule.ProcessResult.OK
      fileCount += 1

      # Save the DB locally in the temp folder. use file id as name to reduce collisions
      lclDbPath = os.path.join(Temp_Dir + "\\usnj\\", "usnj.txt")
      ContentUtils.writeToFile(file, File(lclDbPath))
      self.log(Level.INFO, "Saved File ==> " + lclDbPath)

      if not PlatformUtil.isWindowsOS():
         self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
         return IngestModule.ProcessResult.OK

      # Run the EXE, saving output to a sqlite database
      self.log(Level.INFO, "Running program ==> " + self.path_to_exe + " " + Temp_Dir + "\\usnj\\usnj.txt" + \
            " " + Temp_Dir + "\\usnj.db3")
      pipe = Popen([self.path_to_exe, Temp_Dir + "\\usnj\\usnj.txt", Temp_Dir + "\\usnj.db3"], stdout=PIPE, stderr=PIPE)
      out_text = pipe.communicate()[0]
      self.log(Level.INFO, "Output from run is ==> " + out_text)

      # Open the DB using JDBC
      lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "usnj.db3")
      self.log(Level.INFO, "Path the system database file created ==> " + lclDbPath)

      try:
         Class.forName("org.sqlite.JDBC").newInstance()
         dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
      except SQLException as e:
         self.log(Level.INFO, "Could not open database file (not SQLite) usnj.db3 (" + e.getMessage() + ")")
         return IngestModule.ProcessResult.OK

      # Query the contacts table in the database and get all columns.
      try:
         stmt = dbConn.createStatement()
         resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER; ")
         self.log(Level.INFO, "query SQLite Master table")
      except SQLException as e:
         self.log(Level.INFO, "Error querying database for system table (" + e.getMessage() + ")")
         return IngestModule.ProcessResult.OK

      try:
          self.log(Level.INFO, "Begin Create New Artifacts")
          artID_usnj = skCase.addArtifactType("TSK_USNJ", "NTFS UsrJrnl entries")
      except:
          self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

      artID_usnj = skCase.getArtifactTypeID("TSK_USNJ")
```

```
        artID_usnj_evt = skCase.getArtifactType("TSK_USNJ")


        # Cycle through each row and create artifacts
        while resultSet.next():
          try:
            self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
            table_name = resultSet.getString("tbl_name")
            SQL_String_1 = "Select * from " + table_name + ";"
            SQL_String_2 = "PRAGMA table_info('" + table_name + "')"

            Column_Names = []
            Column_Types = []
            resultSet2  = stmt.executeQuery(SQL_String_2)
            while resultSet2.next():
              Column_Names.append(resultSet2.getString("name").upper())
              Column_Types.append(resultSet2.getString("type"))
              if resultSet2.getString("type").upper() == "TEXT":
                try:
                  attID_ex1 = skCase.addArtifactAttributeType("TSK_USNJ_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                except:
                  self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
              else:
                try:
                  attID_ex1 = skCase.addArtifactAttributeType("TSK_USNJ_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                except:
                  self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")

            resultSet3 = stmt.executeQuery(SQL_String_1)
            while resultSet3.next():
              art = file.newArtifact(artID_usnj)
              Column_Number = 1
              for col_name in Column_Names:
                c_name = "TSK_USNJ_" + col_name
                attID_ex1 = skCase.getAttributeType(c_name)
                if Column_Types[Column_Number - 1] == "TEXT":
                  art.addAttribute(BlackboardAttribute(attID_ex1, ParseUsnJIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                else:
                  art.addAttribute(BlackboardAttribute(attID_ex1, ParseUsnJIngestModuleFactory.moduleName,
resultSet3.getInt(Column_Number)))
                Column_Number = Column_Number + 1

          except SQLException as e:
            self.log(Level.INFO, "Error getting values from Shimcache table (" + e.getMessage() + ")")

      # Clean up
        stmt.close()
        dbConn.close()
        # Fire an event to notify the UI and others that there are new artifacts
        IngestServices.getInstance().fireModuleDataEvent(
          ModuleDataEvent(ParseUsnJIngestModuleFactory.moduleName, artID_usnj_evt, None))

      #Clean up EventLog directory and files
      os.remove(lclDbPath)
      try:
        os.remove(Temp_Dir + "\\usnj\\usnj.txt")
      except:
        self.log(Level.INFO, "removal of usnj.txt file failed " + Temp_Dir + "\\" + file.getName())
      try:
         os.rmdir(Temp_Dir + "\\usnj")
      except:
         self.log(Level.INFO, "removal of usnj directory failed " + Temp_Dir)

      # After all databases, post a message to the ingest messages in box.
      message = IngestMessage.createMessage(IngestMessage.MessageType.DATA, "Usnj Parser", " Usnj Has Been Analyzed " )
      IngestServices.getInstance().postMessage(message)

      # Fire an event to notify the UI and others that there are new artifacts
      IngestServices.getInstance().fireModuleDataEvent(
        ModuleDataEvent(ParseUsnJIngestModuleFactory.moduleName, artID_usnj_evt, None))

      return IngestModule.ProcessResult.OK
```

- ***ParseAmcache***

```
  def process(self, dataSource, progressBar):

      if len(self.List_Of_tables) < 1:
         message = IngestMessage.createMessage(IngestMessage.MessageType.DATA, "ParseEvtx", " No Amcache tables Selected
to Parse " )
         IngestServices.getInstance().postMessage(message)
         return IngestModule.ProcessResult.ERROR

      # we don't know how much work there is yet
      progressBar.switchToIndeterminate()

     # Set the database to be read to the once created by the prefetch parser program
      skCase = Case.getCurrentCase().getSleuthkitCase();
      fileManager = Case.getCurrentCase().getServices().getFileManager()
      files = fileManager.findFiles(dataSource, "Amcache.hve")
      numFiles = len(files)
      self.log(Level.INFO, "found " + str(numFiles) + " files")
      progressBar.switchToDeterminate(numFiles)
      fileCount = 0;

      # Create Event Log directory in temp directory, if it exists then continue on processing
      Temp_Dir = Case.getCurrentCase().getTempDirectory()
      self.log(Level.INFO, "create Directory " + Temp_Dir)
      try:
         os.mkdir(Temp_Dir + "\Amcache")
      except:
         self.log(Level.INFO, "Amcache Directory already exists " + Temp_Dir)

      # Write out each Event Log file to the temp directory
      for file in files:

         # Check if the user pressed cancel while we were busy
         if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK

         #self.log(Level.INFO, "Processing file: " + file.getName())
         fileCount += 1

         # Save the DB locally in the temp folder. use file id as name to reduce collisions
         lclDbPath = os.path.join(Temp_Dir + "\Amcache", file.getName())
         ContentUtils.writeToFile(file, File(lclDbPath))


      # Example has only a Windows EXE, so bail if we aren't on Windows
      if not PlatformUtil.isWindowsOS():
         self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
         return IngestModule.ProcessResult.OK

      # Run the EXE, saving output to a sqlite database
      self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "\Amcache\Amcache.hve  Parm 2 ==> " +
Temp_Dir + "\Amcache.db3")
      subprocess.Popen([self.path_to_exe, Temp_Dir + "\Amcache\Amcache.hve", Temp_Dir +
"\Amcache.db3"]).communicate()[0]

      for file in files:
        # Open the DB using JDBC
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "Amcache.db3")
        self.log(Level.INFO, "Path the Amcache database file created ==> " + lclDbPath)
        try:
           Class.forName("org.sqlite.JDBC").newInstance()
           dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
           self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
           return IngestModule.ProcessResult.OK

        # Query the contacts table in the database and get all columns.
        for am_table_name in self.List_Of_tables:
           try:
              stmt = dbConn.createStatement()
              resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER where lower(tbl_name) in ('" +
am_table_name + "'); ")
```

```
            # resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER where lower(tbl_name) in
('associated_file_entries', " + \
                                # "'unassociated_programs', 'program_entries'); ")
            self.log(Level.INFO, "query SQLite Master table for " + am_table_name)
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for Prefetch table (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Cycle through each row and create artifacts
        while resultSet.next():
            try:
                self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
                table_name = resultSet.getString("tbl_name")
                #self.log(Level.INFO, "Result get information from table " + resultSet.getString("tbl_name") + " ")
                SQL_String_1 = "Select * from " + table_name + ";"
                SQL_String_2 = "PRAGMA table_info('" + table_name + "')"
                artifact_name = "TSK_" + table_name.upper()
                artifact_desc = "Amcache " + table_name.upper()
                #self.log(Level.INFO, SQL_String_1)
                #self.log(Level.INFO, "Artifact_Name ==> " + artifact_name)
                #self.log(Level.INFO, "Artifact_desc ==> " + artifact_desc)
                #self.log(Level.INFO, SQL_String_2)
                try:
                    self.log(Level.INFO, "Begin Create New Artifacts")
                    artID_amc = skCase.addArtifactType( artifact_name, artifact_desc)
                except:
                    self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

                artID_amc = skCase.getArtifactTypeID(artifact_name)
                artID_amc_evt = skCase.getArtifactType(artifact_name)

                Column_Names = []
                Column_Types = []
                resultSet2  = stmt.executeQuery(SQL_String_2)
                while resultSet2.next():
                  Column_Names.append(resultSet2.getString("name").upper())
                  Column_Types.append(resultSet2.getString("type").upper())
                  #self.log(Level.INFO, "Add Attribute TSK_" + resultSet2.getString("name").upper() + " ==> " +
resultSet2.getString("type"))
                  #self.log(Level.INFO, "Add Attribute TSK_" + resultSet2.getString("name").upper() + " ==> " +
resultSet2.getString("name"))
                  #attID_ex1 = skCase.addAttrType("TSK_" + resultSet2.getString("name").upper(), resultSet2.getString("name"))
                  #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
                  if resultSet2.getString("type").upper() == "TEXT":
                    try:
                      attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                      #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
                    except:
                      self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
                  elif resultSet2.getString("type").upper() == "":
                    try:
                      attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                      #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
                    except:
                      self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
                  else:
                    try:
                      attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                      #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
                    except:
                      self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")


                resultSet3 = stmt.executeQuery(SQL_String_1)
                while resultSet3.next():
                  art = file.newArtifact(artID_amc)
                  Column_Number = 1
                  for col_name in Column_Names:
                    #self.log(Level.INFO, "Result get information for column " + Column_Names[Column_Number - 1] + " ")
                    #self.log(Level.INFO, "Result get information for column_number " + str(Column_Number) + " ")
                    #self.log(Level.INFO, "Result get information for column type " + Column_Types[Column_Number - 1] + " <==
")
                    c_name = "TSK_" + col_name
```

```
                    #self.log(Level.INFO, "Attribute Name is " + c_name + " ")
                    attID_ex1 = skCase.getAttributeType(c_name)
                    if Column_Types[Column_Number - 1] == "TEXT":
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseAmcacheIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
#                       art.addAttribute(BlackboardAttribute(attID_ex1, ParseAmcacheIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                    elif Column_Types[Column_Number - 1] == "":
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseAmcacheIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
#                     elif Column_Types[Column_Number - 1] == "BLOB":
#                       art.addAttribute(BlackboardAttribute(attID_ex1, ParseAmcacheIngestModuleFactory.moduleName, "BLOBS
Not Supported"))
#                     elif Column_Types[Column_Number - 1] == "REAL":
#                       art.addAttribute(BlackboardAttribute(attID_ex1, ParseAmcacheIngestModuleFactory.moduleName,
resultSet3.getFloat(Column_Number)))
                    else:
                        #self.log(Level.INFO, "Value for column type ==> " + str(resultSet3.getInt(Column_Number)) + " <== ")
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseAmcacheIngestModuleFactory.moduleName,
long(resultSet3.getInt(Column_Number))))
                    Column_Number = Column_Number + 1

IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(ParseAmcacheIngestModuleFactory.moduleName,
artID_amc_evt, None))

            except SQLException as e:
                self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

        # Clean up
            stmt.close()
        dbConn.close()
    #os.remove(lclDbPath)

    #Clean up EventLog directory and files
    for file in files:
        try:
            os.remove(Temp_Dir + "\\" + file.getName())
        except:
            self.log(Level.INFO, "removal of Amcache file failed " + Temp_Dir + "\\" + file.getName())
    try:
        os.rmdir(Temp_Dir)
    except:
        self.log(Level.INFO, "removal of Amcache directory failed " + Temp_Dir)

    # After all databases, post a message to the ingest messages in box.
    message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
        "Amcache Parser", " Amcache Has Been Analyzed " )
    IngestServices.getInstance().postMessage(message)

    return IngestModule.ProcessResult.OK
```

- ***ParseEvtx***

```
  def process(self, dataSource, progressBar):

      #Check to see if event logs were selected, if not then send message and error out else process events selected
      self.log(Level.INFO, "List Of Events ==> " + str(self.List_Of_Events) + " <== Number of Events ==> " +
str(len(self.List_Of_Events)))
      if len(self.List_Of_Events) < 1:
         message = IngestMessage.createMessage(IngestMessage.MessageType.DATA, "ParseEvtx", " No Event Logs Selected to
Parse " )
         IngestServices.getInstance().postMessage(message)
         return IngestModule.ProcessResult.ERROR
      else:
         # Check to see if the artifacts exist and if not then create it, also check to see if the attributes
         # exist and if not then create them
         skCase = Case.getCurrentCase().getSleuthkitCase();
         skCase_Tran = skCase.beginTransaction()
         try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_evtx = skCase.addArtifactType( "TSK_EVTX_LOGS", "Windows Event Logs")
         except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
            artID_evtx = skCase.getArtifactTypeID("TSK_EVTX_LOGS")

         try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_evtx_Long = skCase.addArtifactType( "TSK_EVTX_LOGS_LONG", "Windows Event Logs Long Tail Analysis")
         except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
            artID_evtx_Long = skCase.getArtifactTypeID("TSK_EVTX_LOGS")

         try:
            attID_ev_fn = skCase.addArtifactAttributeType("TSK_EVTX_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Log File Name")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Log File Name. ==> ")
         try:
            attID_ev_rc = skCase.addArtifactAttributeType("TSK_EVTX_RECOVERED_RECORD",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Recovered Record")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Recovered Record. ==> ")
         try:
            attID_ev_cn = skCase.addArtifactAttributeType("TSK_EVTX_COMPUTER_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Computer Name")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Computer Name. ==> ")
         try:
            attID_ev_ei = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_IDENTIFIER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "Event Identiifier")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Log File Name. ==> ")
         try:
            attID_ev_eiq = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_IDENTIFIER_QUALIFERS",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Identifier Qualifiers")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Identifier Qualifiers. ==> ")
         try:
            attID_ev_el = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_LEVEL",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Level")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Level. ==> ")
         try:
            attID_ev_oif = skCase.addArtifactAttributeType("TSK_EVTX_OFFSET_IN_FILE",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Offset In File")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Offset In File. ==> ")
         try:
            attID_ev_id = skCase.addArtifactAttributeType("TSK_EVTX_IDENTIFIER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Identifier")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Identifier. ==> ")
         try:
```

```
            attID_ev_sn = skCase.addArtifactAttributeType("TSK_EVTX_SOURCE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Source Name")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Source Name. ==> ")
        try:
            attID_ev_usi = skCase.addArtifactAttributeType("TSK_EVTX_USER_SECURITY_ID",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "User Security ID")
        except:
            self.log(Level.INFO, "Attributes Creation Error, User Security ID. ==> ")
        try:
            attID_ev_et = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_TIME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Time")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Event Time. ==> ")
        try:
            attID_ev_ete = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_TIME_EPOCH",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Time Epoch")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Identifier. ==> ")
        try:
            attID_ev_dt = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_DETAIL_TEXT",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Detail")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Event Detail. ==> ")

        try:
            attID_ev_cnt = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_ID_COUNT",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "Event Id Count")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Event ID Count. ==> ")

        # Get the new artifacts and attributes that were just created
        artID_evtx = skCase.getArtifactTypeID("TSK_EVTX_LOGS")
        artID_evtx_evt = skCase.getArtifactType("TSK_EVTX_LOGS")
        artID_evtx_Long = skCase.getArtifactTypeID("TSK_EVTX_LOGS_LONG")
        artID_evtx_Long_evt = skCase.getArtifactType("TSK_EVTX_LOGS_LONG")
        attID_ev_fn = skCase.getAttributeType("TSK_EVTX_FILE_NAME")
        attID_ev_rc = skCase.getAttributeType("TSK_EVTX_RECOVERED_RECORD")
        attID_ev_cn = skCase.getAttributeType("TSK_EVTX_COMPUTER_NAME")
        attID_ev_ei = skCase.getAttributeType("TSK_EVTX_EVENT_IDENTIFIER")
        attID_ev_eiq = skCase.getAttributeType("TSK_EVTX_EVENT_IDENTIFIER_QUALIFERS")
        attID_ev_el = skCase.getAttributeType("TSK_EVTX_EVENT_LEVEL")
        attID_ev_oif = skCase.getAttributeType("TSK_EVTX_OFFSET_IN_FILE")
        attID_ev_id = skCase.getAttributeType("TSK_EVTX_IDENTIFIER")
        attID_ev_sn = skCase.getAttributeType("TSK_EVTX_SOURCE_NAME")
        attID_ev_usi = skCase.getAttributeType("TSK_EVTX_USER_SECURITY_ID")
        attID_ev_et = skCase.getAttributeType("TSK_EVTX_EVENT_TIME")
        attID_ev_ete = skCase.getAttributeType("TSK_EVTX_EVENT_TIME_EPOCH")
        attID_ev_dt = skCase.getAttributeType("TSK_EVTX_EVENT_DETAIL_TEXT")
        attID_ev_cnt = skCase.getAttributeType("TSK_EVTX_EVENT_ID_COUNT")

        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Find the Windows Event Log Files
        files = []
        fileManager = Case.getCurrentCase().getServices().getFileManager()
        if self.List_Of_Events[0] == 'ALL':
          files = fileManager.findFiles(dataSource, "%.evtx")
        else:
          for eventlog in self.List_Of_Events:
            file_name = fileManager.findFiles(dataSource, eventlog)
            files.extend(file_name)

        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        # Create Event Log directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir)
        try:
          os.mkdir(Temp_Dir + "\EventLogs")
        except:
          self.log(Level.INFO, "Event Log Directory already exists " + Temp_Dir)
```

```
        # Write out each Event Log file to the temp directory
        for file in files:

            # Check if the user pressed cancel while we were busy
            if self.context.isJobCancelled():
                return IngestModule.ProcessResult.OK

            #self.log(Level.INFO, "Processing file: " + file.getName())
            fileCount += 1

            # Save the DB locally in the temp folder. use file id as name to reduce collisions
            lclDbPath = os.path.join(Temp_Dir + "\EventLogs", file.getName())
            ContentUtils.writeToFile(file, File(lclDbPath))


        # Example has only a Windows EXE, so bail if we aren't on Windows
        if not PlatformUtil.isWindowsOS():
            self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
            return IngestModule.ProcessResult.OK

        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "  Parm 2 ==> " + Temp_Dir +
"\EventLogs.db3")
        subprocess.Popen([self.path_to_exe, Temp_Dir + "\EventLogs", Temp_Dir + "\EventLogs.db3"]).communicate()[0]

        # Set the database to be read to the one created by the Event_EVTX program
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "EventLogs.db3")
        self.log(Level.INFO, "Path to the Eventlogs database file created ==> " + lclDbPath)

        # Open the DB using JDBC
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        files = []
        fileManager = Case.getCurrentCase().getServices().getFileManager()
        if self.List_Of_Events[0] == 'ALL':
           files = fileManager.findFiles(dataSource, "%.evtx")
        else:
          for eventlog in self.List_Of_Events:
            file_name = fileManager.findFiles(dataSource, eventlog)
            files.extend(file_name)
            #self.log(Level.INFO, "found " + str(file_name) + " files")
        #self.log(Level.INFO, "found " + str(files) + " files")

        for file in files:
            file_name = file.getName()
            self.log(Level.INFO, "File To process in SQL " + file_name + "  <<======")
            # Query the contacts table in the database and get all columns.
            try:
                stmt = dbConn.createStatement()
                SQL_Statement = "SELECT File_Name, Recovered_Record, Computer_name, Event_Identifier, " + \
                            " Event_Identifier_Qualifiers, Event_Level, Event_offset, Identifier, " + \
                            " Event_source_Name, Event_User_Security_Identifier, Event_Time, " + \
                            " Event_Time_Epoch, Event_Detail_Text FROM Event_Logs where upper(File_Name) = upper('" +
file_name + "')"
                #self.log(Level.INFO, "SQL Statement " + SQL_Statement + "  <<======")
                resultSet = stmt.executeQuery(SQL_Statement)
            except SQLException as e:
                self.log(Level.INFO, "Error querying database for EventLogs table (" + e.getMessage() + ")")
                return IngestModule.ProcessResult.OK

            # Cycle through each row and create artifacts
            while resultSet.next():
                try:
                    #File_Name  = resultSet.getString("File_Name")
                    #Recovered_Record = resultSet.getString("Recovered_Record")
                    Computer_Name = resultSet.getString("Computer_Name")
                    Event_Identifier = resultSet.getInt("Event_Identifier")
                    #Event_Identifier_Qualifiers = resultSet.getString("Event_Identifier_Qualifiers")
                    Event_Level = resultSet.getString("Event_Level")
                    #Event_Offset = resultSet.getString("Event_Offset")
```

```
                #Identifier = resultSet.getString("Identifier")
                Event_Source_Name = resultSet.getString("Event_Source_Name")
                Event_User_Security_Identifier = resultSet.getString("Event_User_Security_Identifier")
                Event_Time = resultSet.getString("Event_Time")
                #Event_Time_Epoch = resultSet.getString("Event_Time_Epoch")
                Event_Detail_Text = resultSet.getString("Event_Detail_Text")
            except SQLException as e:
                self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

            # Make an artifact on the blackboard, TSK_PROG_RUN and give it attributes for each of the fields
            # Make artifact for TSK_EVTX_LOGS
            art = file.newArtifact(artID_evtx)

            art.addAttributes(((BlackboardAttribute(attID_ev_cn, ParseEvtxDbIngestModuleFactory.moduleName,
Computer_Name)), \
                               (BlackboardAttribute(attID_ev_ei, ParseEvtxDbIngestModuleFactory.moduleName, Event_Identifier)), \
                               (BlackboardAttribute(attID_ev_el, ParseEvtxDbIngestModuleFactory.moduleName, Event_Level)), \
                               (BlackboardAttribute(attID_ev_sn, ParseEvtxDbIngestModuleFactory.moduleName,
Event_Source_Name)), \
                               (BlackboardAttribute(attID_ev_usi, ParseEvtxDbIngestModuleFactory.moduleName,
Event_User_Security_Identifier)), \
                               (BlackboardAttribute(attID_ev_et, ParseEvtxDbIngestModuleFactory.moduleName, Event_Time)), \
                               (BlackboardAttribute(attID_ev_dt, ParseEvtxDbIngestModuleFactory.moduleName, Event_Detail_Text))))
            # These attributes may also be added in the future
        try:
            stmt_1 = dbConn.createStatement()
            SQL_Statement_1 = "select event_identifier, file_name, count(*) 'Number_Of_Events'  " + \
                      " FROM Event_Logs where upper(File_Name) = upper('" + file_name + "')" + \
                      " group by event_identifier, file_name order by 3;"
            #self.log(Level.INFO, "SQL Statement " + SQL_Statement_1 + "  <<=====")
            resultSet_1 = stmt_1.executeQuery(SQL_Statement_1)
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for EventLogs table (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Cycle through each row and create artifacts
        while resultSet_1.next():
            try:
                Event_Identifier = resultSet_1.getInt("Event_Identifier")
                Event_ID_Count = resultSet_1.getInt("Number_Of_Events")
            except SQLException as e:
                self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

            # Make an artifact on the blackboard, TSK_PROG_RUN and give it attributes for each of the fields
            # Make artifact for TSK_EVTX_LOGS
            art_1 = file.newArtifact(artID_evtx_Long)

            self.log(Level.INFO, "Type of Object is ==> " + str(type(Event_ID_Count)))

            art_1.addAttributes(((BlackboardAttribute(attID_ev_ei, ParseEvtxDbIngestModuleFactory.moduleName,
Event_Identifier)), \
                               (BlackboardAttribute(attID_ev_cnt, ParseEvtxDbIngestModuleFactory.moduleName, Event_ID_Count))))

    # Fire an event to notify the UI and others that there are new artifacts
    IngestServices.getInstance().fireModuleDataEvent(
        ModuleDataEvent(ParseEvtxDbIngestModuleFactory.moduleName, artID_evtx_evt, None))
    IngestServices.getInstance().fireModuleDataEvent(
        ModuleDataEvent(ParseEvtxDbIngestModuleFactory.moduleName, artID_evtx_Long_evt, None))

    # Clean up
    stmt_1.close()
    stmt.close()
    dbConn.close()
    os.remove(lclDbPath)

    #Clean up EventLog directory and files
    for file in files:
        try:
            os.remove(Temp_Dir + "\\" + file.getName())
        except:
            self.log(Level.INFO, "removal of Event Log file failed " + Temp_Dir + "\\" + file.getName())
    try:
        os.rmdir(Temp_Dir)
    except:
        self.log(Level.INFO, "removal of Event Logs directory failed " + Temp_Dir)
```

```
        # Fire an event to notify the UI and others that there are new artifacts
        IngestServices.getInstance().fireModuleDataEvent(
            ModuleDataEvent(ParseEvtxDbIngestModuleFactory.moduleName, artID_evtx_evt, None))

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "ParseEvtx", " Event Logs have been parsed " )
        IngestServices.getInstance().postMessage(message)

        # Fire an event to notify the UI and others that there are new artifacts
        IngestServices.getInstance().fireModuleDataEvent(
            ModuleDataEvent(ParseEvtxDbIngestModuleFactory.moduleName, artID_evtx_evt, None))

        return IngestModule.ProcessResult.OK
```

- ***Parse_Evtx_By_EventID***

```
  def process(self, dataSource, progressBar):

      #Check to see if event logs were selected, if not then send message and error out else process events selected
      self.log(Level.INFO, "List Of Events ==> " + str(self.List_Of_Events) + " <== Number of Events ==> " +
str(len(self.List_Of_Events)))
      if len(self.List_Of_Events) < 1:
         message = IngestMessage.createMessage(IngestMessage.MessageType.DATA, "ParseEvtx", " No Event Logs Selected to
Parse " )
         IngestServices.getInstance().postMessage(message)
         return IngestModule.ProcessResult.ERROR
      else:
         # Check to see if the artifacts exist and if not then create it, also check to see if the attributes
         # exist and if not then create them
         skCase = Case.getCurrentCase().getSleuthkitCase();
         skCase_Tran = skCase.beginTransaction()
         try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_evtx = skCase.addArtifactType( "TSK_EVTX_LOGS", "Windows Event Logs")
         except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
            artID_evtx = skCase.getArtifactTypeID("TSK_EVTX_LOGS")

         try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_evtx_Long = skCase.addArtifactType( "TSK_EVTX_LOGS_BY_ID", "Windows Event Logs By Event Id")
         except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
            artID_evtx_Long = skCase.getArtifactTypeID("TSK_EVTX_LOGS_BY_ID")

         try:
            attID_ev_fn = skCase.addArtifactAttributeType("TSK_EVTX_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Log File Name")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Log File Name. ==> ")
         try:
            attID_ev_rc = skCase.addArtifactAttributeType("TSK_EVTX_RECOVERED_RECORD",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Recovered Record")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Recovered Record. ==> ")
         try:
            attID_ev_cn = skCase.addArtifactAttributeType("TSK_EVTX_COMPUTER_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Computer Name")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Computer Name. ==> ")
         try:
            attID_ev_ei = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_IDENTIFIER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "Event Identiifier")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Log File Name. ==> ")
         try:
            attID_ev_eiq = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_IDENTIFIER_QUALIFERS",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Identifier Qualifiers")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Identifier Qualifiers. ==> ")
         try:
            attID_ev_el = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_LEVEL",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Level")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Level. ==> ")
         try:
            attID_ev_oif = skCase.addArtifactAttributeType("TSK_EVTX_OFFSET_IN_FILE",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Offset In File")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Event Offset In File. ==> ")
         try:
            attID_ev_id = skCase.addArtifactAttributeType("TSK_EVTX_IDENTIFIER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Identifier")
         except:
            self.log(Level.INFO, "Attributes Creation Error, Identifier. ==> ")
         try:
```

```
            attID_ev_sn = skCase.addArtifactAttributeType("TSK_EVTX_SOURCE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Source Name")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Source Name. ==> ")
        try:
            attID_ev_usi = skCase.addArtifactAttributeType("TSK_EVTX_USER_SECURITY_ID",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "User Security ID")
        except:
            self.log(Level.INFO, "Attributes Creation Error, User Security ID. ==> ")
        try:
            attID_ev_et = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_TIME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Time")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Event Time. ==> ")
        try:
            attID_ev_ete = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_TIME_EPOCH",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Time Epoch")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Identifier. ==> ")
        try:
            attID_ev_dt = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_DETAIL_TEXT",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Event Detail")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Event Detail. ==> ")

        try:
            attID_ev_cnt = skCase.addArtifactAttributeType("TSK_EVTX_EVENT_ID_COUNT",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "Event Id Count")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Event ID Count. ==> ")

        # Get the new artifacts and attributes that were just created
        artID_evtx = skCase.getArtifactTypeID("TSK_EVTX_LOGS")
        artID_evtx_evt = skCase.getArtifactType("TSK_EVTX_LOGS")
        artID_evtx_Long = skCase.getArtifactTypeID("TSK_EVTX_LOGS_BY_ID")
        artID_evtx_Long_evt = skCase.getArtifactType("TSK_EVTX_LOGS_BY_ID")
        attID_ev_fn = skCase.getAttributeType("TSK_EVTX_FILE_NAME")
        attID_ev_rc = skCase.getAttributeType("TSK_EVTX_RECOVERED_RECORD")
        attID_ev_cn = skCase.getAttributeType("TSK_EVTX_COMPUTER_NAME")
        attID_ev_ei = skCase.getAttributeType("TSK_EVTX_EVENT_IDENTIFIER")
        attID_ev_eiq = skCase.getAttributeType("TSK_EVTX_EVENT_IDENTIFIER_QUALIFERS")
        attID_ev_el = skCase.getAttributeType("TSK_EVTX_EVENT_LEVEL")
        attID_ev_oif = skCase.getAttributeType("TSK_EVTX_OFFSET_IN_FILE")
        attID_ev_id = skCase.getAttributeType("TSK_EVTX_IDENTIFIER")
        attID_ev_sn = skCase.getAttributeType("TSK_EVTX_SOURCE_NAME")
        attID_ev_usi = skCase.getAttributeType("TSK_EVTX_USER_SECURITY_ID")
        attID_ev_et = skCase.getAttributeType("TSK_EVTX_EVENT_TIME")
        attID_ev_ete = skCase.getAttributeType("TSK_EVTX_EVENT_TIME_EPOCH")
        attID_ev_dt = skCase.getAttributeType("TSK_EVTX_EVENT_DETAIL_TEXT")
        attID_ev_cnt = skCase.getAttributeType("TSK_EVTX_EVENT_ID_COUNT")


        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Find the Windows Event Log Files
        files = []
        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, "%.evtx")

        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        # Create Event Log directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir)
        try:
            os.mkdir(Temp_Dir + "\EventLogs")
        except:
            self.log(Level.INFO, "Event Log Directory already exists " + Temp_Dir)

        # Write out each Event Log file to the temp directory
        for file in files:
```

```
        # Check if the user pressed cancel while we were busy
        if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK

        #self.log(Level.INFO, "Processing file: " + file.getName())
        fileCount += 1

        # Save the DB locally in the temp folder. use file id as name to reduce collisions
        lclDbPath = os.path.join(Temp_Dir + "\EventLogs", file.getName())
        ContentUtils.writeToFile(file, File(lclDbPath))


    # Example has only a Windows EXE, so bail if we aren't on Windows
    if not PlatformUtil.isWindowsOS():
        self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
        return IngestModule.ProcessResult.OK

    # Run the EXE, saving output to a sqlite database
    self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "  Parm 2 ==> " + Temp_Dir +
"\EventLogs.db3")
    subprocess.Popen([self.path_to_exe, Temp_Dir + "\EventLogs", Temp_Dir + "\EventLogs.db3"]).communicate()[0]

    # Set the database to be read to the one created by the Event_EVTX program
    lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "EventLogs.db3")
    self.log(Level.INFO, "Path to the Eventlogs database file created ==> " + lclDbPath)

    # Open the DB using JDBC
    try:
        Class.forName("org.sqlite.JDBC").newInstance()
        dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
    except SQLException as e:
        self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

    for file in files:
        file_name = file.getName()
        self.log(Level.INFO, "File To process in SQL " + file_name + "  <<======")
        # Query the contacts table in the database and get all columns.
        if self.List_Of_Events[0] != 'ALL':
            try:
                stmt = dbConn.createStatement()
                SQL_Statement = "SELECT File_Name, Recovered_Record, Computer_name, Event_Identifier, " + \
                            " Event_Identifier_Qualifiers, Event_Level, Event_offset, Identifier, " + \
                            " Event_source_Name, Event_User_Security_Identifier, Event_Time, " + \
                            " Event_Time_Epoch, Event_Detail_Text FROM Event_Logs where upper(File_Name) = upper('" +
file_name + "')" + \
                            " and Event_Identifier in ('" + self.Event_Id_List + "');"
                self.log(Level.INFO, "SQL Statement " + SQL_Statement + "  <<======")
                resultSet = stmt.executeQuery(SQL_Statement)
            except SQLException as e:
                self.log(Level.INFO, "Error querying database for EventLogs table (" + e.getMessage() + ")")
                return IngestModule.ProcessResult.OK

            # Cycle through each row and create artifacts
            while resultSet.next():
                try:
                    Computer_Name = resultSet.getString("Computer_Name")
                    Event_Identifier = resultSet.getInt("Event_Identifier")
                    Event_Level = resultSet.getString("Event_Level")
                    Event_Source_Name = resultSet.getString("Event_Source_Name")
                    Event_User_Security_Identifier = resultSet.getString("Event_User_Security_Identifier")
                    Event_Time = resultSet.getString("Event_Time")
                    Event_Detail_Text = resultSet.getString("Event_Detail_Text")
                except SQLException as e:
                    self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

                # Make an artifact on the blackboard, TSK_PROG_RUN and give it attributes for each of the fields
                # Make artifact for TSK_EVTX_LOGS
                art = file.newArtifact(artID_evtx)

                art.addAttributes(((BlackboardAttribute(attID_ev_cn, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Computer_Name)), \
                                (BlackboardAttribute(attID_ev_ei, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_Identifier)), \
                                (BlackboardAttribute(attID_ev_el, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_Level)), \
```

```
                        (BlackboardAttribute(attID_ev_sn, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_Source_Name)), \
                        (BlackboardAttribute(attID_ev_usi, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_User_Security_Identifier)), \
                        (BlackboardAttribute(attID_ev_et, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_Time)), \
                        (BlackboardAttribute(attID_ev_dt, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_Detail_Text))))
            else:
                try:
                    stmt_1 = dbConn.createStatement()
                    SQL_Statement_1 = "select event_identifier, file_name, count(*) 'Number_Of_Events'  " + \
                            " FROM Event_Logs where upper(File_Name) = upper('" + file_name + "')" + \
                            " group by event_identifier, file_name order by 3;"
                    self.log(Level.INFO, "SQL Statement " + SQL_Statement_1 + "  <<=====")
                    resultSet_1 = stmt_1.executeQuery(SQL_Statement_1)
                except SQLException as e:
                    self.log(Level.INFO, "Error querying database for EventLogs table (" + e.getMessage() + ")")
                    return IngestModule.ProcessResult.OK
                self.log(Level.INFO, "This is the to see what the FU is")
                # Cycle through each row and create artifacts
                while resultSet_1.next():
                    try:
                        self.log(Level.INFO, "This is the to see what the FU is 2")
                        Event_Identifier = resultSet_1.getInt("Event_Identifier")
                        Event_ID_Count = resultSet_1.getInt("Number_Of_Events")
                    except SQLException as e:
                        self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

                    self.log(Level.INFO, "This is the to see what the FU is 3")

                    # Make an artifact on the blackboard, TSK_PROG_RUN and give it attributes for each of the fields
                    # Make artifact for TSK_EVTX_LOGS
                    art_1 = file.newArtifact(artID_evtx_Long)

                    self.log(Level.INFO, "Type of Object is ==> " + str(type(Event_ID_Count)))

                    art_1.addAttributes(((BlackboardAttribute(attID_ev_ei, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_Identifier)), \
                        (BlackboardAttribute(attID_ev_cnt, ParseEvtxByEventIDIngestModuleFactory.moduleName,
Event_ID_Count))))

            # Fire an event to notify the UI and others that there are new artifacts
            IngestServices.getInstance().fireModuleDataEvent(
                ModuleDataEvent(ParseEvtxByEventIDIngestModuleFactory.moduleName, artID_evtx_evt, None))
            IngestServices.getInstance().fireModuleDataEvent(
                ModuleDataEvent(ParseEvtxByEventIDIngestModuleFactory.moduleName, artID_evtx_Long_evt, None))

            # Clean up
            try:
                if self.List_Of_Events[0] != 'ALL':
                    stmt.close()
                else:
                    stmt_1.close()
                dbConn.close()
                os.remove(lclDbPath)
            except:
                self.log(Level.INFO, "Error closing the statment, closing the database or removing the file")

            #Clean up EventLog directory and files
            for file in files:
                try:
                    os.remove(Temp_Dir + "\\" + file.getName())
                except:
                    self.log(Level.INFO, "removal of Event Log file failed " + Temp_Dir + "\\" + file.getName())
            try:
                os.rmdir(Temp_Dir)
            except:
                self.log(Level.INFO, "removal of Event Logs directory failed " + Temp_Dir)

            # Fire an event to notify the UI and others that there are new artifacts
            IngestServices.getInstance().fireModuleDataEvent(
                ModuleDataEvent(ParseEvtxByEventIDIngestModuleFactory.moduleName, artID_evtx_evt, None))

            # After all databases, post a message to the ingest messages in box.
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
```

```
            "ParseEvtx", " Event Logs have been parsed " )
        IngestServices.getInstance().postMessage(message)

        # Fire an event to notify the UI and others that there are new artifacts
        IngestServices.getInstance().fireModuleDataEvent(
            ModuleDataEvent(ParseEvtxByEventIDIngestModuleFactory.moduleName, artID_evtx_evt, None))

        return IngestModule.ProcessResult.OK
```

- ***Process_Extract_VSS***

```
  def process(self, dataSource, progressBar):

      self.log(Level.INFO, "Starting to process, Just before call to parse_safari_history")

      # we don't know how much work there is yet
      progressBar.switchToIndeterminate()

      skCase = Case.getCurrentCase().getSleuthkitCase();

      self.log(Level.INFO, "Starting Processing of Image")

      image_names = dataSource.getPaths()
      self.log(Level.INFO, "Image names ==> " + str(image_names[0]))
      image_name = str(image_names[0])

      # Create VSS directory in ModuleOutput directory, if it exists then continue on processing
      Mod_Dir = Case.getCurrentCase().getModulesOutputDirAbsPath()
      Temp_Dir = Case.getCurrentCase().getTempDirectory()
      self.log(Level.INFO, "create Directory " + Temp_Dir)
      vss_output = os.path.join(Mod_Dir, "vss")

      try:
          os.mkdir(vss_output)
      except:
          self.log(Level.INFO, "Vss already exists " + Temp_Dir)

      lclDbPath = os.path.join(vss_output, "vss_extract_info.db3")
      vss_error_log = os.path.join(vss_output, "bad_files.log")

      # Run the Processing/Extraction process
      self.log(Level.INFO, "Running prog ==> " + self.path_to_exe_vss + " " + image_name + " " + lclDbPath + " " + vss_output +
" " + vss_error_log)
      pipe = Popen([self.path_to_exe_vss, image_name, lclDbPath, vss_output, vss_error_log], stdout=PIPE, stderr=PIPE)
      out_text = pipe.communicate()[0]
      self.log(Level.INFO, "Output from run is ==> " + out_text)

      try:
          attID_vs_fn = skCase.addArtifactAttributeType("TSK_VSS_MFT_NUMBER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "MFT Number")
      except:
          self.log(Level.INFO, "Attributes Creation Error, MFT Number. ==> ")
      try:
          attID_vs_ct = skCase.addArtifactAttributeType("TSK_VSS_DATETIME_CHANGED",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "Recovered Record")
      except:
          self.log(Level.INFO, "Attributes Creation Error, changed time. ==> ")
      try:
          attID_vs_sz = skCase.addArtifactAttributeType("TSK_VSS_FILE_SIZE",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "File Size")
      except:
          self.log(Level.INFO, "Attributes Creation Error, Computer Name. ==> ")


      try:
          Class.forName("org.sqlite.JDBC").newInstance()
          dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
      except SQLException as e:
          self.log(Level.INFO, "Could not open database file (not SQLite) " +" (" + e.getMessage() + ")")
          return IngestModule.ProcessResult.OK

      try:
          stmt = dbConn.createStatement()
          SQL_Statement = "select ' - '||vss_identifier||' - '||DATETIME((SUBSTR(vss_create_dttm,1,11)-
11644473600),'UNIXEPOCH') 'VOL_NAME', " + \
                      " vss_num, volume_id, vss_identifier from vss_info;"
          self.log(Level.INFO, "SQL Statement " + SQL_Statement + "  <<======")
          resultSet = stmt.executeQuery(SQL_Statement)
      except SQLException as e:
          self.log(Level.INFO, "Error querying database for EventLogs table (" + e.getMessage() + ")")
          return IngestModule.ProcessResult.OK
```

```
        # Cycle through each row and create artifacts
        while resultSet.next():

            dir_list = []
            vss_identifier = resultSet.getString("vss_identifier")
            vss_num = int(resultSet.getString("vss_num")) - 1
            dir_list.append(vss_output + "\\vss" + str(vss_num))

            services = IngestServices.getInstance()

            progress_updater = ProgressUpdater()
            newDataSources = []

            fileManager = Case.getCurrentCase().getServices().getFileManager()
            skcase_data = Case.getCurrentCase()

            # Get a Unique device id using uuid
            device_id = UUID.randomUUID()
            self.log(Level.INFO, "device id: ==> " + str(device_id))

            skcase_data.notifyAddingDataSource(device_id)

            # Add data source with files
            newDataSource = fileManager.addLocalFilesDataSource(str(device_id), "vss" + str(vss_num) +
resultSet.getString("VOL_NAME"), "", dir_list, progress_updater)

            newDataSources.append(newDataSource.getRootDirectory())

            # Get the files that were added
            files_added = progress_updater.getFiles()

            for file_added in files_added:
                skcase_data.notifyDataSourceAdded(file_added, device_id)

            #skcase.notifyDataSourceAdded(device_id)

            skCse = Case.getCurrentCase().getSleuthkitCase()
            vss_fileManager = Case.getCurrentCase().getServices().getFileManager()
            vss_files = fileManager.findFiles(dataSource, "%" + vss_identifier + "%", "System Volume Information")
            vss_numFiles = len(vss_files)

            for vs in vss_files:
                if vs.getName() in "-slack":
                    pass
            try:
                self.log(Level.INFO, "Begin Create New Artifacts")
                artID_vss = skCase.addArtifactType( "TSK_VS_VOLUME_" + str(vss_num), "vss" + str(vss_num) +
resultSet.getString("VOL_NAME") + " Files")
            except:
                self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
                artID_vss = skCase.getArtifactTypeID("TSK_VS_VOLUME_" + str(vss_num))

            artID_vss = skCase.getArtifactTypeID("TSK_VS_VOLUME_" + str(vss_num))
            artID_vss_evt = skCase.getArtifactType("TSK_VS_VOLUME_" + str(vss_num))
            attID_vs_fn = skCase.getAttributeType("TSK_VSS_MFT_NUMBER")
            attID_vs_ct = skCase.getAttributeType("TSK_VSS_DATETIME_CHANGED")
            attID_vs_sz = skCase.getAttributeType("TSK_VSS_FILE_SIZE")
            attID_vs_nm = skCase.getAttributeType("TSK_NAME")
            attID_vs_pa = skCase.getAttributeType("TSK_PATH")
            attID_vs_md = skCase.getAttributeType("TSK_DATETIME_MODIFIED")
            attID_vs_ad = skCase.getAttributeType("TSK_DATETIME_ACCESSED")
            attID_vs_cr = skCase.getAttributeType("TSK_DATETIME_CREATED")

            for vs_file in vss_files:
                if "-slack" in vs_file.getName():
                    pass
                else:
                    self.log(Level.INFO, "VSS FIles is ==> " + str(vs_file))

                    try:
                        stmt_1 = dbConn.createStatement()
                        SQL_Statement_1 = "select file_name, inode, directory, ctime, mtime, atime, crtime, size " + \
                                          " from vss1_diff where lower(f_type) <> 'dir';"
                        self.log(Level.INFO, "SQL Statement " + SQL_Statement_1 + "  <<=====")
                        resultSet_1 = stmt_1.executeQuery(SQL_Statement_1)
                    except SQLException as e:
```

```
            self.log(Level.INFO, "Error querying database for vss diff tables (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Cycle through each row and create artifacts
        while resultSet_1.next():
            try:
                File_Name  = resultSet_1.getString("file_name")
                Path_Name = resultSet_1.getString("directory")
                MFT_Number = resultSet_1.getString("inode")
                Ctime = resultSet_1.getInt("ctime")
                Mtime = resultSet_1.getInt("mtime")
                Atime = resultSet_1.getInt("atime")
                Crtime = resultSet_1.getInt("crtime")
                File_Size = resultSet_1.getInt("size")
            except SQLException as e:
                self.log(Level.INFO, "Error getting values from vss diff table (" + e.getMessage() + ")")

            # Make an artifact on the blackboard, TSK_PROG_RUN and give it attributes for each of the fields
            # Make artifact for TSK_EVTX_LOGS
            art = vs_file.newArtifact(artID_vss)

            art.addAttributes(((BlackboardAttribute(attID_vs_nm, VSSIngesttModuleFactory.moduleName, File_Name)), \
                        (BlackboardAttribute(attID_vs_fn, VSSIngesttModuleFactory.moduleName, MFT_Number)), \
                        (BlackboardAttribute(attID_vs_pa, VSSIngesttModuleFactory.moduleName, Path_Name)), \
                        (BlackboardAttribute(attID_vs_cr, VSSIngesttModuleFactory.moduleName, Crtime)), \
                        (BlackboardAttribute(attID_vs_md, VSSIngesttModuleFactory.moduleName, Mtime)), \
                        (BlackboardAttribute(attID_vs_ad, VSSIngesttModuleFactory.moduleName, Atime)), \
                        (BlackboardAttribute(attID_vs_ct, VSSIngesttModuleFactory.moduleName, Ctime)), \
                        (BlackboardAttribute(attID_vs_sz, VSSIngesttModuleFactory.moduleName, File_Size))))

        # Fire an event to notify the UI and others that there are new artifacts
        IngestServices.getInstance().fireModuleDataEvent(
            ModuleDataEvent(VSSIngesttModuleFactory.moduleName, artID_vss_evt, None))

    message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
        "Process/Extract VS", " Volume Shadow has been analyzed " )
    IngestServices.getInstance().postMessage(message)

    return IngestModule.ProcessResult.OK
```

- ***Process_Prefetch_Files_V41***

```
def process(self, dataSource, progressBar):

    # Check to see if the artifacts exist and if not then create it, also check to see if the attributes
    # exist and if not then create them
    skCase = Case.getCurrentCase().getSleuthkitCase();
    try:
        self.log(Level.INFO, "Begin Create New Artifacts")
        artID_pf = skCase.addArtifactType( "TSK_PREFETCH", "Windows Prefetch")
    except:
        self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
        artID_pf = skCase.getArtifactTypeID("TSK_PREFETCH")

    # Create the attribute type, if it exists then catch the error
    try:
       attID_pf_fn = skCase.addArtifactAttributeType("TSK_PREFETCH_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Prefetch File Name")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Prefetch File Name. ==> ")

    try:
       attID_pf_an = skCase.addArtifactAttributeType("TSK_PREFETCH_ACTUAL_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Actual File Name")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Actual File Name. ==> ")

    try:
       attID_nr = skCase.addArtifactAttributeType("TSK_PF_RUN_COUNT",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Program Number Runs")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Program Number Runs. ==> ")

    try:
       attID_ex1 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_1",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 1")
    except:
        self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 1. ==> ")

    try:
       attID_ex2 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_2",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 2")
    except:
        self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 2. ==> ")

    try:
       attID_ex3 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_3",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 3")
    except:
        self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 3. ==> ")

    try:
       attID_ex4 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_4",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 4")
    except:
        self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 4 ==> ")

    try:
       attID_ex5 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_5",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 5")
    except:
        self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 5. ==> ")

    try:
       attID_ex6 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_6",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 6")
    except:
        self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 6. ==> ")

    try:
       attID_ex7 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_7",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 7")
    except:
```

```
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 7. ==> ")

        try:
            attID_ex8 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_8",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "PF Execution DTTM 8")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 8 ==> ")

        self.log(Level.INFO, "Get Artifacts after they were created.")
        # Get the new artifacts and attributes that were just created
        artID_pf = skCase.getArtifactTypeID("TSK_PREFETCH")
        artID_pf_evt = skCase.getArtifactType("TSK_PREFETCH")
        attID_pf_fn = skCase.getAttributeType("TSK_PREFETCH_FILE_NAME")
        attID_pf_an = skCase.getAttributeType("TSK_PREFETCH_ACTUAL_FILE_NAME")
        attID_nr = skCase.getAttributeType("TSK_PF_RUN_COUNT")
        attID_ex1 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_1")
        attID_ex2 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_2")
        attID_ex3 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_3")
        attID_ex4 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_4")
        attID_ex5 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_5")
        attID_ex6 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_6")
        attID_ex7 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_7")
        attID_ex8 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_8")

        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Find the prefetch files and the layout.ini file from the /windows/prefetch folder
        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, "%.pf")

        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        # Create prefetch directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory() + "\Prefetch_Files"
        self.log(Level.INFO, "create Directory " + Temp_Dir)
        try:
            os.mkdir(Temp_Dir)
        except:
            self.log(Level.INFO, "Prefetch Directory already exists " + Temp_Dir)

        # Write out each prefetch file to the temp directory
        for file in files:

            # Check if the user pressed cancel while we were busy
            if self.context.isJobCancelled():
                return IngestModule.ProcessResult.OK
            fileCount += 1

            # Save the DB locally in the temp folder. use file id as name to reduce collisions
            lclDbPath = os.path.join(Temp_Dir, file.getName())
            ContentUtils.writeToFile(file, File(lclDbPath))


        # Example has only a Windows EXE, so bail if we aren't on Windows
        if not PlatformUtil.isWindowsOS():
            self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
            return IngestModule.ProcessResult.OK

        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "  Parm 2 ==> " +
Case.getCurrentCase().getTempDirectory())
        subprocess.Popen([self.path_to_exe, Temp_Dir, Case.getCurrentCase().getTempDirectory()]).communicate()[0]

        # Set the database to be read to the once created by the prefetch parser program
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "Autopsy_PF_DB.db3")
        self.log(Level.INFO, "Path the prefetch database file created ==> " + lclDbPath)

        # Open the DB using JDBC
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
```

```
        self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

    # Query the contacts table in the database and get all columns.
    try:
        stmt = dbConn.createStatement()
        resultSet = stmt.executeQuery("Select prefetch_File_Name, actual_File_Name, Number_time_file_run, " +
                                      " Embeded_date_Time_Unix_1, " +
                                      " Embeded_date_Time_Unix_2, " +
                                      " Embeded_date_Time_Unix_3, " +
                                      " Embeded_date_Time_Unix_4, " +
                                      " Embeded_date_Time_Unix_5, " +
                                      " Embeded_date_Time_Unix_6, " +
                                      " Embeded_date_Time_Unix_7, " +
                                      " Embeded_date_Time_Unix_8 " +
                                      " from prefetch_file_info ")
    except SQLException as e:
        self.log(Level.INFO, "Error querying database for Prefetch table (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

    # Cycle through each row and create artifacts
    while resultSet.next():
        try:
            self.log(Level.INFO, "Result (" + resultSet.getString("Prefetch_File_Name") + ")")
            Prefetch_File_Name  = resultSet.getString("Prefetch_File_Name")
            Actual_File_Name = resultSet.getString("Actual_File_Name")
            Number_Of_Runs = resultSet.getString("Number_Time_File_Run")
            Time_1 = resultSet.getString("Embeded_date_Time_Unix_1")
            Time_2 = resultSet.getString("Embeded_date_Time_Unix_2")
            Time_3 = resultSet.getString("Embeded_date_Time_Unix_3")
            Time_4 = resultSet.getString("Embeded_date_Time_Unix_4")
            Time_5 = resultSet.getString("Embeded_date_Time_Unix_5")
            Time_6 = resultSet.getString("Embeded_date_Time_Unix_6")
            Time_7 = resultSet.getString("Embeded_date_Time_Unix_7")
            Time_8 = resultSet.getString("Embeded_date_Time_Unix_8")
        except SQLException as e:
            self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, Prefetch_File_Name)

        for file in files:
            # Make artifact for TSK_PREFETCH,  this can happen when custom attributes are fully supported
            #art = file.newArtifact(artID_pf)
            art = file.newArtifact(artID_pf)

            # Add the attributes to the artifact.
            art.addAttributes(((BlackboardAttribute(attID_pf_fn, ParsePrefetchDbIngestModuleFactory.moduleName,
Prefetch_File_Name)), \
                              (BlackboardAttribute(attID_pf_an, ParsePrefetchDbIngestModuleFactory.moduleName, Actual_File_Name)),
\
                              (BlackboardAttribute(attID_nr, ParsePrefetchDbIngestModuleFactory.moduleName, Number_Of_Runs)), \
                              (BlackboardAttribute(attID_ex1, ParsePrefetchDbIngestModuleFactory.moduleName, Time_1)), \
                              (BlackboardAttribute(attID_ex2, ParsePrefetchDbIngestModuleFactory.moduleName, Time_2)), \
                              (BlackboardAttribute(attID_ex3, ParsePrefetchDbIngestModuleFactory.moduleName, Time_3)), \
                              (BlackboardAttribute(attID_ex4, ParsePrefetchDbIngestModuleFactory.moduleName, Time_4)), \
                              (BlackboardAttribute(attID_ex5, ParsePrefetchDbIngestModuleFactory.moduleName, Time_5)), \
                              (BlackboardAttribute(attID_ex6, ParsePrefetchDbIngestModuleFactory.moduleName, Time_6)), \
                              (BlackboardAttribute(attID_ex7, ParsePrefetchDbIngestModuleFactory.moduleName, Time_7)), \
                              (BlackboardAttribute(attID_ex8, ParsePrefetchDbIngestModuleFactory.moduleName, Time_8))))

    # Fire an event to notify the UI and others that there are new artifacts
    IngestServices.getInstance().fireModuleDataEvent(
        ModuleDataEvent(ParsePrefetchDbIngestModuleFactory.moduleName, artID_pf_evt, None))

    # Clean up
    stmt.close()
    dbConn.close()
    os.remove(lclDbPath)


    #Clean up prefetch directory and files
    for file in files:
        try:
            os.remove(Temp_Dir + "\\" + file.getName())
        except:
```

```
            self.log(Level.INFO, "removal of prefetch file failed " + Temp_Dir + "\\" + file.getName())
        try:
            os.rmdir(Temp_Dir)
        except:
            self.log(Level.INFO, "removal of prefetch directory failed " + Temp_Dir)

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "Prefetch Analyzer", " Prefetch Has Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        # Fire an event to notify the UI and others that there are new artifacts
        IngestServices.getInstance().fireModuleDataEvent(
            ModuleDataEvent(ParsePrefetchDbIngestModuleFactory.moduleName, artID_pf_evt, None))

        return IngestModule.ProcessResult.OK
```

- ***Parse_SRUDB***

```
  def process(self, dataSource, progressBar):

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

   # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "SRUDB.DAT")
    numFiles = len(files)
    self.log(Level.INFO, "found " + str(numFiles) + " files")
    progressBar.switchToDeterminate(numFiles)
    fileCount = 0;

    # Create Event Log directory in temp directory, if it exists then continue on processing
    Temp_Dir = Case.getCurrentCase().getTempDirectory()
    self.log(Level.INFO, "create Directory " + Temp_Dir)
    try:
      os.mkdir(Temp_Dir + "\SRUDB")
    except:
      self.log(Level.INFO, "SRUDB Directory already exists " + Temp_Dir)

    # Write out each Event Log file to the temp directory
    for file in files:

      # Check if the user pressed cancel while we were busy
      if self.context.isJobCancelled():
        return IngestModule.ProcessResult.OK

      #self.log(Level.INFO, "Processing file: " + file.getName())
      fileCount += 1

      # Save the DB locally in the temp folder. use file id as name to reduce collisions
      lclDbPath = os.path.join(Temp_Dir + "\SRUDB", file.getName())
      ContentUtils.writeToFile(file, File(lclDbPath))


    # Example has only a Windows EXE, so bail if we aren't on Windows
    if not PlatformUtil.isWindowsOS():
      self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
      return IngestModule.ProcessResult.OK

    # Run the EXE, saving output to a sqlite database
    self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "  Parm 2 ==> " + Temp_Dir +
"\SRUDB.db3")
    subprocess.Popen([self.path_to_exe, Temp_Dir + "\SRUDB\SRUDB.DAT", Temp_Dir + "\SRUDB.db3"]).communicate()[0]

    for file in files:
      # Open the DB using JDBC
      lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "SRUDB.db3")
      #lclDbPath = "C:\\Users\\Forensic_User\\OneDrive\\Code\\Python_Scripts\\SRUDB\SRUDB.DB3"
      self.log(Level.INFO, "Path the SRUDB database file created ==> " + lclDbPath)
      try:
        Class.forName("org.sqlite.JDBC").newInstance()
        dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
      except SQLException as e:
        self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

      #PSlsit => TSK_PROG_RUN
      #

      # Query the contacts table in the database and get all columns.
      for SR_table_name in self.List_Of_SRUDB:
        try:
          stmt = dbConn.createStatement()
          resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER where lower(tbl_name) in ('" +
SR_table_name + "'); ")
          self.log(Level.INFO, "query SQLite Master table")
        except SQLException as e:
          self.log(Level.INFO, "Error querying database for Prefetch table (" + e.getMessage() + ")")
```

```
        return IngestModule.ProcessResult.OK

      # Cycle through each row and create artifacts
      while resultSet.next():
        try:
          self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
          table_name = resultSet.getString("tbl_name")
          self.log(Level.INFO, "Result get information from table " + resultSet.getString("tbl_name") + " ")
          SQL_String_1 = "Select * from " + table_name + ";"
          SQL_String_2 = "PRAGMA table_info('" + table_name + "')"
          #self.log(Level.INFO, SQL_String_1)
          #self.log(Level.INFO, SQL_String_2)
          artifact_name = "TSK_" + table_name.upper()
          artifact_desc = "System Resource Usage " + table_name.upper()
          try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_amc = skCase.addArtifactType( artifact_name, artifact_desc)
          except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

          artID_sru = skCase.getArtifactTypeID(artifact_name)
          artID_sru_evt = skCase.getArtifactType(artifact_name)

          Column_Names = []
          Column_Types = []
          resultSet2  = stmt.executeQuery(SQL_String_2)
          while resultSet2.next():
            Column_Names.append(resultSet2.getString("name").upper())
            Column_Types.append(resultSet2.getString("type").upper())
            #attID_ex1 = skCase.addAttrType("TSK_" + resultSet2.getString("name").upper(), resultSet2.getString("name"))
            #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
            if resultSet2.getString("type").upper() == "TEXT":
              try:
                attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
              except:
                self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
            elif resultSet2.getString("type").upper() == "":
              try:
                attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
              except:
                self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
            else:
              try:
                attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                #self.log(Level.INFO, "attribure id for " + "TSK_" + resultSet2.getString("name") + " == " + str(attID_ex1))
              except:
                self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")



          resultSet3 = stmt.executeQuery(SQL_String_1)
          while resultSet3.next():
            art = file.newArtifact(artID_sru)
            Column_Number = 1
            for col_name in Column_Names:
              self.log(Level.INFO, "Result get information for column " + Column_Names[Column_Number - 1] + " ")
              self.log(Level.INFO, "Result get information for column_number " + str(Column_Number) + " ")
              c_name = "TSK_" + col_name
              self.log(Level.INFO, "Attribute Name is " + c_name + " ")
              attID_ex1 = skCase.getAttributeType(c_name)
              if Column_Types[Column_Number - 1] == "TEXT":
                art.addAttribute(BlackboardAttribute(attID_ex1, ParseSRUDBIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
              elif Column_Types[Column_Number - 1] == "":
                art.addAttribute(BlackboardAttribute(attID_ex1, ParseSRUDBIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
#             elif Column_Types[Column_Number - 1] == "BLOB":
#               art.addAttribute(BlackboardAttribute(attID_ex1, ParseSRUDBIngestModuleFactory.moduleName, "BLOBS
Not Supported"))
#             elif Column_Types[Column_Number - 1] == "REAL":
```

```
#                       art.addAttribute(BlackboardAttribute(attID_ex1, ParseSRUDBIngestModuleFactory.moduleName,
resultSet3.getFloat(Column_Number)))
                    else:
                        #self.log(Level.INFO, "Value for column type ==> " + str(resultSet3.getInt(Column_Number)) + " <== ")
                        art.addAttribute(BlackboardAttribute(attID_ex1, ParseSRUDBIngestModuleFactory.moduleName,
long(resultSet3.getInt(Column_Number))))
                    Column_Number = Column_Number + 1

                IngestServices.getInstance().fireModuleDataEvent(
                    ModuleDataEvent(ParseSRUDBIngestModuleFactory.moduleName, artID_sru_evt, None))
            except SQLException as e:
                self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

        # Clean up
        os.remove(lclDbPath)

        #Clean up EventLog directory and files
        for file in files:
            try:
                os.remove(Temp_Dir + "\\" + file.getName())
            except:
                self.log(Level.INFO, "removal of SRUDB file failed " + Temp_Dir + "\\" + file.getName())
        try:
            os.rmdir(Temp_Dir)
        except:
            self.log(Level.INFO, "removal of SRUDB directory failed " + Temp_Dir)

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "System Resourse Usage DB", " SRUDB Has Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK
```

- ***Thumbcache_parser***

```
def process(self, dataSource, progressBar):

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

   # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    thumb_files = fileManager.findFiles(dataSource, "thumbcache_%.db", "")
    numFiles = len(thumb_files)
    self.log(Level.INFO, "Number of Thumbs.db files found ==> " + str(numFiles))

    # Create Event Log directory in temp directory, if it exists then continue on processing
    Temp_Dir = Case.getCurrentCase().getModulesOutputDirAbsPath()
    tmp_dir = Case.getCurrentCase().getTempDirectory()
    self.log(Level.INFO, "create Directory " + Temp_Dir)
    try:
       os.mkdir(Temp_Dir + "\Thumbcache")
    except:
       self.log(Level.INFO, "Thumbcache directory already exists " + Temp_Dir)

    for thumb_file in thumb_files:
       if self.context.isJobCancelled():
          return IngestModule.ProcessResult.OK

       self.log(Level.INFO, "Processing file: " + thumb_file.getName())
       #fileCount += 1

       out_dir = os.path.join(Temp_Dir + "\Thumbcache", str(thumb_file.getId()) + "-" + thumb_file.getName())
       try:
          os.mkdir(Temp_Dir + "\Thumbcache\\" + str(thumb_file.getId()) + "-" + thumb_file.getName())
       except:
          self.log(Level.INFO, str(thumb_file.getId()) + "-" + thumb_file.getName() + " Directory already exists " + Temp_Dir)


       # Save the thumbs.DB locally in the ModuleOutput folder. use file id as name to reduce collisions
       lclDbPath = os.path.join(tmp_dir, str(thumb_file.getId()) + "-" + thumb_file.getName())
       ContentUtils.writeToFile(thumb_file, File(lclDbPath))

       # Run thumbs_viewer against the selected Database
       self.log(Level.INFO, "Running prog ==> " + self.path_to_exe_thumbs + " -O " + out_dir + " " + lclDbPath)
       pipe = Popen([self.path_to_exe_thumbs, "-O", out_dir, lclDbPath], stdout=PIPE, stderr=PIPE)
       out_text = pipe.communicate()[0]
       self.log(Level.INFO, "Output from run is ==> " + out_text)

       # Get the parent abstract file Information
       abstract_file_info = skCase.getAbstractFileById(thumb_file.getId())
       #self.log(Level.INFO, "Abstract File Info ==> " + str(abstract_file_info))

       files = next(os.walk(out_dir))[2]
       for file in files:
          self.log(Level.INFO, " File Name is ==> " + file)

          dev_file = os.path.join(out_dir, file)
          local_file = os.path.join("ModuleOutput\\thumbcache\\" + str(thumb_file.getId()) + "-" + thumb_file.getName(), file)
          self.log(Level.INFO, " Dev File Name is ==> " + dev_file)
          self.log(Level.INFO, " Local File Name is ==> " + local_file)

          if not(self.check_dervived_existance(dataSource, file, abstract_file_info)):

             # Add dervived file
             # Parameters Are:
             #    File Name, Local Path, size, ctime, crtime, atime, mtime, isFile, Parent File, rederive Details, Tool Name,
             #     Tool Version, Other Details, Encoding Type
             dervived_file = skCase.addDerivedFile(file, local_file, os.path.getsize(dev_file), + \
                                0, 0, 0, 0, True, abstract_file_info, "", "thumbcache_viewer_cmd.exe", "1.0.3.4", "",
TskData.EncodingType.NONE)
             #self.log(Level.INFO, "Derived File ==> " + str(dervived_file))
          else:
             pass
```

```
            try:
              os.remove(lclDbPath)
            except:
              self.log(Level.INFO, "removal of thumbcache file " + lclDbPath + " failed " )

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "Thumbcache", " Thumbcache Files Have Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK

    def check_dervived_existance(self, dataSource, file_name, parent_file_abstract):

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        dervived_file = fileManager.findFiles(dataSource, file_name, parent_file_abstract)
        numFiles = len(dervived_file)

        if numFiles == 0:
            return True
        else:
            return False
```

- ***Thumb_parser***

```
def process(self, dataSource, progressBar):

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

   # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase()
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    thumb_files = fileManager.findFiles(dataSource, "thumbs.db", "")
    numFiles = len(thumb_files)
    self.log(Level.INFO, "Number of Thumbs.db files found ==> " + str(numFiles))

    # Create Event Log directory in temp directory, if it exists then continue on processing
    Temp_Dir = Case.getCurrentCase().getModulesOutputDirAbsPath()
    tmp_dir = Case.getCurrentCase().getTempDirectory()
    self.log(Level.INFO, "create Directory " + Temp_Dir)
    try:
        os.mkdir(Temp_Dir + "\Thumbs.db")
    except:
        self.log(Level.INFO, "Thumbs.db Directory already exists " + Temp_Dir)

    for thumb_file in thumb_files:
        if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK

        self.log(Level.INFO, "Processing file: " + thumb_file.getName())
        #fileCount += 1

        out_dir = os.path.join(Temp_Dir + "\Thumbs.db", str(thumb_file.getId()) + "-" + thumb_file.getName())
        try:
            os.mkdir(Temp_Dir + "\Thumbs.db\\" + str(thumb_file.getId()) + "-" + thumb_file.getName())
        except:
            self.log(Level.INFO, str(thumb_file.getId()) + "-" + thumb_file.getName() + " Directory already exists " + Temp_Dir)


        # Save the thumbs.DB locally in the ModuleOutput folder. use file id as name to reduce collisions
        lclDbPath = os.path.join(tmp_dir, str(thumb_file.getId()) + "-" + thumb_file.getName())
        ContentUtils.writeToFile(thumb_file, File(lclDbPath))

        # Run thumbs_viewer against the selected Database
        self.log(Level.INFO, "Running prog ==> " + self.path_to_exe_thumbs + " -O " + out_dir + " " + lclDbPath)
        pipe = Popen([self.path_to_exe_thumbs, "-O", out_dir, lclDbPath], stdout=PIPE, stderr=PIPE)
        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Get the parent abstract file Information
        abstract_file_info = skCase.getAbstractFileById(thumb_file.getId())
        #self.log(Level.INFO, "Abstract File Info ==> " + str(abstract_file_info))

        files = next(os.walk(out_dir))[2]
        for file in files:
            self.log(Level.INFO, " File Name is ==> " + file)

            dev_file = os.path.join(out_dir, file)
            local_file = os.path.join("ModuleOutput\\thumbs.db\\" + str(thumb_file.getId()) + "-" + thumb_file.getName(), file)
            self.log(Level.INFO, " Dev File Name is ==> " + dev_file)
            self.log(Level.INFO, " Local File Name is ==> " + local_file)

            if not(self.check_dervived_existance(dataSource, file, abstract_file_info)):

                # Add dervived file
                # Parameters Are:
                #    File Name, Local Path, size, ctime, crtime, atime, mtime, isFile, Parent File, rederive Details, Tool Name,
                #     Tool Version, Other Details, Encoding Type
                dervived_file = skCase.addDerivedFile(file, local_file, os.path.getsize(dev_file), + \
                                0, 0, 0, 0, True, abstract_file_info, "", "thumb_viewer", "1.0.2.6", "", TskData.EncodingType.NONE)
                #self.log(Level.INFO, "Derived File ==> " + str(dervived_file))
            else:
                pass
```

```
            try:
              os.remove(lclDbPath)
            except:
              self.log(Level.INFO, "removal of thumbs.db file " + lclDbPath + " failed " )

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "Thumbs.db", " Thumbs.db Files Have Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK


    def check_dervived_existance(self, dataSource, file_name, parent_file_abstract):

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        dervived_file = fileManager.findFiles(dataSource, file_name, parent_file_abstract)
        numFiles = len(dervived_file)

        if numFiles == 0:
            return True
        else:
            return False
```

- ***Volatility***
  - ***Volatility***

```
  def process(self, dataSource, progressBar):

    self.log(Level.INFO, "Starting to process, Just before call to parse_safari_history")

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    # Get the temp directory and create the sub directory
    Temp_Dir = Case.getCurrentCase().getModulesOutputDirAbsPath()
    try:
       os.mkdir(Temp_Dir + "\Volatility")
    except:
       self.log(Level.INFO, "Plaso Import Directory already exists " + Temp_Dir)

    # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "%", "/")
    numFiles = len(files)
    self.log(Level.INFO, "Number of files to process ==> " + str(numFiles))
    #file_name = os.path.basename(self.path_to_storage_file)
    #self.log(Level.INFO, "File Name ==> " + file_name)
    #base_file_name = os.path.splitext(file_name)[0]
    #self.database_file = Temp_Dir + "\\volatility\\Plaso.db3"
    for file in files:
       self.log(Level.INFO, "File name to process is ==> " + str(file))
       self.log(Level.INFO, "File name to process is ==> " + str(file.getLocalAbsPath()))
       image_file = file.getLocalAbsPath()
       if image_file != None:
          self.log(Level.INFO, "File name to process is ==> " + str(file.getLocalAbsPath()))
          file_name = os.path.basename(file.getLocalAbsPath())
          self.log(Level.INFO, "File Name ==> " + file_name)
          base_file_name = os.path.splitext(file_name)[0]
          self.database_file = Temp_Dir + "\\volatility\\" + base_file_name + ".db3"
          self.log(Level.INFO, "File Name ==> " + self.database_file)
          if self.isAutodetect:
             self.find_profile(image_file)
          if self.Profile == None:
             continue
          for plugin_to_run in self.Plugins:
             if self.Python_Program:
                self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + file.getLocalAbsPath() + " " + \
                    "--profile=" + self.Profile + " --output=sqlite --output-file=" + self.database_file + " " + self.Additional_Parms
+ " " + plugin_to_run)
                pipe = Popen(["Python.exe", self.Volatility_Executable, "-f", file.getLocalAbsPath(), "--profile=" + self.Profile, "--
output=sqlite", \
                    "--output-file=" + self.database_file, self.Additional_Parms, plugin_to_run], stdout=PIPE, stderr=PIPE)
             else:
                self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + file.getLocalAbsPath() + " " + \
                    "--profile=" + self.Profile + " --output=sqlite --output-file=" + self.database_file + " " + self.Additional_Parms
+ " " + plugin_to_run)
                pipe = Popen([self.Volatility_Executable, "-f", file.getLocalAbsPath(), "--profile=" + self.Profile, "--output=sqlite", \
                    "--output-file=" + self.database_file, self.Additional_Parms, plugin_to_run], stdout=PIPE, stderr=PIPE)

             out_text = pipe.communicate()[0]
             self.log(Level.INFO, "Output from run is ==> " + out_text)

          # Open the DB using JDBC
          self.log(Level.INFO, "Path the volatility database file created ==> " + self.database_file)
          try:
             Class.forName("org.sqlite.JDBC").newInstance()
             dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % self.database_file)
          except SQLException as e:
             self.log(Level.INFO, "Could not open database file (not SQLite) " + self.database_file + " (" + e.getMessage() + ")")

          try:
             exestmt = dbConn.createStatement()
             resultx = exestmt.execute('create table plugins_loaded_to_Autopsy (table_name text);')
          except SQLException as e:
```

```
            self.log(Level.INFO, "Could not create table plugins_loaded_to_autopsy")

        # Query the database
        try:
            stmt = dbConn.createStatement()
            stmt2 = dbConn.createStatement()
            stmt3 = dbConn.createStatement()
            stmt4 = dbConn.createStatement()
            resultSet1 = stmt.executeQuery("Select upper(tbl_name) tbl_name from SQLITE_MASTER where upper(tbl_name) " \
                                           " not in (select table_name from plugins_loaded_to_Autopsy)" \
                                           " and upper(tbl_name) <> 'PLUGINS_LOADED_TO_AUTOPSY';")
            # Cycle through each row and create artifacts
            while resultSet1.next():
                try:
                    self.log(Level.INFO, "Begin Create New Artifacts ==> " + resultSet1.getString("tbl_name"))
                    artID_art = skCase.addArtifactType( "TSK_VOL_" + resultSet1.getString("tbl_name") + "_" + file_name,
"Volatility " + \
                                                        resultSet1.getString("tbl_name") + " " + file_name)
                except:
                    self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

                # Get the artifact and attributes
                artID_art = skCase.getArtifactTypeID("TSK_VOL_" + resultSet1.getString("tbl_name") + "_" + file_name)
                artID_art_evt = skCase.getArtifactType("TSK_VOL_" + resultSet1.getString("tbl_name") + "_" + file_name)
                try:
                    self.log(Level.INFO, "Result (" + resultSet1.getString("tbl_name") + ")")
                    table_name = resultSet1.getString("tbl_name")
                    resultSet4  = stmt4.executeQuery("Select count(*) 'NumRows' from " + resultSet1.getString("tbl_name") + " ")
                    row_count = resultSet4.getInt("NumRows")
                    self.log(Level.INFO, " Number of Rows is " + str(row_count) + " ")
                    if row_count >= 1:
                        SQL_String_1 = "Select * from " + table_name + ";"
                        SQL_String_2 = "PRAGMA table_info('" + table_name + "')"
                        self.log(Level.INFO, SQL_String_1)
                        self.log(Level.INFO, SQL_String_2)
                        artifact_name = "TSK_VOL_" + table_name.upper() + "_" + file_name

                        artID_sql = skCase.getArtifactTypeID(artifact_name)
                        artID_sql_evt = skCase.getArtifactType(artifact_name)

                        Column_Names = []
                        Column_Types = []
                        resultSet2  = stmt2.executeQuery(SQL_String_2)
                        while resultSet2.next():
                            Column_Names.append(resultSet2.getString("name").upper())
                            Column_Types.append(resultSet2.getString("type").upper())
                            attribute_name = "TSK_VOL_" + table_name + "_" + resultSet2.getString("name").upper()
                            if resultSet2.getString("type").upper() == "TEXT":
                                try:
                                    attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                                except:
                                    self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                            elif resultSet2.getString("type").upper() == "LONGVARCHAR":
                                try:
                                    attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                                except:
                                    self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                            elif resultSet2.getString("type").upper() == "":
                                try:
                                    attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                                except:
                                    self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                            elif resultSet2.getString("type").upper() == "BLOB":
                                try:
                                    attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                                except:
                                    self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                            elif resultSet2.getString("type").upper() == "REAL":
                                try:
                                    attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                                except:
```

```
                                self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")
                            else:
                                try:
                                    attID_ex1 = skCase.addArtifactAttributeType(attribute_name,
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                                except:
                                    self.log(Level.INFO, "Attributes Creation Error, " + attribute_name + " ==> ")


                            resultSet3 = stmt3.executeQuery(SQL_String_1)
                            while resultSet3.next():
                                art = file.newArtifact(artID_sql)
                                Column_Number = 1
                                for col_name in Column_Names:
                                    c_name = "TSK_VOL_" + table_name.upper() + "_" + Column_Names[Column_Number - 1]
                                    attID_ex1 = skCase.getAttributeType(c_name)
                                    if Column_Types[Column_Number - 1] == "TEXT":
                                        if resultSet3.getString(Column_Number) == None:
                                            art.addAttribute(BlackboardAttribute(attID_ex1, VolatilityIngestModuleFactory.moduleName, " "))
                                        else:
                                            art.addAttribute(BlackboardAttribute(attID_ex1, VolatilityIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                                    elif Column_Types[Column_Number - 1] == "":
                                        art.addAttribute(BlackboardAttribute(attID_ex1, VolatilityIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                                    elif Column_Types[Column_Number - 1] == "LONGVARCHAR":
                                        art.addAttribute(BlackboardAttribute(attID_ex1, VolatilityIngestModuleFactory.moduleName, "BLOBS
Not Supported - Look at actual file"))
                                    elif Column_Types[Column_Number - 1] == "BLOB":
                                        art.addAttribute(BlackboardAttribute(attID_ex1, VolatilityIngestModuleFactory.moduleName, "BLOBS
Not Supported - Look at actual file"))
                                    elif Column_Types[Column_Number - 1] == "REAL":
                                        art.addAttribute(BlackboardAttribute(attID_ex1, VolatilityIngestModuleFactory.moduleName,
long(resultSet3.getFloat(Column_Number))))
                                    else:
                                        art.addAttribute(BlackboardAttribute(attID_ex1, VolatilityIngestModuleFactory.moduleName,
long(resultSet3.getString(Column_Number))))
                                    Column_Number = Column_Number + 1


IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(VolatilityIngestModuleFactory.moduleName, \
                                artID_sql_evt, None))

                    except SQLException as e:
                        self.log(Level.INFO, "Error getting values from table " +  resultSet.getString("tbl_name") + " (" +
e.getMessage() + ")")
                    try:
#                       exestmt = createStatement()
                        resultx = exestmt.execute("insert into plugins_loaded_to_Autopsy values ('" + table_name + "');")
                    except SQLException as e:
                        self.log(Level.INFO, "Could not create table plugins_loaded_to_autopsy")

                except SQLException as e:
                    self.log(Level.INFO, "Error querying database " + file.getName() + " (" + e.getMessage() + ")")

        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "VolatilitySettings", " VolatilitySettings Has Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK

    def find_profile(self, image_file):

        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir)

        self.log(Level.INFO, "File name to process is ==> " + str(image_file))
        file_name = os.path.basename(image_file)
        self.log(Level.INFO, "File Name ==> " + file_name)
        base_file_name = os.path.splitext(file_name)[0]
        database_file = Temp_Dir + "\\" + base_file_name + ".db3"
        self.log(Level.INFO, "File Name ==> " + self.database_file)
        found_profile = False

        if os.path.isfile(self.database_file):
```

```
        self.log(Level.INFO, "Path the volatility database file created ==> " + self.database_file)
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % self.database_file)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the database
        try:
            stmt = dbConn.createStatement()
            resultSet1 = stmt.executeQuery('Select "Suggested Profile(s)" from imageinfo')
            self.log(Level.INFO, "query " + str(resultSet1))
            # Cycle through each row and create artifacts
            profile_names = None
            while resultSet1.next():
                try:
                    profile_names = resultSet1.getString("Suggested Profile(s)")
                    if profile_names == None:
                        self.Profile = None
                    elif ',' in profile_names:
                        profile_list = profile_names.split(",")
                        self.Profile = profile_list[0]
                    elif ' ' in profle_names:
                        profile_list = profile_names.split(" ")
                        self.Profile = profile_list[0]
                    else:
                        self.Profile = profile_names
                    found_profile = True
                except:
                    self.log(Level.INFO, "Error getting profile name, Profile name is ==> " + profile_names + " <==")
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")

        try:
            stmt.close()
            dbConn.close()
            #os.remove(database_name)
        except:
            self.log(Level.INFO, "removal of volatility imageinfo database failed " + Temp_Dir)

    if found_profile:
        pass
    else:
        if self.Python_Program:
            self.log(Level.INFO, "Running program ==> " + "Python " + self.Volatility_Executable + " -f " + image_file + " " + \
                    " --output=sqlite --output-file=" + self.database_file + " imageinfo")
            pipe = Popen(["Python.exe", self.Volatility_Executable, "-f", image_file, "--output=sqlite", \
                "--output-file=" + self.database_file, "imageinfo"], stdout=PIPE, stderr=PIPE)
        else:
            self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + image_file + " " + \
                    " --output=sqlite --output-file=" + self.database_file + " imageinfo")
            pipe = Popen([self.Volatility_Executable, "-f", image_file, "--output=sqlite", \
                "--output-file=" + self.database_file, "imageinfo"], stdout=PIPE, stderr=PIPE)

        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Open the DB using JDBC
        self.log(Level.INFO, "Path the volatility database file created ==> " + self.database_file)
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % self.database_file)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the database
        try:
            stmt = dbConn.createStatement()
            resultSet1 = stmt.executeQuery('Select "Suggested Profile(s)" from imageinfo')
            self.log(Level.INFO, "query SQLite Master table ==> " )
            self.log(Level.INFO, "query " + str(resultSet1))
            # Cycle through each row and create artifacts
            profile_names = None
            while resultSet1.next():
```

```
            try:
              profile_names = resultSet1.getString("Suggested Profile(s)")
              if profile_names == None:
                 self.Profile = None
              elif ',' in profile_names:
                 profile_list = profile_names.split(",")
                 self.Profile = profile_list[0]
              elif ' ' in profle_names:
                 profile_list = profile_names.split(" ")
                 self.Profile = profile_list[0]
              else:
                 self.Profile = profile_names

            except:
              self.log(Level.INFO, "Error getting profile name, Profile name is ==> " + profile_names + " <==")
         try:
            exestmt = dbConn.createStatement()
            resultx = exestmt.execute('create table plugins_loaded_to_Autopsy (table_name text);')
         except SQLException as e:
            self.log(Level.INFO, "Could not create table plugins_loaded_to_autopsy")
            return IngestModule.ProcessResult.OK

      except SQLException as e:
        self.log(Level.INFO, "Could not open database file (not SQLite) " + database_file + " (" + e.getMessage() + ")")

      try:
          stmt.close()
          exestmt.close()
          dbConn.close()
          #os.remove(database_name)
      except:
          self.log(Level.INFO, "removal of volatility imageinfo database failed " + Temp_Dir)
```

○ *Volatility_Convert*

```
  def process(self, dataSource, progressBar):

    self.log(Level.INFO, "Starting to process Hiberfil.sys and Crash Dumps")

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    # Get the temp directory and create the sub directory
    if self.hiber_flag:
      Mod_Dir = Case.getCurrentCase().getModulesOutputDirAbsPath()
      try:
        ModOut_Dir = os.path.join(Mod_Dir, "Volatility\\Memory-Image-hiberfil")
        self.log(Level.INFO, "Module Output Directory ===>  " + ModOut_Dir)
        #dir_util.mkpath(ModOut_Dir)
        os.mkdir(Mod_Dir + "\\Volatility")
        os.mkdir(ModOut_Dir)
      except:
        self.log(Level.INFO, "***** Error Module Output Directory already exists " + ModOut_Dir)

      # Set the database to be read to the once created by the prefetch parser program
      skCase = Case.getCurrentCase().getSleuthkitCase();
      fileManager = Case.getCurrentCase().getServices().getFileManager()
      files = fileManager.findFiles(dataSource, "hiberfil.sys", "/")
      numFiles = len(files)
      self.log(Level.INFO, "Number of files to process ==> " + str(numFiles))

      for file in files:
        self.log(Level.INFO, "File to process is ==> " + str(file))
        self.log(Level.INFO, "File name to process is ==> " + file.getName())
        tmp_Dir = Case.getCurrentCase().getTempDirectory()
        Hiber_File = os.path.join(tmp_Dir, file.getName())
        ContentUtils.writeToFile(file, File(Hiber_File))
        self.log(Level.INFO, "File name to process is ==> " + Hiber_File)
        # Create the directory to dump the hiberfil
        dump_file = os.path.join(ModOut_Dir, "Memory-Image-from-hiberfil.img")
        if self.Python_Program:
          self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " imagecopy -f " + Hiber_File + " " + \
              " -O " + dump_file)
          pipe = Popen(["Python.exe", self.Volatility_Executable, "imagecopy", "-f", Hiber_File, "-O" + dump_file],
stdout=PIPE, stderr=PIPE)
        else:
          self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " imagecopy -f " + Hiber_File + " " + \
              " -O " + dump_file)
          pipe = Popen([self.Volatility_Executable, "imagecopy", "-f", Hiber_File, "-O" + dump_file], stdout=PIPE, stderr=PIPE)
        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Add hiberfil memory image to a new local data source
        services = IngestServices.getInstance()

        progress_updater = ProgressUpdater()
        newDataSources = []

        dump_file = os.path.join(ModOut_Dir, "Memory-Image-from-hiberfil.img")
        dir_list = []
        dir_list.append(dump_file)

        # skCase = Case.getCurrentCase().getSleuthkitCase();
        fileManager_2 = Case.getCurrentCase().getServices().getFileManager()
        skcase_data = Case.getCurrentCase()

        # Get a Unique device id using uuid
        device_id = UUID.randomUUID()
        self.log(Level.INFO, "device id: ==> " + str(device_id))

        skcase_data.notifyAddingDataSource(device_id)

        # Add data source with files
        newDataSource = fileManager_2.addLocalFilesDataSource(str(device_id), "Hiberfile Memory Image", "", dir_list,
progress_updater)
```

```
            newDataSources.append(newDataSource.getRootDirectory())

            # Get the files that were added
            files_added = progress_updater.getFiles()
            #self.log(Level.INFO, "Fire Module1: ==> " + str(files_added))

            for file_added in files_added:
                skcase_data.notifyDataSourceAdded(file_added, device_id)
                self.log(Level.INFO, "Fire Module1: ==> " + str(file_added))


        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "HiberFil_Crash", " Hiberfil/Crash Dumps have been extracted fro Image. " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK
```

- ***Volatility_Dump***

```
  def process(self, dataSource, progressBar):

    self.log(Level.INFO, "Starting to process, Just before call to parse_safari_history")

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    # Get the temp directory and create the sub directory
    Temp_Dir = Case.getCurrentCase().getModulesOutputDirAbsPath()
    try:
      os.mkdir(Temp_Dir + "\Volatility\\Dump-Files")
    except:
      self.log(Level.INFO, "Volatility Directory already exists " + Temp_Dir)
    self.log(Level.INFO, "Volatility Directory already exists " + str(dataSource.getId()))
    # Set the database to be read to the once created by the prefetch parser program
    skCase = Case.getCurrentCase().getSleuthkitCase()
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "%", "/")
    numFiles = len(files)
    self.log(Level.INFO, "Number of files to process ==> " + str(numFiles))
    #file_name = os.path.basename(self.path_to_storage_file)
    #self.log(Level.INFO, "File Name ==> " + file_name)
    #base_file_name = os.path.splitext(file_name)[0]

    for file in files:
      if '/LogicalFileSet1/' == file.parentPath:
        self.log(Level.INFO, "File name to process is ==> " + str(file))
        self.log(Level.INFO, "File name to process is ==> " + str(file.getLocalAbsPath()))

        mem_abstract_file_info = skCase.getAbstractFileById(file.getId())

        image_file = file.getLocalAbsPath()
        if image_file != None:
          self.log(Level.INFO, "File name to process is ==> " + str(file.getLocalAbsPath()))
          file_name = os.path.basename(file.getLocalAbsPath())
          base_file_name = os.path.splitext(file_name)[0]
          self.database_file = Temp_Dir + "\\volatility\\" + base_file_name + ".db3"
          # self.log(Level.INFO, "File Name ==> " + self.database_file)
          derived_dir = "ModuleOutput\\volatility\\Dump-Files\\"
          dump_file = Temp_Dir + "\\volatility\\Dump-Files"
          if self.isAutodetect:
            self.find_profile(image_file)
          if self.Profile == None:
            continue
          for plugin_to_run in self.Plugins:
            plugin_dir = os.path.join(dump_file, plugin_to_run)
            try:
              os.mkdir(plugin_dir)
            except:
              pass
            if self.Python_Program:
              if self.Process_Ids_To_Dump == "":
                new_derived_dir = self.add_Volatility_Dump_dir(dataSource, mem_abstract_file_info, dump_file,
plugin_to_run, derived_dir)
                self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + file.getLocalAbsPath() + "
" + \
                  "--profile=" + self.Profile + " --dump-dir=" + plugin_dir + " " + plugin_to_run)
                pipe = Popen(["Python.exe", self.Volatility_Executable, "-f", file.getLocalAbsPath(), "--profile=" + self.Profile,
\
                  "--dump-dir=" + plugin_dir, plugin_to_run], stdout=PIPE, stderr=PIPE)
                out_text = pipe.communicate()[0]
                self.log(Level.INFO, "Output from run is ==> " + out_text)
                self.add_Volatility_Dump_file(dataSource, new_derived_dir, plugin_dir, derived_dir + "\\" + plugin_to_run)
              else:
                for pid_to_run in self.Process_Ids_To_Dump:
                  new_derived_dir = self.add_Volatility_Dump_dir(dataSource, mem_abstract_file_info, dump_file,
plugin_to_run, derived_dir)
                  pid_dir = os.path.join(plugin_dir, pid_to_run.lstrip())
                  try:
                    os.mkdir(pid_dir)
                  except:
```

```
                        pass
                    new_derived_dir_pid = self.add_Volatility_Dump_dir(dataSource, new_derived_dir, pid_dir,
pid_to_run.lstrip(), derived_dir + "\\" + plugin_to_run)
                    self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + file.getLocalAbsPath()
+ " " + \
                        "--profile=" + self.Profile + " --dump-dir=" + pid_dir + " --pid=" + pid_to_run.lstrip() + " " +
plugin_to_run)
                    pipe = Popen(["Python.exe", self.Volatility_Executable, "-f", file.getLocalAbsPath(), "--profile=" +
self.Profile, \
                      "--dump-dir=" + pid_dir, "--pid=" + pid_to_run.lstrip(), plugin_to_run], stdout=PIPE, stderr=PIPE)
                    out_text = pipe.communicate()[0]
                    self.log(Level.INFO, "Output from run is ==> " + out_text)
                    self.add_Volatility_Dump_file(dataSource, new_derived_dir_pid, pid_dir, derived_dir + "\\" + plugin_to_run
+ "\\" + pid_to_run.lstrip(), pid_to_run.lstrip())
                else:
                  if self.Process_Ids_To_Dump == "":
                    new_derived_dir = self.add_Volatility_Dump_dir(dataSource, mem_abstract_file_info, dump_file,
plugin_to_run, derived_dir)
                    self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + file.getLocalAbsPath() + "
" + \
                        "--profile=" + self.Profile + " --dump-dir=" + plugin_dir + " " + plugin_to_run)
                    pipe = Popen([self.Volatility_Executable, "-f", file.getLocalAbsPath(), "--profile=" + self.Profile, \
                        "--dump-dir=" + plugin_dir, plugin_to_run], stdout=PIPE, stderr=PIPE)
                    out_text = pipe.communicate()[0]
                    self.log(Level.INFO, "Output from run is ==> " + out_text)
                    self.add_Volatility_Dump_file(dataSource, new_derived_dir, plugin_dir, derived_dir + "\\" + plugin_to_run, "
")
                  else:
                    for pid_to_run in self.Process_Ids_To_Dump:
                      new_derived_dir = self.add_Volatility_Dump_dir(dataSource, mem_abstract_file_info, dump_file,
plugin_to_run, derived_dir)
                      pid_dir = os.path.join(plugin_dir, pid_to_run.lstrip())
                      try:
                        os.mkdir(pid_dir)
                      except:
                        pass
                      new_derived_dir_pid = self.add_Volatility_Dump_dir(dataSource, new_derived_dir, plugin_dir,
pid_to_run.lstrip(), derived_dir + "\\" + plugin_to_run)
                      self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + file.getLocalAbsPath()
+ " " + \
                          "--profile=" + self.Profile + " --dump-dir=" + pid_dir + " --pid=" + pid_to_run.lstrip() + " " +
plugin_to_run)
                      pipe = Popen([self.Volatility_Executable, "-f", file.getLocalAbsPath(), "--profile=" + self.Profile, \
                          "--dump-dir=" + pid_dir, "--pid=" + pid_to_run.lstrip(), plugin_to_run], stdout=PIPE, stderr=PIPE)
                      out_text = pipe.communicate()[0]
                      self.log(Level.INFO, "Output from run is ==> " + out_text)
                      self.add_Volatility_Dump_file(dataSource, new_derived_dir_pid, pid_dir, derived_dir + "\\" + plugin_to_run
+ "\\" + pid_to_run.lstrip(), pid_to_run.lstrip())


        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
            "VolatilitySettings", " VolatilitySettings Has Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK

    def find_profile(self, image_file):

        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir)

        self.log(Level.INFO, "File name to process is ==> " + str(image_file))
        file_name = os.path.basename(image_file)
        self.log(Level.INFO, "File Name ==> " + file_name)
        base_file_name = os.path.splitext(file_name)[0]
        #database_file = Temp_Dir + "\\" + base_file_name + ".db3"
        self.log(Level.INFO, "File Name ==> " + self.database_file)

        found_profile = False

        if os.path.isfile(self.database_file):
            self.log(Level.INFO, "Path the volatility database file created ==> " + self.database_file)
            try:
                Class.forName("org.sqlite.JDBC").newInstance()
                dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % self.database_file)
```

```
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + self.database_file + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the database
        try:
            stmt = dbConn.createStatement()
            resultSet1 = stmt.executeQuery('Select "Suggested Profile(s)" from imageinfo')
            self.log(Level.INFO, "query " + str(resultSet1))
            # Cycle through each row and create artifacts
            profile_names = None
            while resultSet1.next():
                try:
                    profile_names = resultSet1.getString("Suggested Profile(s)")
                    if profile_names == None:
                        self.Profile = None
                    elif ',' in profile_names:
                        profile_list = profile_names.split(",")
                        self.Profile = profile_list[0]
                    elif ' ' in profle_names:
                        profile_list = profile_names.split(" ")
                        self.Profile = profile_list[0]
                    else:
                        self.Profile = profile_names
                    found_profile = True
                except:
                    self.log(Level.INFO, "Error getting profile name, Profile name is ==> " + profile_names + " <==")
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + self.database_file + " (" + e.getMessage() + ")")

        try:
            stmt.close()
            dbConn.close()
            #os.remove(database_name)
        except:
            self.log(Level.INFO, "removal of volatility imageinfo database failed " + Temp_Dir)

    if found_profile:
        pass
    else:
        if self.Python_Program:
            self.log(Level.INFO, "Running program ==> " + "Python " + self.Volatility_Executable + " -f " + image_file + " " + \
                    " --output=sqlite --output-file=" + self.database_file + " imageinfo")
            pipe = Popen(["Python.exe", self.Volatility_Executable, "-f", image_file, "--output=sqlite", \
                    "--output-file=" + self.database_file, "imageinfo"], stdout=PIPE, stderr=PIPE)
        else:
            self.log(Level.INFO, "Running program ==> " + self.Volatility_Executable + " -f " + image_file + " " + \
                    " --output=sqlite --output-file=" + self.database_file + " imageinfo")
            pipe = Popen([self.Volatility_Executable, "-f", image_file, "--output=sqlite", \
                    "--output-file=" + self.database_file, "imageinfo"], stdout=PIPE, stderr=PIPE)

        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Open the DB using JDBC
        self.log(Level.INFO, "Path the volatility database file created ==> " + self.database_file)
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % self.database_file)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + self.database_file + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the database
        try:
            stmt = dbConn.createStatement()
            resultSet1 = stmt.executeQuery('Select "Suggested Profile(s)" from imageinfo')
            self.log(Level.INFO, "query SQLite Master table ==> " )
            self.log(Level.INFO, "query " + str(resultSet1))
            # Cycle through each row and create artifacts
            profile_names = None
            while resultSet1.next():
                try:
                    profile_names = resultSet1.getString("Suggested Profile(s)")
                    if profile_names == None:
                        self.Profile = None
```

```
                elif ',' in profile_names:
                    profile_list = profile_names.split(",")
                    self.Profile = profile_list[0]
                elif ' ' in profile_names:
                    profile_list = profile_names.split(" ")
                    self.Profile = profile_list[0]
                else:
                    self.Profile = profile_names

            except:
                self.log(Level.INFO, "Error getting profile name, Profile name is ==> " + profile_names + " <==")
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + self.database_file + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        try:
            stmt.close()
            dbConn.close()
            #os.remove(database_name)
        except:
            self.log(Level.INFO, "removal of volatility imageinfo database failed " + Temp_Dir)

    def add_Volatility_Dump_dir(self, dataSource, dir_abstract_file_info, dump_dir, dir_name, local_dir):

        skCase = Case.getCurrentCase().getSleuthkitCase()
        self.log(Level.INFO, " dir Name is ==> " + dir_name)
        self.log(Level.INFO, " abstract parentPath is ==> " + str(dir_abstract_file_info.parentPath))
        self.log(Level.INFO, "Dump Dir is ==> " + dump_dir)
        self.log(Level.INFO, "Local Directory is ==> " + local_dir)

        dev_file = os.path.join(dump_dir, dir_name)
        local_file = os.path.join(local_dir, dir_name)

        if not(self.check_derived_existance(dataSource, dir_name, dir_abstract_file_info.parentPath)):

            # Add derived file
            # Parameters Are:
            #    File Name, Local Path, size, ctime, crtime, atime, mtime, isFile, Parent File, rederive Details, Tool Name,
            #     Tool Version, Other Details, Encoding Type
            derived_file = skCase.addDerivedFile(dir_name, local_file, os.path.getsize(dev_file), + \
                                0, 0, 0, 0, True, dir_abstract_file_info, "", "Volatility", self.Volatility_Version, "",
TskData.EncodingType.NONE)
            IngestServices.getInstance().fireModuleContentEvent(ModuleContentEvent(derived_file))
#           self.context.addFilesToJob(df_list)
            #self.log(Level.INFO, "Derived File ==> " + str(derived_file))
        else:
            pass

        #self.log(Level.INFO, " derived File Is ==> " + str(derived_file))

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        new_derived_file = fileManager.findFiles(dataSource, dir_name, dir_abstract_file_info.parentPath)
        numFiles = len(new_derived_file)

        self.log(Level.INFO, " print number of files is " + str(numFiles))

        for file in new_derived_file:
            self.log(Level.INFO, "File Exists ==> " + str(file))
            self.log(Level.INFO, "Local Directory ==> " + str(file.localPath))
            self.log(Level.INFO, "Local Directory ==> " + local_file)
            if local_file == file.localPath:
                self.log(Level.INFO, "File Exists ==> " + str(file))
                return file

        self.log(Level.INFO, "File Exists2 ==> " + str(new_derived_file[0]))
        return new_derived_file[0]

    def add_Volatility_Dump_file(self, dataSource, dir_abstract_file_info, dump_dir, local_dir, pid_name):


        self.log(Level.INFO, "Adding Files from Dump Directory")
        self.log(Level.INFO, "Dump Dir is ==> " + dump_dir)
        self.log(Level.INFO, "Local Directory is ==> " + local_dir)
        self.log(Level.INFO, "Parent Path is ==> " + str(dir_abstract_file_info))

        #skCase = Case.getCurrentCase().getSleuthkitCase()
```

```
        skCase = Case.getCurrentCase().getServices().getFileManager()
        files = next(os.walk(dump_dir))[2]
        for file in files:
            self.log(Level.INFO, " File Name is ==> " + file)

            dev_file = os.path.join(dump_dir, file)
            local_file = os.path.join(local_dir, file)
            self.log(Level.INFO, " Dev File Name is ==> " + dev_file)
            self.log(Level.INFO, " Local File Name is ==> " + local_file)

            if not(self.check_derived_existance(dataSource, file, dir_abstract_file_info.parentPath)):

                # Add derived file
                # Parameters Are:
                #    File Name, Local Path, size, ctime, crtime, atime, mtime, isFile, Parent File, rederive Details, Tool Name,
                #     Tool Version, Other Details, Encoding Type
                derived_file = skCase.addDerivedFile(file, local_file, os.path.getsize(dev_file), + \
                                0, 0, 0, 0, True, dir_abstract_file_info, "", "Volatility", self.Volatility_Version, "",
TskData.EncodingType.NONE)
                IngestServices.getInstance().fireModuleContentEvent(ModuleContentEvent(derived_file))
                #self.log(Level.INFO, "Derived File ==> " + str(derived_file))
            else:
                pass

    def check_derived_existance(self, dataSource, file_name, parent_file_path):

        self.log(Level.INFO, "File Name is ==> " + str(file_name) + "  <==> Parent File Dir ==> " + str(parent_file_path))

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        derived_file = fileManager.findFiles(dataSource, file_name, parent_file_path)
        numFiles = len(derived_file)

        if numFiles == 0:
            self.log(Level.INFO, "File Does Not Exists ==> " + str(file_name))
            return False
        else:
            for file in derived_file:
                self.log(Level.INFO, "File Exists ==> " + str(file_name))
                if parent_file_path == file.parentPath:
                    self.log(Level.INFO, "File Exists ==> " + str(file_name))
                    return True
            self.log(Level.INFO, "File Does Not Exists ==> " + str(file_name))
            return False
```

- ***Windows_Internals***

```
import jarray
import inspect
import os
import shutil
from subprocess import Popen, PIPE

from javax.swing import JCheckBox
from java.awt import GridLayout
from java.awt import GridBagLayout
from java.awt import GridBagConstraints
from javax.swing import JPanel
from javax.swing import JFileChooser
from javax.swing import JScrollPane
from javax.swing.filechooser import FileNameExtensionFilter

from java.lang import Class
from java.lang import System
from java.sql  import DriverManager, SQLException
from java.util.logging import Level
from java.io import File
from org.sleuthkit.datamodel import SleuthkitCase
from org.sleuthkit.datamodel import AbstractFile
from org.sleuthkit.datamodel import ReadContentInputStream
from org.sleuthkit.datamodel import BlackboardArtifact
from org.sleuthkit.datamodel import BlackboardAttribute
from org.sleuthkit.autopsy.ingest import IngestModule
from org.sleuthkit.autopsy.ingest.IngestModule import IngestModuleException
from org.sleuthkit.autopsy.ingest import DataSourceIngestModule
from org.sleuthkit.autopsy.ingest import IngestModuleFactoryAdapter
from org.sleuthkit.autopsy.ingest import IngestModuleIngestJobSettings
from org.sleuthkit.autopsy.ingest import IngestModuleIngestJobSettingsPanel
from org.sleuthkit.autopsy.ingest import IngestMessage
from org.sleuthkit.autopsy.ingest import IngestServices
from org.sleuthkit.autopsy.ingest import ModuleDataEvent
from org.sleuthkit.autopsy.coreutils import Logger
from org.sleuthkit.autopsy.coreutils import PlatformUtil
from org.sleuthkit.autopsy.casemodule import Case
from org.sleuthkit.autopsy.casemodule.services import Services
from org.sleuthkit.autopsy.casemodule.services import FileManager
from org.sleuthkit.autopsy.datamodel import ContentUtils


# Factory that defines the name and details of the module and allows Autopsy
# to create instances of the modules that will do the analysis.
class Windows_InternalsIngestModuleFactory(IngestModuleFactoryAdapter):

    def __init__(self):
        self.settings = None

    moduleName = "Windows Internals"

    def getModuleDisplayName(self):
        return self.moduleName

    def getModuleDescription(self):
        return "Windows Internals"

    def getModuleVersionNumber(self):
        return "1.0"

    def getDefaultIngestJobSettings(self):
        return Windows_InternalsWithUISettings()

    def hasIngestJobSettingsPanel(self):
        return True

    # TODO: Update class names to ones that you create below
    def getIngestJobSettingsPanel(self, settings):
        if not isinstance(settings, Windows_InternalsWithUISettings):
            raise IllegalArgumentException("Expected settings argument to be instanceof SampleIngestModuleSettings")
        self.settings = settings
```

```
        return Windows_InternalsWithUISettingsPanel(self.settings)

    def isDataSourceIngestModuleFactory(self):
        return True

    def createDataSourceIngestModule(self, ingestOptions):
        return Windows_InternalsIngestModule(self.settings)

# Data Source-level ingest module.  One gets created per data source.
class Windows_InternalsIngestModule(DataSourceIngestModule):

    _logger = Logger.getLogger(Windows_InternalsIngestModuleFactory.moduleName)

    def log(self, level, msg):
        self._logger.logp(level, self.__class__.__name__, inspect.stack()[1][3], msg)

    def __init__(self, settings):
        self.context = None
        self.local_settings = settings
        self.List_Of_Windows_Internals = []
        self.List_Of_tables = []

    # Where any setup and configuration is done
    # 'context' is an instance of org.sleuthkit.autopsy.ingest.IngestJobContext.
    # See: http://sleuthkit.org/autopsy/docs/api-docs/3.1/classorg_1_1sleuthkit_1_1autopsy_1_1ingest_1_1_ingest_job_context.html
    def startUp(self, context):
        self.context = context

        if self.local_settings.getRecentlyused_Flag():
            self.log(Level.INFO, "Recently Used ==> " + str(self.local_settings.getRecentlyused_Flag()))
            self.path_to_Recentlyused_file = os.path.join(os.path.dirname(os.path.abspath(__file__)),
"show_ccm_recentlyusedapps.exe")
            if not os.path.exists(self.path_to_Recentlyused_file):
                raise IngestModuleException("Recentlyused Executable does not exist")

        if self.local_settings.getFilehistory_Flag():
            self.log(Level.INFO, "File Hsitory ==> " + str(self.local_settings.getFilehistory_Flag()))
            self.path_to_Filehistory_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "Export_FileHistory.exe")
            if not os.path.exists(self.path_to_Filehistory_file):
                raise IngestModuleException("Export_Filehistory Executable does not exist")

        if self.local_settings.getJumplist_Flag():
            self.log(Level.INFO, "Jumplist ==> " + str(self.local_settings.getJumplist_Flag()))
            self.path_to_Jumplist_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "export_jl_ad.exe")
            if not os.path.exists(self.path_to_Jumplist_file):
                raise IngestModuleException("Jumplist Executable does not exist")

        if self.local_settings.getPrefetch_Flag():
            self.log(Level.INFO, "Prefetch ==> " + str(self.local_settings.getPrefetch_Flag()))
            self.path_to_Prefetch_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "prefetch_parser_autopsy.exe")
            if not os.path.exists(self.path_to_Prefetch_file):
                raise IngestModuleException("Prefetch Executable does not exist")

        if self.local_settings.getSAM_Flag():
            self.log(Level.INFO, "SAM ==> " + str(self.local_settings.getSAM_Flag()))
            self.path_to_SAM_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "samparse.exe")
            if not os.path.exists(self.path_to_SAM_file):
                raise IngestModuleException("SAM Executable does not exist")

        if self.local_settings.getShellbags_Flag():
            self.log(Level.INFO, "shellbags ==> " + str(self.local_settings.getShellbags_Flag()))
            self.path_to_Shellbags_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "shellbags.exe")
            if not os.path.exists(self.path_to_Shellbags_file):
                raise IngestModuleException("Shellbags Executable does not exist")

        if self.local_settings.getShimcache_Flag():
            self.log(Level.INFO, "Shimcache ==> " + str(self.local_settings.getShimcache_Flag()))
            self.path_to_Shimcache_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "shimcache_parser.exe")
            if not os.path.exists(self.path_to_Shimcache_file):
                raise IngestModuleException("Shimcache Executable does not exist")

        if self.local_settings.getUsnj_Flag():
            self.log(Level.INFO, "USN ==> " + str(self.local_settings.getUsnj_Flag()))
            self.path_to_Usnj_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "parseusn.exe")
            if not os.path.exists(self.path_to_Usnj_file):
                raise IngestModuleException("Usnj Executable does not exist")
```

```
        if self.local_settings.getWebcache_Flag():
            self.log(Level.INFO, "Webcache ==> " + str(self.local_settings.getWebcache_Flag()))
            self.path_to_Webcache_file = os.path.join(os.path.dirname(os.path.abspath(__file__)), "Export_Webcache.exe")
            if not os.path.exists(self.path_to_Webcache_file):
                raise IngestModuleException("Webcache Executable does not exist")

        # Throw an IngestModule.IngestModuleException exception if there was a problem setting up
        # raise IngestModuleException(IngestModule(), "Oh No!")
        pass

    # Where the analysis is done.
    # The 'dataSource' object being passed in is of type org.sleuthkit.datamodel.Content.
    # See:x http://www.sleuthkit.org/sleuthkit/docs/jni-docs/interfaceorg_1_1sleuthkit_1_1datamodel_1_1_content.html
    # 'progressBar' is of type org.sleuthkit.autopsy.ingest.DataSourceIngestModuleProgress
    # See: http://sleuthkit.org/autopsy/docs/api-
docs/3.1/classorg_1_1sleuthkit_1_1autopsy_1_1ingest_1_1_data_source_ingest_module_progress.html
    def process(self, dataSource, progressBar):

        self.log(Level.INFO, "Starting to process, Just before call to parse_safari_history")

        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        if self.local_settings.getRecentlyused_Flag():
            progressBar.progress("Processing Recently Used Apps")
            self.process_Recentlyused(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Windows_Internals", " Recentlyused Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getFilehistory_Flag():
            progressBar.progress("Processing File History")
            self.process_Filehistory(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Windows_Internals", " File History Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getJumplist_Flag():
            progressBar.progress("Processing Jumplists")
            self.log(Level.INFO, "Starting to process Jumplist")
            self.process_Jumplist(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Windows_Internals", " Jumplist Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getPrefetch_Flag():
            progressBar.progress("Processing Prefetch")
            self.process_Prefetch(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Windows_Internals", " Prefetch Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getSAM_Flag():
            progressBar.progress("Processing SAM")
            self.process_SAM(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Windows_Internals", " SAM Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getShellbags_Flag():
            progressBar.progress("Processing Shellbags")
            self.process_Shellbags(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Windows_Internals", " Shellbags Have Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getShimcache_Flag():
            progressBar.progress("Processing Shimcache")
            self.process_Shimcache(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
                "Windows_Internals", " Shimcache Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getUsnj_Flag():
            progressBar.progress("Processing UsnJ")
```

```
            self.process_Usnj(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
              "Windows_Internals", " Usnj Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)

        if self.local_settings.getWebcache_Flag():
            progressBar.progress("Processing Webcache")
            self.process_Webcache(dataSource, progressBar)
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
              "Windows_Internals", " Webcache Has Been Analyzed " )
            IngestServices.getInstance().postMessage(message)


        # After all databases, post a message to the ingest messages in box.
        message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
          "Windows_Internals", " Windows_Internals Has Been Analyzed " )
        IngestServices.getInstance().postMessage(message)

        return IngestModule.ProcessResult.OK

    def process_Recentlyused(self, dataSource, progressBar):
        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Set the database to be read to the once created by the prefetch parser program
        skCase = Case.getCurrentCase().getSleuthkitCase();

        # Use blackboard class to index blackboard artifacts for keyword search
        blackboard = Case.getCurrentCase().getServices().getBlackboard()

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, "%", "/Windows/System32/wbem/Repository/")
        numFiles = len(files)
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        # Create Event Log directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir)
        try:
            os.mkdir(Temp_Dir + "\Recently_Used")
        except:
            self.log(Level.INFO, "Recently Used Directory already exists " + Temp_Dir)

        # Write out each Event Log file to the temp directory
        for file in files:

            # Check if the user pressed cancel while we were busy
            if self.context.isJobCancelled():
                return IngestModule.ProcessResult.OK

            #self.log(Level.INFO, "Processing file: " + file.getName())
            fileCount += 1

            if (file.getName() == '.' or file.getName() == '..'):
                self.log(Level.INFO, "Parent or Root Directory File not writing")
            else:
                # Save the DB locally in the temp folder. use file id as name to reduce collisions
                lclDbPath = os.path.join(Temp_Dir + "\Recently_Used", file.getName())
                ContentUtils.writeToFile(file, File(lclDbPath))

        self.log(Level.INFO, "Running prog ==> " + self.path_to_Recentlyused_file + " win7 " + Temp_Dir + "\Recently_Used " + " "
+ \
                                Temp_Dir + "\Recently_Used\\recentlyUsedApps.db3")
        pipe = Popen([self.path_to_Recentlyused_file, "win7", Temp_Dir + "\Recently_Used", Temp_Dir +
"\Recently_Used\\recentlyUsedApps.db3"], stdout=PIPE, stderr=PIPE)

        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory() + "\Recently_Used", "recentlyUsedApps.db3")
        if ("Exiting" in out_text):
            message = IngestMessage.createMessage(IngestMessage.MessageType.DATA,
              "CCM Recently Used Apps", " Error in CCM Recently Used Apps module " )
            IngestServices.getInstance().postMessage(message)
        else:
```

```
        # Add custom Artifact to blackboard
        try:
          self.log(Level.INFO, "Begin Create New Artifacts ==> TSK_CCM_RECENTLY_USED_APPS")
          artID_art = skCase.addArtifactType("TSK_CCM_RECENTLY_USED_APPS", "WMI Recently Used Apps")
        except:
          self.log(Level.INFO, "Artifacts Creation Error, artifact TSK_CCM_RECENTLY_USED_APPS exists. ==> ")

        # Add Custom attributes to blackboard
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_EXPLORER_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Explorer File Name")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Explorer File Name ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_FILE_SIZE",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Size")
        except:
          self.log(Level.INFO, "Attributes Creation Error, File Size ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_LAST_USED_TIME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "Last Used Time")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Last Used Time ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_TIME_ZONE_OFFSET",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Time Zone Offset")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Time Zone Offset ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_LAUNCH_COUNT",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Launch Count")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Launch Count ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_ORIG_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Original File Name")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Original File Name ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_FILE_DESC",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Description")
        except:
          self.log(Level.INFO, "Attributes Creation Error, File Description ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_PROD_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Product Name")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Product Name ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_PROD_VERSION",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Product Version")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Product Version ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_FILE_VERSION",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Version")
        except:
          self.log(Level.INFO, "Attributes Creation Error, File Version ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_ADDITIONAL_PROD_CODES",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Additional Product Codes")
        except:
          self.log(Level.INFO, "Attributes Creation Error, Additional Product Codes ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_MSI_VERSION",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "MSI Version")
        except:
          self.log(Level.INFO, "Attributes Creation Error, MSI Version ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_MSI_DISPLAY_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "MSI Display Name")
        except:
          self.log(Level.INFO, "Attributes Creation Error, MSI Display Name ==> ")
        try:
          attID_efn = skCase.addArtifactAttributeType("TSK_PRODUCT_CODE",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Product Code")
```

```
        except:
            self.log(Level.INFO, "Attributes Creation Error, Product Code ==> ")
        try:
            attID_efn = skCase.addArtifactAttributeType("TSK_SOFTWARE_PROP_HASH",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Software Property Hash")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Software Property Hash ==> ")
        try:
            attID_efn = skCase.addArtifactAttributeType("TSK_PROD_LANG",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Product Language")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Product Language ==> ")
        try:
            attID_efn = skCase.addArtifactAttributeType("TSK_FILE_PROP_HASH",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Property Hash")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Property Hash ==> ")
        try:
            attID_efn = skCase.addArtifactAttributeType("TSK_MSI_PUBLISHER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "MSI Publisher")
        except:
            self.log(Level.INFO, "Attributes Creation Error, MSI Publisher ==> ")

        for file in files:
            if (file.getName() == "OBJECTS.DATA"):

                # Open the DB using JDBC
                lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory() + "\\Recently_Used", "recentlyUsedApps.db3")
                self.log(Level.INFO, "Path the recentlyUsedApps.db3 database file created ==> " + lclDbPath)
                try:
                    Class.forName("org.sqlite.JDBC").newInstance()
                    dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
                except SQLException as e:
                    self.log(Level.INFO, "Could not open database file (not SQLite) recentlyUsedApps.db3 (" + e.getMessage() + ")")
                    return IngestModule.ProcessResult.OK

                # Query the history_visits table in the database and get all columns.
                try:
                    stmt = dbConn.createStatement()
                    recently_used_sql = "select FolderPath 'TSK_PATH', ExplorerFileName 'TSK_EXPLORER_FILE_NAME', " + \
                                        "FileSize 'TSK_FILE_SIZE', LastUserName 'TSK_USER_ID', strftime('%s',LastUsedTime) " + \
                                        "'TSK_LAST_USED_TIME', TimeZoneOffset 'TSK_TIME_ZONE_OFFSET', LaunchCount " + \
                                        "'TSK_LAUNCH_COUNT', OriginalFileName 'TSK_ORIG_FILE_NAME', FileDescription " + \
                                        "'TSK_FILE_DESC', CompanyName 'TSK_ORGANIZATION', ProductName 'TSK_PROD_NAME', "
+ \
                                        "ProductVersion 'TSK_PROD_VERSION', FileVersion 'TSK_FILE_VERSION', " + \
                                        "AdditionalProductCodes 'TSK_ADDITIONAL_PROD_CODES', msiVersion " + \
                                        "'TSK_MSI_VERSION', msiDisplayName 'TSK_MSI_DISPLAY_NAME', " + \
                                        "ProductCode 'TSK_PRODUCT_CODE', SoftwarePropertiesHash " + \
                                        "'TSK_SOFTWARE_PROP_HASH', ProductLanguage 'TSK_PROD_LANG', " + \
                                        "FilePropertiesHash 'TSK_FILE_PROP_HASH', msiPublisher 'TSK_MSI_PUBLISHER' " + \
                                        "from recently_used;"
                    self.log(Level.INFO, recently_used_sql)
                    resultSet = stmt.executeQuery(recently_used_sql)
                    self.log(Level.INFO, "query recently_used table")
                except SQLException as e:
                    self.log(Level.INFO, "Error querying database for recently_used table (" + e.getMessage() + ")")
                    return IngestModule.ProcessResult.OK

                artID_hst = skCase.getArtifactTypeID("TSK_CCM_RECENTLY_USED_APPS")
                artID_hst_evt = skCase.getArtifactType("TSK_CCM_RECENTLY_USED_APPS")

                meta = resultSet.getMetaData()
                columncount = meta.getColumnCount()
                column_names = []
                self.log(Level.INFO, "Number of Columns in the table ==> " + str(columncount))
                for x in range (1, columncount + 1):
                    self.log(Level.INFO, "Column Name ==> " + meta.getColumnLabel(x))
                    column_names.append(meta.getColumnLabel(x))

                self.log(Level.INFO, "All Columns ==> " + str(column_names))
                # Cycle through each row and create artifacts
                while resultSet.next():
                    try:
                        #self.log(Level.INFO, SQL_String_1)
                        self.log(Level.INFO, "Artifact Is ==> " + str(artID_hst))
```

```
                art = file.newArtifact(artID_hst)
                self.log(Level.INFO, "Inserting attribute URL")
                for col_name in column_names:
                  attID_ex1 = skCase.getAttributeType(col_name)
                  self.log(Level.INFO, "Inserting attribute ==> " + str(attID_ex1))
                  self.log(Level.INFO, "Attribute Type ==> " + str(attID_ex1.getValueType()))
                  if attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet.getString(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes String Creation Error, " + col_name + " ==> ")
                  elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.INTEGER:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet.getInt(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes Integer Creation Error, " + col_name + " ==> ")
                  elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet.getInt(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes Long Creation Error, " + col_name + " ==> ")
                  elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DOUBLE:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet.getInt(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes Double Creation Error, " + col_name + " ==> ")
                  elif attID_ex1.getValueType() ==
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.BYTE:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet.getString(col_name)))
                    except:
                      self.log(Level.INFO, "Attributes Byte Creation Error, " + col_name + " ==> ")
                  else:
                    try:
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
int(resultSet.getString(col_name))))
                    except:
                      self.log(Level.INFO, "Attributes Datatime Creation Error, " + col_name + " ==> ")

                # index the artifact for keyword search
                try:
                  blackboard.indexArtifact(art)
                except:
                  self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())

              except SQLException as e:
                self.log(Level.INFO, "Error getting values from web_history table (" + e.getMessage() + ")")

            IngestServices.getInstance().fireModuleDataEvent(
                ModuleDataEvent(Windows_InternalsIngestModuleFactory.moduleName, artID_hst_evt, None))

            stmt.close()
            dbConn.close()

        # Clean up
        try:
          os.remove(lclDbPath)
        except:
          self.log(Level.INFO, "removal of Recently Used database failed ")
        #Clean up EventLog directory and files
        for file in files:
          try:
            os.remove(Temp_Dir + "\\Recently_Used" + "\\" + file.getName())
          except:
            self.log(Level.INFO, "removal of Recently Used files failed " + Temp_Dir + "\\" + file.getName())
        try:
```

```
            shutil.rmtree(Temp_Dir + "\\Recently_Used")
        except:
            self.log(Level.INFO, "removal of recently used directory failed " + Temp_Dir)

    def process_Filehistory(self, dataSource, progressBar):


        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Check to see if the artifacts exist and if not then create it, also check to see if the attributes
        # exist and if not then create them
        skCase = Case.getCurrentCase().getSleuthkitCase();

        # This will work in 4.0.1 and beyond
        # Use blackboard class to index blackboard artifacts for keyword search
        blackboard = Case.getCurrentCase().getServices().getBlackboard()

        try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_cat1 = skCase.addArtifactType( "TSK_FH_CATALOG_1", "File History Catalog 1")
        except:
            self.log(Level.INFO, "Artifacts Creation Error, Catalog 1. ==> ")
            artID_cat1 = skCase.getArtifactTypeID("TSK_FH_CATALOG_1")
        try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_cat2 = skCase.addArtifactType( "TSK_FH_CATALOG_2", "File History Catalog 2")
        except:
            self.log(Level.INFO, "Artifacts Creation Error, Catalog 2. ==> ")
            artID_cat2 = skCase.getArtifactTypeID("TSK_FH_CATALOG_2")

        # Create the attribute type, if it exists then catch the error
        try:
            attID_fh_pn = skCase.addArtifactAttributeType('TSK_FH_PATH',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Parent Path")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Prefetch Parent Path. ==> ")

        try:
            attID_fh_fn = skCase.addArtifactAttributeType('TSK_FH_FILE_NAME',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Name")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Name. ==> ")

        try:
            attID_fh_fs = skCase.addArtifactAttributeType('TSK_FH_FILE_SIZE',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Size")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Size. ==> ")

        try:
            attID_fh_usn = skCase.addArtifactAttributeType('TSK_FH_USN_JOURNAL_ENTRY',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "USN Journal Entry")
        except:
            self.log(Level.INFO, "Attributes Creation Error, USN Journal Entry. ==> ")

        try:
            attID_fh_fc = skCase.addArtifactAttributeType('TSK_FH_FILE_CREATED',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "File Created")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Created. ==> ")

        try:
            attID_fh_fm = skCase.addArtifactAttributeType('TSK_FH_FILE_MODIFIED',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "File Modified")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 3. ==> ")

        try:
            attID_fh_bq = skCase.addArtifactAttributeType('TSK_FH_BACKUP_QUEUED',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "Backup Queued")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Backup Queued ==> ")

        try:
```

```
        attID_fh_bc = skCase.addArtifactAttributeType('TSK_FH_BACKUP_CREATED',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "Backup Created")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Backup Created ==> ")

    try:
        attID_fh_bcp = skCase.addArtifactAttributeType('TSK_FH_BACKUP_CAPTURED',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "Backup Captured")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Backup Captured. ==> ")

    try:
        attID_fh_bu = skCase.addArtifactAttributeType('TSK_FH_BACKUP_UPDATED',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "Backup Updated")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Backup Updated. ==> ")

    try:
        attID_fh_bv = skCase.addArtifactAttributeType('TSK_FH_BACKUP_VISIBLE',
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "Backup Visible")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Backup Visible ==> ")

    self.log(Level.INFO, "Get Artifacts after they were created.")
    # Get the new artifacts and attributes that were just created
    attID_fh_pn = skCase.getAttributeType("TSK_FH_PATH")
    attID_fh_fn = skCase.getAttributeType("TSK_FH_FILE_NAME")
    attID_fh_fs = skCase.getAttributeType("TSK_FH_FILE_SIZE")
    attID_fh_usn = skCase.getAttributeType("TSK_FH_USN_JOURNAL_ENTRY")
    attID_fh_fc = skCase.getAttributeType("TSK_FH_FILE_CREATED")
    attID_fh_fm = skCase.getAttributeType("TSK_FH_FILE_MODIFIED")
    attID_fh_bq = skCase.getAttributeType("TSK_FH_BACKUP_QUEUED")
    attID_fh_bc = skCase.getAttributeType("TSK_FH_BACKUP_CREATED")
    attID_fh_bcp = skCase.getAttributeType("TSK_FH_BACKUP_CAPTURED")
    attID_fh_bu = skCase.getAttributeType("TSK_FH_BACKUP_UPDATED")
    attID_fh_bv = skCase.getAttributeType("TSK_FH_BACKUP_VISIBLE")

    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    # Find the file history files from the users folders
    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "%edb", "%/Windows/FileHistory/%")

    numFiles = len(files)
    self.log(Level.INFO, "found " + str(numFiles) + " files")
    progressBar.switchToDeterminate(numFiles)
    fileCount = 0;

    # Create file history directory in temp directory, if it exists then continue on processing
    Temp_Dir = Case.getCurrentCase().getTempDirectory() + "\File_History"
    self.log(Level.INFO, "create Directory " + Temp_Dir)
    try:
        os.mkdir(Temp_Dir)
    except:
        self.log(Level.INFO, "File_History Directory already exists " + Temp_Dir)

    # Write out each catalog esedb database to the temp directory
    for file in files:

        # Check if the user pressed cancel while we were busy
        if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK
        fileCount += 1

        # Save the DB locally in the temp folder. use file id as name to reduce collisions
        lclDbPath = os.path.join(Temp_Dir, file.getName() + "_" + str(file.getId()))
        db_name = os.path.splitext(file.getName())[0]
        lclSQLPath = os.path.join(Temp_Dir, db_name + "_" + str(file.getId()) + ".db3")
        ContentUtils.writeToFile(file, File(lclDbPath))

        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program on data source parm 1 ==> " + self.path_to_Filehistory_file + " " + lclDbPath + " "
+ lclSQLPath)
        pipe = Popen([self.path_to_Filehistory_file, lclDbPath, lclSQLPath], stdout=PIPE, stderr=PIPE)
```

```
        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        if db_name == "Catalog1":
            artID_fh = skCase.getArtifactTypeID("TSK_FH_CATALOG_1")
            artID_fh_evt = skCase.getArtifactType("TSK_FH_CATALOG_1")
        else:
            artID_fh = skCase.getArtifactTypeID("TSK_FH_CATALOG_2")
            artID_fh_evt = skCase.getArtifactType("TSK_FH_CATALOG_2")

        userpath = file.getParentPath()
        username = userpath.split('/')
        self.log(Level.INFO, "Getting Username " + username[2]   )

        # Open the DB using JDBC
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclSQLPath)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + lclSQLPath + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the contacts table in the database and get all columns.
        try:
            stmt = dbConn.createStatement()
            SQL_Statement = "Select ParentName 'TSK_FH_PATH', Childname 'TSK_FH_FILE_NAME', " + \
                        "Filesize 'TSK_FH_FILE_SIZE', " + \
                        "usn 'TSK_FH_USN_JOURNAL_ENTRY', " + \
                        "FileCreated 'TSK_FH_FILE_CREATED', filemodified 'TSK_FH_FILE_MODIFIED', " + \
                        "tqueued 'TSK_FH_BACKUP_QUEUED', tcreated 'TSK_FH_BACKUP_CREATED', " + \
                        "tcaptured 'TSK_FH_BACKUP_CAPTURED', tupdated 'TSK_FH_BACKUP_UPDATED', " + \
                        "tvisible 'TSK_FH_BACKUP_VISIBLE' from file_history"
            self.log(Level.INFO, "SQL Statement --> " + SQL_Statement)
            resultSet = stmt.executeQuery(SQL_Statement)
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for File_History table (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Cycle through each row and create artifacts
        while resultSet.next():
            try:
                FH_Path  = resultSet.getString("TSK_FH_PATH")
                FH_File_Name = resultSet.getString("TSK_FH_FILE_NAME")
                FH_Filesize = resultSet.getString("TSK_FH_FILE_SIZE")
                FH_Usn = resultSet.getString("TSK_FH_USN_JOURNAL_ENTRY")
                FH_FC = resultSet.getInt("TSK_FH_FILE_CREATED")
                FH_FM = resultSet.getInt("TSK_FH_FILE_MODIFIED")
                FH_BQ = resultSet.getInt("TSK_FH_BACKUP_QUEUED")
                FH_BC = resultSet.getInt("TSK_FH_BACKUP_CREATED")
                FH_BCP = resultSet.getInt("TSK_FH_BACKUP_CAPTURED")
                FH_BU = resultSet.getInt("TSK_FH_BACKUP_UPDATED")
                FH_BV = resultSet.getInt("TSK_FH_BACKUP_VISIBLE")
            except SQLException as e:
                self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

            # Make artifact for TSK_PREFETCH,  this can happen when custom attributes are fully supported
            art = file.newArtifact(artID_fh)


            # Add the attributes to the artifact.
            art.addAttributes(((BlackboardAttribute(attID_fh_pn, Windows_InternalsIngestModuleFactory.moduleName, FH_Path)),
\
                        (BlackboardAttribute(attID_fh_fn, Windows_InternalsIngestModuleFactory.moduleName, FH_File_Name)), \
                        (BlackboardAttribute(attID_fh_fs, Windows_InternalsIngestModuleFactory.moduleName, FH_Filesize)), \
                        (BlackboardAttribute(attID_fh_usn, Windows_InternalsIngestModuleFactory.moduleName, FH_Usn)), \
                        (BlackboardAttribute(attID_fh_fc, Windows_InternalsIngestModuleFactory.moduleName, FH_FC)), \
                        (BlackboardAttribute(attID_fh_fm, Windows_InternalsIngestModuleFactory.moduleName, FH_FM)), \
                        (BlackboardAttribute(attID_fh_bq, Windows_InternalsIngestModuleFactory.moduleName, FH_BQ)), \
                        (BlackboardAttribute(attID_fh_bc, Windows_InternalsIngestModuleFactory.moduleName, FH_BC)), \
                        (BlackboardAttribute(attID_fh_bcp, Windows_InternalsIngestModuleFactory.moduleName, FH_BCP)), \
                        (BlackboardAttribute(attID_fh_bu, Windows_InternalsIngestModuleFactory.moduleName, FH_BU)), \
                        (BlackboardAttribute(attID_fh_bv, Windows_InternalsIngestModuleFactory.moduleName, FH_BV)), \
                        (BlackboardAttribute(BlackboardAttribute.ATTRIBUTE_TYPE.TSK_USER_NAME.getTypeID(), \
                         Windows_InternalsIngestModuleFactory.moduleName, username[2])))

            try:
```

```
                #index the artifact for keyword search
                blackboard.indexArtifact(art)
            except Blackboard.BlackboardException as e:
                self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())


IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(Windows_InternalsIngestModuleFactory.moduleName,
artID_fh_evt, None))

        # Clean up
        stmt.close()
        dbConn.close()
        #os.remove(lclDbPath)

    #Clean up prefetch directory and files
    try:
        shutil.rmtree(Temp_Dir)
    except:
        self.log(Level.INFO, "removal of directory tree failed " + Temp_Dir)

  def process_Jumplist(self, dataSource, progressBar):

    self.path_to_app_id_db = os.path.join(os.path.dirname(os.path.abspath(__file__)), "Jump_List_App_Ids.db3")

    #skCase Check to see if the artifacts exist and if not then create it, also check to see if the attributes
    # exist and if not then create them
    skCase = Case.getCurrentCase().getSleuthkitCase();

    # Use blackboard class to index blackboard artifacts for keyword search
    blackboard = Case.getCurrentCase().getServices().getBlackboard()

    try:
        self.log(Level.INFO, "Begin Create New Artifacts")
        artID_jl_ad = skCase.addArtifactType( "TSK_JL_AD", "Jump List Auto Dest")
    except:
        self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
        artID_jl_ad = skCase.getArtifactTypeID("TSK_JL_AD")

    try:
        attID_jl_fn = skCase.addArtifactAttributeType("TSK_JLAD_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "JumpList File Name")
    except:
        self.log(Level.INFO, "Attributes Creation Error, JL AD File Name. ==> ")
    try:
        attID_jl_fg = skCase.addArtifactAttributeType("TSK_JLAD_FILE_DESCRIPTION",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Description")
    except:
        self.log(Level.INFO, "Attributes Creation Error, File Description. ==> ")
    try:
        attID_jl_in = skCase.addArtifactAttributeType("TSK_JLAD_ITEM_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Item Name")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Item Name. ==> ")
    try:
        attID_jl_cl = skCase.addArtifactAttributeType("TSK_JLAD_COMMAND_LINE_ARGS",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Command Line Args")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Command Line Arguments. ==> ")
    try:
        attID_jl_dt = skCase.addArtifactAttributeType("TSK_JLAD_Drive Type",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "Drive Type")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Drive Type. ==> ")
    try:
        attID_jl_dsn = skCase.addArtifactAttributeType("TSK_JLAD_DRIVE_SERIAL_NUMBER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "Drive Serial Number")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Drive Serial Number. ==> ")
    try:
        attID_jl_des = skCase.addArtifactAttributeType("TSK_JLAD_DESCRIPTION",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Description")
    except:
        self.log(Level.INFO, "Attributes Creation Error, Description. ==> ")
    try:
        attID_jl_evl = skCase.addArtifactAttributeType("TSK_JLAD_ENVIRONMENT_VARIABLES_LOCATION",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Env Var Location")
```

```
        except:
            self.log(Level.INFO, "Attributes Creation Error, Env Var Location. ==> ")
        try:
            attID_jl_fat = skCase.addArtifactAttributeType("TSK_JLAD_FILE_ACCESS_TIME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Access Time")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Access Time. ==> ")
        try:
            attID_jl_faf = skCase.addArtifactAttributeType("TSK_JLAD_FILE_ATTRIBUTE_FLAGS",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "File Attribute Flags")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Attribute Flags. ==> ")
        try:
            attID_jl_fct = skCase.addArtifactAttributeType("TSK_JLAD_FILE_CREATION_TIME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Creation Time")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Creation Time. ==> ")
        try:
            attID_jl_fmt = skCase.addArtifactAttributeType("TSK_JLAD_FILE_MODIFICATION_TIME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "File Modification Time")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Modification Time. ==> ")
        try:
            attID_jl_fs = skCase.addArtifactAttributeType("TSK_JLAD_FILE_SIZE",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, "File Size")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Size. ==> ")
        try:
            attID_jl_ic = skCase.addArtifactAttributeType("TSK_JLAD_ICON_LOCATION",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Icon Location")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Icon Location. ==> ")
        try:
            attID_jl_ltid = skCase.addArtifactAttributeType("TSK_JLAD_LINK_TARGET_IDENTIFIER_DATA",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Link Target Identifier Data")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Link Target Identifier Data. ==> ")
        try:
            attID_jl_lp = skCase.addArtifactAttributeType("TSK_JLAD_LOCAL_PATH",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Local Path")
        except:
            self.log(Level.INFO, "Attributes Creation Error, File Modification Time. ==> ")
        try:
            attID_jl_mi = skCase.addArtifactAttributeType("TSK_JLAD_FILE_MACHINE_IDENTIFIER",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Machine Identifier")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Machine Identifier. ==> ")
        try:
            attID_jl_np = skCase.addArtifactAttributeType("TSK_JLAD_NETWORK_PATH",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Network Path")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Network Path. ==> ")
        try:
            attID_jl_rp = skCase.addArtifactAttributeType("TSK_JLAD_RELATIVE_PATH",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Relative Path")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Relative Path. ==> ")
        try:
            attID_jl_vl = skCase.addArtifactAttributeType("TSK_JLAD_VOLUME_LABEL",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Volume Label")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Volume Label. ==> ")
        try:
            attID_jl_wc = skCase.addArtifactAttributeType("TSK_JLAD_WORKING_DIRECTORY",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Working Directory")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Working Directory. ==> ")

        # Get the new artifacts and attributes that were just created
        artID_jl_ad = skCase.getArtifactTypeID("TSK_JL_AD")
        artID_jl_ad_evt = skCase.getArtifactType("TSK_JL_AD")
        attID_jl_fn = skCase.getAttributeType("TSK_JLAD_FILE_NAME")
        attID_jl_fg = skCase.getAttributeType("TSK_JLAD_FILE_DESCRIPTION")
        attID_jl_in = skCase.getAttributeType("TSK_JLAD_ITEM_NAME")
        attID_jl_cl = skCase.getAttributeType("TSK_JLAD_COMMAND_LINE_ARGS")
        attID_jl_dt = skCase.getAttributeType("TSK_JLAD_Drive Type")
```

```
        attID_jl_dsn = skCase.getAttributeType("TSK_JLAD_DRIVE_SERIAL_NUMBER")
        attID_jl_des = skCase.getAttributeType("TSK_JLAD_DESCRIPTION")
        attID_jl_evl = skCase.getAttributeType("TSK_JLAD_ENVIRONMENT_VARIABLES_LOCATION")
        attID_jl_fat = skCase.getAttributeType("TSK_JLAD_FILE_ACCESS_TIME")
        attID_jl_faf = skCase.getAttributeType("TSK_JLAD_FILE_ATTRIBUTE_FLAGS")
        attID_jl_fct = skCase.getAttributeType("TSK_JLAD_FILE_CREATION_TIME")
        attID_jl_fmt = skCase.getAttributeType("TSK_JLAD_FILE_MODIFICATION_TIME")
        attID_jl_fs = skCase.getAttributeType("TSK_JLAD_FILE_SIZE")
        attID_jl_ic = skCase.getAttributeType("TSK_JLAD_ICON_LOCATION")
        attID_jl_ltid = skCase.getAttributeType("TSK_JLAD_LINK_TARGET_IDENTIFIER_DATA")
        attID_jl_lp = skCase.getAttributeType("TSK_JLAD_LOCAL_PATH")
        attID_jl_mi = skCase.getAttributeType("TSK_JLAD_FILE_MACHINE_IDENTIFIER")
        attID_jl_np = skCase.getAttributeType("TSK_JLAD_NETWORK_PATH")
        attID_jl_rp = skCase.getAttributeType("TSK_JLAD_RELATIVE_PATH")
        attID_jl_vl = skCase.getAttributeType("TSK_JLAD_VOLUME_LABEL")
        attID_jl_wd = skCase.getAttributeType("TSK_JLAD_WORKING_DIRECTORY")

        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Find the Windows Event Log Files
        files = []

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, "%.automaticDestinations-ms")

        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        # Create Event Log directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir + "\JL_AD")
        try:
            os.mkdir(Temp_Dir + "\JL_AD")
        except:
            self.log(Level.INFO, "JL_AD Directory already exists " + Temp_Dir)

        # Write out each Event Log file to the temp directory
        for file in files:

            # Check if the user pressed cancel while we were busy
            if self.context.isJobCancelled():
                return IngestModule.ProcessResult.OK

            #self.log(Level.INFO, "Processing file: " + file.getName())
            fileCount += 1

            # Save the DB locally in the temp folder. use file id as name to reduce collisions
            lclDbPath = os.path.join(Temp_Dir + "\JL_AD", file.getName())
            ContentUtils.writeToFile(file, File(lclDbPath))


        # Example has only a Windows EXE, so bail if we aren't on Windows
        if not PlatformUtil.isWindowsOS():
            self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
            return IngestModule.ProcessResult.OK

        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "\JL_AD" + "  Parm 2 ==> " + Temp_Dir
+ "\JL_AD.db3")
        pipe = Popen([self.path_to_Jumplist_file, Temp_Dir + "\JL_AD", Temp_Dir + "\JL_AD.db3", self.path_to_app_id_db],
stdout=PIPE, stderr=PIPE)

        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Set the database to be read to the one created by the Event_EVTX program
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "JL_AD.db3")
        self.log(Level.INFO, "Path to the JL_AD database file created ==> " + lclDbPath)

        # Open the DB using JDBC
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
```

```
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, "%.automaticDestinations-ms")

        for file in files:
            file_name = os.path.splitext(file.getName())[0]
            self.log(Level.INFO, "File To process in SQL " + file_name + "  <<=====")
            # Query the table in the database and get all columns.
            try:
                stmt = dbConn.createStatement()
                SQL_Statement = "select File_Name, File_Description, Item_Name, command_line_arguments, drive_type,
drive_serial_number, " + \
                                " description, environment_variables_location, file_access_time, file_attribute_flags, file_creation_time, " + \
                                " file_modification_time, file_size, icon_location, link_target_identifier_data, local_path, " + \
                                " machine_identifier, network_path, relative_path, volume_label, working_directory " + \
                                " from Automatic_destinations_JL where upper(File_Name) = upper('" + file_name + "');"
                resultSet = stmt.executeQuery(SQL_Statement)
            except SQLException as e:
                self.log(Level.INFO, "Error querying database for table (" + e.getMessage() + ")")
                return IngestModule.ProcessResult.OK

            # Cycle through each row and create artifacts
            while resultSet.next():
                try:

                    File_Name = resultSet.getString("File_Name")
                    File_Description = resultSet.getString("File_Description")
                    Item_Name = resultSet.getString("Item_Name")
                    Command_Line_Arguments = resultSet.getString("command_line_arguments")
                    Drive_Type = resultSet.getInt("drive_type")
                    Drive_Serial_Number = resultSet.getInt("drive_serial_number")
                    Description = resultSet.getString("description")
                    Environment_Variables_Location = resultSet.getString("environment_variables_location")
                    File_Access_Time = resultSet.getString("file_access_time")
                    File_Attribute_Flags = resultSet.getInt("file_attribute_flags")
                    File_Creation_Time = resultSet.getString("file_creation_time")
                    File_Modification_Time = resultSet.getString("file_modification_time")
                    File_Size = resultSet.getInt("file_size")
                    Icon_Location = resultSet.getString("icon_location")
                    Link_Target_Identifier_Data = resultSet.getString("link_target_identifier_data")
                    Local_Path = resultSet.getString("local_path")
                    Machine_Identifier = resultSet.getString("machine_identifier")
                    Network_Path = resultSet.getString("network_path")
                    Relative_Path = resultSet.getString("relative_path")
                    Volume_Label = resultSet.getString("volume_label")
                    Working_Directory = resultSet.getString("working_directory")
                except SQLException as e:
                    self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

                art = file.newArtifact(artID_jl_ad)

                # Add attributes
                art.addAttributes(((BlackboardAttribute(attID_jl_fn, Windows_InternalsIngestModuleFactory.moduleName,
File_Name)), \
                                   (BlackboardAttribute(attID_jl_fg, Windows_InternalsIngestModuleFactory.moduleName, File_Description)),
\
                                   (BlackboardAttribute(attID_jl_in, Windows_InternalsIngestModuleFactory.moduleName, Item_Name)), \
                                   (BlackboardAttribute(attID_jl_cl, Windows_InternalsIngestModuleFactory.moduleName,
Command_Line_Arguments)), \
                                   (BlackboardAttribute(attID_jl_dt, Windows_InternalsIngestModuleFactory.moduleName, Drive_Type)), \
                                   (BlackboardAttribute(attID_jl_dsn, Windows_InternalsIngestModuleFactory.moduleName,
Drive_Serial_Number)), \
                                   (BlackboardAttribute(attID_jl_des, Windows_InternalsIngestModuleFactory.moduleName, Description)), \
                                   (BlackboardAttribute(attID_jl_evl, Windows_InternalsIngestModuleFactory.moduleName,
Environment_Variables_Location)), \
                                   (BlackboardAttribute(attID_jl_fat, Windows_InternalsIngestModuleFactory.moduleName,
File_Access_Time)), \
                                   (BlackboardAttribute(attID_jl_faf, Windows_InternalsIngestModuleFactory.moduleName,
File_Attribute_Flags)), \
                                   (BlackboardAttribute(attID_jl_fct, Windows_InternalsIngestModuleFactory.moduleName,
File_Creation_Time)), \
                                   (BlackboardAttribute(attID_jl_fmt, Windows_InternalsIngestModuleFactory.moduleName,
File_Modification_Time)), \
```

```
                    (BlackboardAttribute(attID_jl_fs, Windows_InternalsIngestModuleFactory.moduleName, File_Size)), \
                    (BlackboardAttribute(attID_jl_ic, Windows_InternalsIngestModuleFactory.moduleName, Icon_Location)), \
                    (BlackboardAttribute(attID_jl_ltid, Windows_InternalsIngestModuleFactory.moduleName,
Link_Target_Identifier_Data)), \
                    (BlackboardAttribute(attID_jl_lp, Windows_InternalsIngestModuleFactory.moduleName, Local_Path)), \
                    (BlackboardAttribute(attID_jl_mi, Windows_InternalsIngestModuleFactory.moduleName,
Machine_Identifier)), \
                    (BlackboardAttribute(attID_jl_np, Windows_InternalsIngestModuleFactory.moduleName, Network_Path)), \
                    (BlackboardAttribute(attID_jl_rp, Windows_InternalsIngestModuleFactory.moduleName, Relative_Path)), \
                    (BlackboardAttribute(attID_jl_vl, Windows_InternalsIngestModuleFactory.moduleName, Volume_Label)), \
                    (BlackboardAttribute(attID_jl_wd, Windows_InternalsIngestModuleFactory.moduleName,
Working_Directory))))

            try:
              # index the artifact for keyword search
              blackboard.indexArtifact(art)
            except:
              self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())

          # Fire an event to notify the UI and others that there are new artifacts
          IngestServices.getInstance().fireModuleDataEvent(
            ModuleDataEvent(Windows_InternalsIngestModuleFactory.moduleName, artID_jl_ad_evt, None))

          # Clean up
          #skCase_Tran.commit()
          stmt.close()

        dbConn.close()
        os.remove(lclDbPath)

        #Clean up EventLog directory and files
        try:
            shutil.rmtree(Temp_Dir + "\\JL_AD")
        except:
            self.log(Level.INFO, "removal of JL_AD directory failed " + Temp_Dir)

    def process_Prefetch(self, dataSource, progressBar):
        # Check to see if the artifacts exist and if not then create it, also check to see if the attributes
        # exist and if not then create them
        skCase = Case.getCurrentCase().getSleuthkitCase();

        # This will work in 4.0.1 and beyond
        # Use blackboard class to index blackboard artifacts for keyword search
        blackboard = Case.getCurrentCase().getServices().getBlackboard()

        try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_pf = skCase.addArtifactType( "TSK_PREFETCH", "Windows Prefetch")
        except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
            artID_pf = skCase.getArtifactTypeID("TSK_PREFETCH")

        # Create the attribute type, if it exists then catch the error
        try:
          attID_pf_fn = skCase.addArtifactAttributeType("TSK_PREFETCH_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Prefetch File Name")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Prefetch File Name. ==> ")

        try:
          attID_pf_an = skCase.addArtifactAttributeType("TSK_PREFETCH_ACTUAL_FILE_NAME",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Actual File Name")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Actual File Name. ==> ")

        try:
          attID_nr = skCase.addArtifactAttributeType("TSK_PF_RUN_COUNT",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, "Program Number Runs")
        except:
            self.log(Level.INFO, "Attributes Creation Error, Program Number Runs. ==> ")

        try:
          attID_ex1 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_1",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 1")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 1. ==> ")
```

```
        try:
            attID_ex2 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_2",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 2")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 2. ==> ")

        try:
            attID_ex3 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_3",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 3")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 3. ==> ")

        try:
            attID_ex4 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_4",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 4")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 4 ==> ")

        try:
            attID_ex5 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_5",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 5")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 5. ==> ")

        try:
            attID_ex6 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_6",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 6")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 6. ==> ")

        try:
            attID_ex7 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_7",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 7")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 7. ==> ")

        try:
            attID_ex8 = skCase.addArtifactAttributeType("TSK_PF_EXEC_DTTM_8",
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.DATETIME, "PF Execution DTTM 8")
        except:
            self.log(Level.INFO, "Attributes Creation Error, PF Execution DTTM 8 ==> ")

        self.log(Level.INFO, "Get Artifacts after they were created.")
        # Get the new artifacts and attributes that were just created
        artID_pf = skCase.getArtifactTypeID("TSK_PREFETCH")
        artID_pf_evt = skCase.getArtifactType("TSK_PREFETCH")
        attID_pf_fn = skCase.getAttributeType("TSK_PREFETCH_FILE_NAME")
        attID_pf_an = skCase.getAttributeType("TSK_PREFETCH_ACTUAL_FILE_NAME")
        attID_nr = skCase.getAttributeType("TSK_PF_RUN_COUNT")
        attID_ex1 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_1")
        attID_ex2 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_2")
        attID_ex3 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_3")
        attID_ex4 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_4")
        attID_ex5 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_5")
        attID_ex6 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_6")
        attID_ex7 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_7")
        attID_ex8 = skCase.getAttributeType("TSK_PF_EXEC_DTTM_8")

        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Find the prefetch files and the layout.ini file from the /windows/prefetch folder
        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, "%.pf")

        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        # Create prefetch directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory() + "\Prefetch_Files"
        self.log(Level.INFO, "create Directory " + Temp_Dir)
        try:
            os.mkdir(Temp_Dir)
```

```
        except:
            self.log(Level.INFO, "Prefetch Directory already exists " + Temp_Dir)

        # Write out each prefetch file to the temp directory
        for file in files:

            # Check if the user pressed cancel while we were busy
            if self.context.isJobCancelled():
                return IngestModule.ProcessResult.OK

            #self.log(Level.INFO, "Processing file: " + file.getName())
            fileCount += 1

            # Save the DB locally in the temp folder. use file id as name to reduce collisions
            lclDbPath = os.path.join(Temp_Dir, file.getName())
            ContentUtils.writeToFile(file, File(lclDbPath))


        # Example has only a Windows EXE, so bail if we aren't on Windows
        if not PlatformUtil.isWindowsOS():
            self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
            return IngestModule.ProcessResult.OK

        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "  Parm 2 ==> " +
Case.getCurrentCase().getTempDirectory())
        pipe = Popen([self.path_to_Prefetch_file, Temp_Dir, Case.getCurrentCase().getTempDirectory()], stdout=PIPE, stderr=PIPE)

        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Set the database to be read to the once created by the prefetch parser program
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "Autopsy_PF_DB.db3")
        self.log(Level.INFO, "Path the prefetch database file created ==> " + lclDbPath)

        # Open the DB using JDBC
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the contacts table in the database and get all columns.
        try:
            stmt = dbConn.createStatement()
            resultSet = stmt.executeQuery("Select prefetch_File_Name, actual_File_Name, Number_time_file_run, " +
                                          " Embeded_date_Time_Unix_1, " +
                                          " Embeded_date_Time_Unix_2, " +
                                          " Embeded_date_Time_Unix_3, " +
                                          " Embeded_date_Time_Unix_4, " +
                                          " Embeded_date_Time_Unix_5, " +
                                          " Embeded_date_Time_Unix_6, " +
                                          " Embeded_date_Time_Unix_7, " +
                                          " Embeded_date_Time_Unix_8 " +
                                          " from prefetch_file_info ")
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for Prefetch table (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Cycle through each row and create artifacts
        while resultSet.next():
            try:
                self.log(Level.INFO, "Result (" + resultSet.getString("Prefetch_File_Name") + ")")
                Prefetch_File_Name  = resultSet.getString("Prefetch_File_Name")
                Actual_File_Name = resultSet.getString("Actual_File_Name")
                Number_Of_Runs = resultSet.getString("Number_Time_File_Run")
                Time_1 = resultSet.getInt("Embeded_date_Time_Unix_1")
                Time_2 = resultSet.getInt("Embeded_date_Time_Unix_2")
                Time_3 = resultSet.getInt("Embeded_date_Time_Unix_3")
                Time_4 = resultSet.getInt("Embeded_date_Time_Unix_4")
                Time_5 = resultSet.getInt("Embeded_date_Time_Unix_5")
                Time_6 = resultSet.getInt("Embeded_date_Time_Unix_6")
                Time_7 = resultSet.getInt("Embeded_date_Time_Unix_7")
                Time_8 = resultSet.getInt("Embeded_date_Time_Unix_8")
            except SQLException as e:
```

```
            self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, Prefetch_File_Name)

        for file in files:
          # Make artifact for TSK_PREFETCH,  this can happen when custom attributes are fully supported
          art = file.newArtifact(artID_pf)

          # Add the attributes to the artifact.
          art.addAttributes(((BlackboardAttribute(attID_pf_fn, Windows_InternalsIngestModuleFactory.moduleName,
Prefetch_File_Name)), \
                      (BlackboardAttribute(attID_pf_an, Windows_InternalsIngestModuleFactory.moduleName,
Actual_File_Name)), \
                      (BlackboardAttribute(attID_nr, Windows_InternalsIngestModuleFactory.moduleName, Number_Of_Runs)), \
                      (BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName, Time_1)), \
                      (BlackboardAttribute(attID_ex2, Windows_InternalsIngestModuleFactory.moduleName, Time_2)), \
                      (BlackboardAttribute(attID_ex3, Windows_InternalsIngestModuleFactory.moduleName, Time_3)), \
                      (BlackboardAttribute(attID_ex4, Windows_InternalsIngestModuleFactory.moduleName, Time_4)), \
                      (BlackboardAttribute(attID_ex5, Windows_InternalsIngestModuleFactory.moduleName, Time_5)), \
                      (BlackboardAttribute(attID_ex6, Windows_InternalsIngestModuleFactory.moduleName, Time_6)), \
                      (BlackboardAttribute(attID_ex7, Windows_InternalsIngestModuleFactory.moduleName, Time_7)), \
                      (BlackboardAttribute(attID_ex8, Windows_InternalsIngestModuleFactory.moduleName, Time_8))))

          try:
            #index the artifact for keyword search
            blackboard.indexArtifact(art)
          except:
            self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())


IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(Windows_InternalsIngestModuleFactory.moduleName,
artID_pf_evt, None))
      # Clean up
      stmt.close()
      dbConn.close()
      os.remove(lclDbPath)

      #Clean up prefetch directory and files
      try:
          shutil.rmtree(Temp_Dir)
      except:
          self.log(Level.INFO, "removal of directory tree failed " + Temp_Dir)

    def process_SAM(self, dataSource, progressBar):
      # we don't know how much work there is yet
      progressBar.switchToIndeterminate()

      # Set the database to be read to the once created by the SAM parser program
      skCase = Case.getCurrentCase().getSleuthkitCase()

      # Use blackboard class to index blackboard artifacts for keyword search
      blackboard = Case.getCurrentCase().getServices().getBlackboard()

      fileManager = Case.getCurrentCase().getServices().getFileManager()
      files = fileManager.findFiles(dataSource, "SAM", "config")
      numFiles = len(files)
      self.log(Level.INFO, "found " + str(numFiles) + " files")
      progressBar.switchToDeterminate(numFiles)
      fileCount = 0;

      # Create Event Log directory in temp directory, if it exists then continue on processing
      Temp_Dir = Case.getCurrentCase().getTempDirectory()
      self.log(Level.INFO, "create Directory " + Temp_Dir)
      try:
        os.mkdir(Temp_Dir + "\SAM")
      except:
        self.log(Level.INFO, "SAM Directory already exists " + Temp_Dir)

      # Write out each SAM file to the temp directory
      for file in files:

        # Check if the user pressed cancel while we were busy
        if self.context.isJobCancelled():
          return IngestModule.ProcessResult.OK
```

```
        #self.log(Level.INFO, "Processing file: " + file.getName())
        fileCount += 1

        # Save the DB locally in the temp folder. use file id as name to reduce collisions
        lclDbPath = os.path.join(Temp_Dir + "\\SAM", file.getName())
        ContentUtils.writeToFile(file, File(lclDbPath))


    # Example has only a Windows EXE, so bail if we aren't on Windows
    if not PlatformUtil.isWindowsOS():
        self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
        return IngestModule.ProcessResult.OK

    # Run the EXE, saving output to a sqlite database
    self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "  Parm 2 ==> " + Temp_Dir +
"\\SAM.db3")
    pipe = Popen([self.path_to_SAM_file, Temp_Dir + "\\SAM\\SAM", Temp_Dir + "\\SAM.db3"], stdout=PIPE, stderr=PIPE)
    out_text = pipe.communicate()[0]
    self.log(Level.INFO, "Output from run is ==> " + out_text)

    for file in files:
      # Open the DB using JDBC
      lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "SAM.db3")
      self.log(Level.INFO, "Path the SAM database file created ==> " + lclDbPath)
      try:
        Class.forName("org.sqlite.JDBC").newInstance()
        dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
      except SQLException as e:
        self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

      # Query the contacts table in the database and get all columns.
      try:
        stmt = dbConn.createStatement()
        resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER; ")
        self.log(Level.INFO, "query SQLite Master table")
      except SQLException as e:
        self.log(Level.INFO, "Error querying database for SAM table (" + e.getMessage() + ")")
        return IngestModule.ProcessResult.OK

      try:
         self.log(Level.INFO, "Begin Create New Artifacts")
         artID_sam = skCase.addArtifactType( "TSK_SAM", "SAM File")
      except:
         self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

      artID_sam = skCase.getArtifactTypeID("TSK_SAM")
      artID_sam_evt = skCase.getArtifactType("TSK_SAM")

      # Cycle through each row and create artifacts
      while resultSet.next():
        try:
          self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
          table_name = resultSet.getString("tbl_name")
          SQL_String_1 = "Select * from " + table_name + ";"
          SQL_String_2 = "PRAGMA table_info('" + table_name + "')"

          Column_Names = []
          Column_Types = []
          resultSet2  = stmt.executeQuery(SQL_String_2)
          while resultSet2.next():
            Column_Names.append(resultSet2.getString("name").upper())
            Column_Types.append(resultSet2.getString("type"))
            if resultSet2.getString("type") == "text":
              try:
                attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
              except:
                self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
            else:
              try:
                attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
              except:
                self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
```

```
            resultSet3 = stmt.executeQuery(SQL_String_1)
            while resultSet3.next():
              art = file.newArtifact(artID_sam)
              Column_Number = 1
              for col_name in Column_Names:
                c_name = "TSK_" + col_name
                attID_ex1 = skCase.getAttributeType(c_name)
                if Column_Types[Column_Number - 1] == "text":
                  art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                else:
                  art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getInt(Column_Number)))
                Column_Number = Column_Number + 1

              # index the artifact for keyword search
              try:
                blackboard.indexArtifact(art)
              except:
                self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())

          except SQLException as e:
            self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

        # Clean up
        stmt.close()
        dbConn.close()
        os.remove(lclDbPath)

        #Clean up EventLog directory and files
        try:
          shutil.rmtree(Temp_Dir)
        except:
          self.log(Level.INFO, "removal of directory tree failed " + Temp_Dir)

      def process_Shimcache(self, dataSource, progressBar):
        # we don't know how much work there is yet
        progressBar.switchToIndeterminate()

        # Set the database to be read to the once created by the SAM parser program
        skCase = Case.getCurrentCase().getSleuthkitCase();

        # Use blackboard class to index blackboard artifacts for keyword search
        blackboard = Case.getCurrentCase().getServices().getBlackboard()

        fileManager = Case.getCurrentCase().getServices().getFileManager()
        files = fileManager.findFiles(dataSource, "SYSTEM", "config")
        numFiles = len(files)
        self.log(Level.INFO, "found " + str(numFiles) + " files")
        progressBar.switchToDeterminate(numFiles)
        fileCount = 0;

        # Create Event Log directory in temp directory, if it exists then continue on processing
        Temp_Dir = Case.getCurrentCase().getTempDirectory()
        self.log(Level.INFO, "create Directory " + Temp_Dir)
        try:
          os.mkdir(Temp_Dir + "/Shimcache")
        except:
          self.log(Level.INFO, "Shimcache Directory already exists " + Temp_Dir)

        for file in files:
          # Check if the user pressed cancel while we were busy
          if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK

          #self.log(Level.INFO, "Processing file: " + file.getName())
          fileCount += 1

          # Save the DB locally in the temp folder. use file id as name to reduce collisions
          lclDbPath = os.path.join(Temp_Dir + "\\Shimcache\\", file.getName())
          ContentUtils.writeToFile(file, File(lclDbPath))
          self.log(Level.INFO, "Saved File ==> " + lclDbPath)

          if not PlatformUtil.isWindowsOS():
            self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
            return IngestModule.ProcessResult.OK
```

```
        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program ==> " + self.path_to_Shimcache_file + " " + Temp_Dir + "\\Shimcache\\" + \
                file.getName() + " " + Temp_Dir + "\\Shimcache_db.db3")
        pipe = Popen([self.path_to_Shimcache_file, Temp_Dir + "\\Shimcache\\" + file.getName(), Temp_Dir + \
                    "\\Shimcache_db.db3"], stdout=PIPE, stderr=PIPE)
        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Open the DB using JDBC
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "Shimcache_db.db3")
        self.log(Level.INFO, "Path the system database file created ==> " + lclDbPath)

        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the contacts table in the database and get all columns.
        try:
            stmt = dbConn.createStatement()
            resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER; ")
            self.log(Level.INFO, "query SQLite Master table")
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for system table (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        try:
             self.log(Level.INFO, "Begin Create New Artifacts")
             artID_shim = skCase.addArtifactType("TSK_SHIMCACHE", "Shimcache")
        except:
             self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

        artID_shim = skCase.getArtifactTypeID("TSK_SHIMCACHE")
        artID_shim_evt = skCase.getArtifactType("TSK_SHIMCACHE")


        # Cycle through each row and create artifacts
        while resultSet.next():
            try:
                self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
                table_name = resultSet.getString("tbl_name")
                SQL_String_1 = "Select * from " + table_name + ";"
                SQL_String_2 = "PRAGMA table_info('" + table_name + "')"

                Column_Names = []
                Column_Types = []
                resultSet2  = stmt.executeQuery(SQL_String_2)
                while resultSet2.next():
                    Column_Names.append(resultSet2.getString("name").upper())
                    Column_Types.append(resultSet2.getString("type"))
                    if resultSet2.getString("type").upper() == "TEXT":
                        try:
                            attID_ex1 = skCase.addArtifactAttributeType("TSK_SHIMCACHE_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                            self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
                    else:
                        try:
                            attID_ex1 = skCase.addArtifactAttributeType("TSK_SHIMCACHE_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                            self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")

                resultSet3 = stmt.executeQuery(SQL_String_1)
                while resultSet3.next():
                    art = file.newArtifact(artID_shim)
                    Column_Number = 1
                    for col_name in Column_Names:
                        c_name = "TSK_SHIMCACHE_" + col_name
                        attID_ex1 = skCase.getAttributeType(c_name)
                        if Column_Types[Column_Number - 1] == "TEXT":
                            art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
```

```
                else:
                    art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getInt(Column_Number)))
                Column_Number = Column_Number + 1

            # index the artifact for keyword search
            try:
                blackboard.indexArtifact(art)
            except:
                self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())

        except SQLException as e:
            self.log(Level.INFO, "Error getting values from Shimcache table (" + e.getMessage() + ")")

    # Clean up
    stmt.close()
    dbConn.close()
    os.remove(lclDbPath)

    try:
        shutil.rmtree(Temp_Dir + "\\Shimcache")
    except:
        self.log(Level.INFO, "removal of directory tree failed " + Temp_Dir + "\\Shimcache")

def process_Usnj(self, dataSource, progressBar):
    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    # Set the database to be read to the once created by the SAM parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();

    # Use blackboard class to index blackboard artifacts for keyword search
    blackboard = Case.getCurrentCase().getServices().getBlackboard()

    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "$UsnJrnl:$J", "$Extend")
    numFiles = len(files)
    self.log(Level.INFO, "found " + str(numFiles) + " files")
    progressBar.switchToDeterminate(numFiles)
    fileCount = 0;

    # Create Event Log directory in temp directory, if it exists then continue on processing
    Temp_Dir = Case.getCurrentCase().getTempDirectory()
    self.log(Level.INFO, "create Directory " + Temp_Dir)
    try:
        os.mkdir(Temp_Dir + "/usnj")
    except:
        self.log(Level.INFO, "Usnj Directory already exists " + Temp_Dir)

    for file in files:
        # Check if the user pressed cancel while we were busy
        if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK

        #self.log(Level.INFO, "Processing file: " + file.getName())
        fileCount += 1

        # Save the DB locally in the temp folder. use file id as name to reduce collisions
        lclDbPath = os.path.join(Temp_Dir + "\\usnj\\", "usnj.txt")
        ContentUtils.writeToFile(file, File(lclDbPath))
        self.log(Level.INFO, "Saved File ==> " + lclDbPath)

        if not PlatformUtil.isWindowsOS():
            self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
            return IngestModule.ProcessResult.OK

        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program ==> " + self.path_to_Usnj_file + " " + Temp_Dir + "\\usnj\\usnj.txt" + \
                " " + Temp_Dir + "\\usnj.db3")
        pipe = Popen([self.path_to_Usnj_file, Temp_Dir + "\\usnj\\usnj.txt", Temp_Dir + "\\usnj.db3"], stdout=PIPE, stderr=PIPE)
        out_text = pipe.communicate()[0]
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Open the DB using JDBC
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "usnj.db3")
        self.log(Level.INFO, "Path the system database file created ==> " + lclDbPath)
```

```
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) usnj.db3 (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        # Query the contacts table in the database and get all columns.
        try:
            stmt = dbConn.createStatement()
            resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER; ")
            self.log(Level.INFO, "query SQLite Master table")
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for system table (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_usnj = skCase.addArtifactType("TSK_USNJ", "NTFS UsrJrnl entries")
        except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

        artID_usnj = skCase.getArtifactTypeID("TSK_USNJ")
        artID_usnj_evt = skCase.getArtifactType("TSK_USNJ")


        # Cycle through each row and create artifacts
        while resultSet.next():
            try:
                self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
                table_name = resultSet.getString("tbl_name")
                SQL_String_1 = "Select * from " + table_name + ";"
                SQL_String_2 = "PRAGMA table_info('" + table_name + "')"

                Column_Names = []
                Column_Types = []
                resultSet2  = stmt.executeQuery(SQL_String_2)
                while resultSet2.next():
                    Column_Names.append(resultSet2.getString("name").upper())
                    Column_Types.append(resultSet2.getString("type"))
                    if resultSet2.getString("type").upper() == "TEXT":
                        try:
                            attID_ex1 = skCase.addArtifactAttributeType("TSK_USNJ_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                        except:
                            self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
                    else:
                        try:
                            attID_ex1 = skCase.addArtifactAttributeType("TSK_USNJ_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                        except:
                            self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")

                resultSet3 = stmt.executeQuery(SQL_String_1)
                while resultSet3.next():
                    art = file.newArtifact(artID_usnj)
                    Column_Number = 1
                    for col_name in Column_Names:
                        c_name = "TSK_USNJ_" + col_name
                        attID_ex1 = skCase.getAttributeType(c_name)
                        if Column_Types[Column_Number - 1] == "TEXT":
                            art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                        else:
                            art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getInt(Column_Number)))
                        Column_Number = Column_Number + 1

                    # index the artifact for keyword search
                    try:
                        blackboard.indexArtifact(art)
                    except:
                        self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())

            except SQLException as e:
```

```
                self.log(Level.INFO, "Error getting values from Shimcache table (" + e.getMessage() + ")")

        # Clean up
          stmt.close()
          dbConn.close()
          # Fire an event to notify the UI and others that there are new artifacts
          IngestServices.getInstance().fireModuleDataEvent(
             ModuleDataEvent(Windows_InternalsIngestModuleFactory.moduleName, artID_usnj_evt, None))

          #Clean up EventLog directory and files
          os.remove(lclDbPath)

        try:
            shutil.rmtree(Temp_Dir)
        except:
            self.log(Level.INFO, "removal of directory tree failed " + Temp_Dir)

    def process_Webcache(self, dataSource, progressBar):
      # we don't know how much work there is yet
       progressBar.switchToIndeterminate()

      # Set the database to be read to the once created by the prefetch parser program
       skCase = Case.getCurrentCase().getSleuthkitCase();

       # Use blackboard class to index blackboard artifacts for keyword search
       blackboard = Case.getCurrentCase().getServices().getBlackboard()

       fileManager = Case.getCurrentCase().getServices().getFileManager()
       files = fileManager.findFiles(dataSource, "WebcacheV01.dat")
       numFiles = len(files)
       self.log(Level.INFO, "found " + str(numFiles) + " files")
       progressBar.switchToDeterminate(numFiles)
       fileCount = 0;

       # Create Event Log directory in temp directory, if it exists then continue on processing
       Temp_Dir = Case.getCurrentCase().getTempDirectory()
       self.log(Level.INFO, "create Directory " + Temp_Dir)
       try:
          os.mkdir(Temp_Dir + "\Webcache")
       except:
          self.log(Level.INFO, "Webcache Directory already exists " + Temp_Dir)

       # Write out each Event Log file to the temp directory
       for file in files:

          # Check if the user pressed cancel while we were busy
          if self.context.isJobCancelled():
             return IngestModule.ProcessResult.OK

          #self.log(Level.INFO, "Processing file: " + file.getName())
          fileCount += 1

          # Save the DB locally in the temp folder. use file id as name to reduce collisions
          lclDbPath = os.path.join(Temp_Dir + "\Webcache", file.getName() + "-" + str(file.getId()))
          DbPath = os.path.join(Temp_Dir, file.getName() + "-" + str(file.getId()) + ".db3")
          self.log(Level.INFO, file.getName() + ' ==> ' + str(file.getId()) + ' ==> ' + file.getUniquePath())
          ContentUtils.writeToFile(file, File(lclDbPath))

          # Run the EXE, saving output to a sqlite database
          self.log(Level.INFO, "Running program on data source parm 1 ==> " + Temp_Dir + "  Parm 2 ==> " + Temp_Dir +
"\WebcacheV01.db3")
          #subprocess.Popen([self.path_to_Webcache_file, lclDbPath, DbPath]).communicate()[0]
          pipe = Popen([self.path_to_Webcache_file, lclDbPath, DbPath], stdout=PIPE, stderr=PIPE)
          out_text = pipe.communicate()[0]
          self.log(Level.INFO, "Output from run is ==> " + out_text)

       # Example has only a Windows EXE, so bail if we aren't on Windows
       if not PlatformUtil.isWindowsOS():
          self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
          return IngestModule.ProcessResult.OK


       for file in files:
          # Open the DB using JDBC
          lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), file.getName() + "-" + str(file.getId()) + ".db3")
          self.log(Level.INFO, "Path the Webcache database file created ==> " + lclDbPath)
```

```
        try:
            Class.forName("org.sqlite.JDBC").newInstance()
            dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
            self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        try:
            stmt = dbConn.createStatement()
            resultSet = stmt.executeQuery("Select distinct container_name from all_containers;")
            self.log(Level.INFO, "query SQLite Master table")
        except SQLException as e:
            self.log(Level.INFO, "Error querying database for Prefetch table (" + e.getMessage() + ")")
            return IngestModule.ProcessResult.OK

        Container_List = []
        while resultSet.next():
          Container_List.append(resultSet.getString("container_name"))

        # Cycle through each row and create artifacts
        for c_name in Container_List:
            try:
                container_name = c_name
                SQL_String_1 = "Select * from all_containers where container_name = '" + container_name + "';"
                SQL_String_2 = "PRAGMA table_info('All_Containers')"
                artifact_name = "TSK_WC_" + container_name.upper()
                artifact_desc = "WebcacheV01 " + container_name.upper()
                try:
                    self.log(Level.INFO, "Begin Create New Artifacts")
                    artID_web = skCase.addArtifactType( artifact_name, artifact_desc)
                except:
                    self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")
                artID_web = skCase.getArtifactTypeID(artifact_name)
                artID_web_evt = skCase.getArtifactType(artifact_name)

                Column_Names = []
                Column_Types = []
                resultSet2  = stmt.executeQuery(SQL_String_2)
                while resultSet2.next():
                  Column_Names.append(resultSet2.getString("name").upper())
                  Column_Types.append(resultSet2.getString("type").upper())
                  if resultSet2.getString("type").upper() == "TEXT":
                    try:
                      attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute. TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                    except:
                      self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
                  elif resultSet2.getString("type").upper() == "":
                    try:
                      attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                    except:
                      self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
                  else:
                    try:
                      attID_ex1 = skCase.addArtifactAttributeType("TSK_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                    except:
                      self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")


                resultSet3 = stmt.executeQuery(SQL_String_1)
                while resultSet3.next():
                  art = file.newArtifact(artID_web)
                  Column_Number = 1
                  for col_name in Column_Names:
                    c_name = "TSK_" + col_name
                    attID_ex1 = skCase.getAttrTypeID(c_name)
                    attID_ex1 = skCase.getAttributeType(c_name)
                    if Column_Types[Column_Number - 1] == "TEXT":
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                    elif Column_Types[Column_Number - 1] == "":
                      art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
```

```
                else:
                    art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
long(resultSet3.getInt(Column_Number))))
                    Column_Number = Column_Number + 1

                # index the artifact for keyword search
                try:
                    blackboard.indexArtifact(art)
                except:
                    self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())

                IngestServices.getInstance().fireModuleDataEvent(
                        ModuleDataEvent(Windows_InternalsIngestModuleFactory.moduleName, artID_web_evt, None))
            except SQLException as e:
                self.log(Level.INFO, "Error getting values from contacts table (" + e.getMessage() + ")")

        stmt.close()
        dbConn.close()

    #Clean up EventLog directory and files
    for file in files:
        try:
            os.remove(Temp_Dir + "\\" + file.getName() + "-" + str(file.getId()) + ".db3")
        except:
            self.log(Level.INFO, "removal of Webcache file failed " + Temp_Dir + "\\" + file.getName() + "-" + str(file.getId()))
    try:
        shutil.rmtree(Temp_Dir + "\\Webcache")
    except:
        self.log(Level.INFO, "removal of Webcache directory failed " + Temp_Dir)

def process_Shellbags(self, dataSource, progressBar):
    # we don't know how much work there is yet
    progressBar.switchToIndeterminate()

    # Set the database to be read to the once created by the SAM parser program
    skCase = Case.getCurrentCase().getSleuthkitCase();

    # Use blackboard class to index blackboard artifacts for keyword search
    blackboard = Case.getCurrentCase().getServices().getBlackboard()

    fileManager = Case.getCurrentCase().getServices().getFileManager()
    files = fileManager.findFiles(dataSource, "ntuser.dat", "")
    numFiles = len(files)
    self.log(Level.INFO, "found " + str(numFiles) + " files")
    progressBar.switchToDeterminate(numFiles)
    fileCount = 0;

    # Create Event Log directory in temp directory, if it exists then continue on processing
    Temp_Dir = Case.getCurrentCase().getTempDirectory()
    self.log(Level.INFO, "create Directory " + Temp_Dir)
    try:
        os.mkdir(Temp_Dir + "/shellbag")
    except:
        self.log(Level.INFO, "Shellbag Directory already exists " + Temp_Dir)

    for file in files:
        # Check if the user pressed cancel while we were busy
        if self.context.isJobCancelled():
            return IngestModule.ProcessResult.OK
        fileCount += 1

        # Save the DB locally in the temp folder. use file id as name to reduce collisions
        lclDbPath = os.path.join(Temp_Dir + "\\shellbag\\", file.getName())
        ContentUtils.writeToFile(file, File(lclDbPath))
        self.log(Level.INFO, "Saved File ==> " + lclDbPath)

        if not PlatformUtil.isWindowsOS():
            self.log(Level.INFO, "Ignoring data source.  Not running on Windows")
            return IngestModule.ProcessResult.OK

        # Run the EXE, saving output to a sqlite database
        self.log(Level.INFO, "Running program ==> " + self.path_to_Shellbags_file + " " + Temp_Dir + "\\shellbag\\" + \
                file.getName() + " " + Temp_Dir + "\\shellbag_db.db3 " + file.getUniquePath())
        pipe = Popen([self.path_to_Shellbags_file, Temp_Dir + "\\shellbag\\" + file.getName(), Temp_Dir + \
                      "\\Shellbag_db.db3", file.getUniquePath()], stdout=PIPE, stderr=PIPE)
        out_text = pipe.communicate()[0]
```

```
        self.log(Level.INFO, "Output from run is ==> " + out_text)

        # Open the DB using JDBC
        lclDbPath = os.path.join(Case.getCurrentCase().getTempDirectory(), "shellbag_db.db3")
        self.log(Level.INFO, "Path the system database file created ==> " + lclDbPath)

        try:
           Class.forName("org.sqlite.JDBC").newInstance()
           dbConn = DriverManager.getConnection("jdbc:sqlite:%s"  % lclDbPath)
        except SQLException as e:
           self.log(Level.INFO, "Could not open database file (not SQLite) " + file.getName() + " (" + e.getMessage() + ")")
           return IngestModule.ProcessResult.OK

        # Query the contacts table in the database and get all columns.
        try:
           stmt = dbConn.createStatement()
           resultSet = stmt.executeQuery("Select tbl_name from SQLITE_MASTER; ")
           self.log(Level.INFO, "query SQLite Master table")
        except SQLException as e:
           self.log(Level.INFO, "Error querying database for system table (" + e.getMessage() + ")")
           return IngestModule.ProcessResult.OK

        try:
            self.log(Level.INFO, "Begin Create New Artifacts")
            artID_shell = skCase.addArtifactType("TSK_SHELLBAGS", "Shellbags")
        except:
            self.log(Level.INFO, "Artifacts Creation Error, some artifacts may not exist now. ==> ")

        artID_shell = skCase.getArtifactTypeID("TSK_SHELLBAGS")
        artID_shell_evt = skCase.getArtifactType("TSK_SHELLBAGS")


        # Cycle through each row and create artifacts
        while resultSet.next():
           try:
              self.log(Level.INFO, "Result (" + resultSet.getString("tbl_name") + ")")
              table_name = resultSet.getString("tbl_name")
              SQL_String_1 = "Select * from " + table_name + ";"
              SQL_String_2 = "PRAGMA table_info('" + table_name + "')"

              Column_Names = []
              Column_Types = []
              resultSet2  = stmt.executeQuery(SQL_String_2)
              while resultSet2.next():
                 Column_Names.append(resultSet2.getString("name").upper())
                 Column_Types.append(resultSet2.getString("type"))
                 if resultSet2.getString("type").upper() == "TEXT":
                    try:
                       attID_ex1 = skCase.addArtifactAttributeType("TSK_SHELLBAG_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING, resultSet2.getString("name"))
                    except:
                       self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")
                 else:
                    try:
                       attID_ex1 = skCase.addArtifactAttributeType("TSK_SHELLBAG_" + resultSet2.getString("name").upper(),
BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.LONG, resultSet2.getString("name"))
                    except:
                       self.log(Level.INFO, "Attributes Creation Error, " + resultSet2.getString("name") + " ==> ")

              resultSet3 = stmt.executeQuery(SQL_String_1)
              while resultSet3.next():
                 art = file.newArtifact(artID_shell)
                 Column_Number = 1
                 for col_name in Column_Names:
                    c_name = "TSK_SHELLBAG_" + col_name
                    attID_ex1 = skCase.getAttributeType(c_name)
                    if Column_Types[Column_Number - 1] == "TEXT":
                       art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getString(Column_Number)))
                    else:
                       art.addAttribute(BlackboardAttribute(attID_ex1, Windows_InternalsIngestModuleFactory.moduleName,
resultSet3.getInt(Column_Number)))
                    Column_Number = Column_Number + 1

                 # index the artifact for keyword search
                 try:
```

```
                    blackboard.indexArtifact(art)
                except:
                    self.log(Level.SEVERE, "Error indexing artifact " + art.getDisplayName())

            except SQLException as e:
                self.log(Level.INFO, "Error getting values from Shellbag table (" + e.getMessage() + ")")

        # Clean up
        stmt.close()
        dbConn.close()
        # Fire an event to notify the UI and others that there are new artifacts
        IngestServices.getInstance().fireModuleDataEvent(
            ModuleDataEvent(Windows_InternalsIngestModuleFactory.moduleName, artID_shell_evt, None))

        #Clean up EventLog directory and files
        #os.remove(lclDbPath)
        for file in files:
            try:
                os.remove(Temp_Dir + "\\Shellbag\\" + file.getName())
            except:
                self.log(Level.INFO, "removal of shellbag file failed " + Temp_Dir + "\\" + file.getName())
        try:
            os.remove(Temp_Dir + "\\Shellbag_db.db3")
            shutil.rmtree(Temp_Dir + "\\Shellbag")
        except:
            self.log(Level.INFO, "removal of Shellbag directory failed " + Temp_Dir)


# Stores the settings that can be changed for each ingest job
# All fields in here must be serializable.  It will be written to disk.
class Windows_InternalsWithUISettings(IngestModuleIngestJobSettings):
    serialVersionUID = 1L

    def __init__(self):
        self.Recentlyused_Flag = False
        self.Jumplist_Flag = False
        self.Prefetch_Flag = False
        self.SAM_Flag = False
        self.Shellbags_Flag = False
        self.Shimcache_Flag = False
        self.Usnj_Flag = False
        self.Webcache_Flag = False
        self.Filehistory_Flag = False

    def getVersionNumber(self):
        return serialVersionUID

    # Define getters and settings for data you want to store from UI
    def getRecentlyused_Flag(self):
        return self.Recentlyused_Flag

    def setRecentlyused_Flag(self, flag):
        self.Recentlyused_Flag = flag

    def getJumplist_Flag(self):
        return self.Jumplist_Flag

    def setJumplist_Flag(self, flag):
        self.Jumplist_Flag = flag

    def getPrefetch_Flag(self):
        return self.Prefetch_Flag

    def setPrefetch_Flag(self, flag):
        self.Prefetch_Flag = flag

    def getSAM_Flag(self):
        return self.SAM_Flag

    def setSAM_Flag(self, flag):
        self.SAM_Flag = flag

    def getShellbags_Flag(self):
        return self.Shellbags_Flag

    def setShellbags_Flag(self, flag):
```

```
        self.Shellbags_Flag = flag

    def getShimcache_Flag(self):
        return self.Shimcache_Flag

    def setShimcache_Flag(self, flag):
        self.Shimcache_Flag = flag

    def getUsnj_Flag(self):
        return self.Usnj_Flag

    def setUsnj_Flag(self, flag):
        self.Usnj_Flag = flag

    def getWebcache_Flag(self):
        return self.Webcache_Flag

    def setWebcache_Flag(self, flag):
        self.Webcache_Flag = flag

    def getFilehistory_Flag(self):
        return self.Filehistory_Flag

    def setFilehistory_Flag(self, flag):
        self.Filehistory_Flag = flag

# UI that is shown to user for each ingest job so they can configure the job.
class Windows_InternalsWithUISettingsPanel(IngestModuleIngestJobSettingsPanel):
    # Note, we can't use a self.settings instance variable.
    # Rather, self.local_settings is used.
    # https://wiki.python.org/jython/UserGuide#javabean-properties
    # Jython Introspector generates a property - 'settings' on the basis
    # of getSettings() defined in this class. Since only getter function
    # is present, it creates a read-only 'settings' property. This auto-
    # generated read-only property overshadows the instance-variable -
    # 'settings'

    # We get passed in a previous version of the settings so that we can
    # prepopulate the UI
    def __init__(self, settings):
        self.local_settings = settings
        self.initComponents()
        self.customizeComponents()

    def checkBoxEvent(self, event):

        if self.Recentlyused_CB.isSelected():
            self.local_settings.setRecentlyused_Flag(True)
        else:
            self.local_settings.setRecentlyused_Flag(False)

        if self.Filehistory_CB.isSelected():
            self.local_settings.setFilehistory_Flag(True)
        else:
            self.local_settings.setFilehistory_Flag(False)

        if self.Jumplist_CB.isSelected():
            self.local_settings.setJumplist_Flag(True)
        else:
            self.local_settings.setJumplist_Flag(False)

        if self.Prefetch_CB.isSelected():
            self.local_settings.setPrefetch_Flag(True)
        else:
            self.local_settings.setPrefetch_Flag(False)

        if self.SAM_CB.isSelected():
            self.local_settings.setSAM_Flag(True)
        else:
            self.local_settings.setSAM_Flag(False)

        if self.Shellbags_CB.isSelected():
            self.local_settings.setShellbags_Flag(True)
        else:
            self.local_settings.setShellbags_Flag(False)
```

```
        if self.Shimcache_CB.isSelected():
            self.local_settings.setShimcache_Flag(True)
        else:
            self.local_settings.setShimcache_Flag(False)

        if self.Usnj_CB.isSelected():
            self.local_settings.setUsnj_Flag(True)
        else:
            self.local_settings.setUsnj_Flag(False)

        if self.Webcache_CB.isSelected():
            self.local_settings.setWebcache_Flag(True)
        else:
            self.local_settings.setWebcache_Flag(False)

    def initComponents(self):
        self.panel0 = JPanel()

        self.gbPanel0 = GridBagLayout()
        self.gbcPanel0 = GridBagConstraints()
        self.panel0.setLayout( self.gbPanel0 )

        self.Recentlyused_CB = JCheckBox( "CCM Recently Used Apps", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 5
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Recentlyused_CB, self.gbcPanel0 )
        self.panel0.add( self.Recentlyused_CB )

        self.Jumplist_CB = JCheckBox( "Parse Jumplist AD", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 7
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Jumplist_CB, self.gbcPanel0 )
        self.panel0.add( self.Jumplist_CB )

        self.Filehistory_CB = JCheckBox( "File History", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 9
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Filehistory_CB, self.gbcPanel0 )
        self.panel0.add( self.Filehistory_CB )

        self.Prefetch_CB = JCheckBox( "Parse Prefetch", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 11
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Prefetch_CB, self.gbcPanel0 )
        self.panel0.add( self.Prefetch_CB )

        self.SAM_CB = JCheckBox( "Parse SAM", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 13
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
```

```
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.SAM_CB, self.gbcPanel0 )
        self.panel0.add( self.SAM_CB )

        self.Shellbags_CB = JCheckBox( "Parse Shellbags", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 15
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Shellbags_CB, self.gbcPanel0 )
        self.panel0.add( self.Shellbags_CB )

        self.Shimcache_CB = JCheckBox( "Parse Shimcache", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 17
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Shimcache_CB, self.gbcPanel0 )
        self.panel0.add( self.Shimcache_CB )

        self.Usnj_CB = JCheckBox( "Parse USN Journal", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 19
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Usnj_CB, self.gbcPanel0 )
        self.panel0.add( self.Usnj_CB )

        self.Webcache_CB = JCheckBox( "Parse Webcache", actionPerformed=self.checkBoxEvent)
        self.gbcPanel0.gridx = 2
        self.gbcPanel0.gridy = 21
        self.gbcPanel0.gridwidth = 1
        self.gbcPanel0.gridheight = 1
        self.gbcPanel0.fill = GridBagConstraints.BOTH
        self.gbcPanel0.weightx = 1
        self.gbcPanel0.weighty = 0
        self.gbcPanel0.anchor = GridBagConstraints.NORTH
        self.gbPanel0.setConstraints( self.Webcache_CB, self.gbcPanel0 )
        self.panel0.add( self.Webcache_CB )

        self.add(self.panel0)

    def customizeComponents(self):
        self.Jumplist_CB.setSelected(self.local_settings.getJumplist_Flag())
        self.Filehistory_CB.setSelected(self.local_settings.getFilehistory_Flag())
        self.Prefetch_CB.setSelected(self.local_settings.getPrefetch_Flag())
        self.SAM_CB.setSelected(self.local_settings.getSAM_Flag())
        self.Shellbags_CB.setSelected(self.local_settings.getShellbags_Flag())
        self.Shimcache_CB.setSelected(self.local_settings.getShimcache_Flag())
        self.Usnj_CB.setSelected(self.local_settings.getUsnj_Flag())
        self.Webcache_CB.setSelected(self.local_settings.getWebcache_Flag())
        self.Recentlyused_CB.setSelected(self.local_settings.getRecentlyused_Flag())

    # Return the settings used
    def getSettings(self):
        return self.local_settings
```

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá