Symantec™ Data Loss Prevention Detection Customization Guide

Version 11.0

Symantec Data Loss Prevention Detection Customization Guide

Documentation version: 11

Legal Notice

Symantec Corporation
350 Ellis Street
Mountain View, CA 94043

http://www.symantec.com

Technical Support

Symantec Technical Support maintains support centers globally. Technical Support's primary role is to respond to specific queries about product features and functionality. The Technical Support group also creates content for our online Knowledge Base. The Technical Support group works collaboratively with the other functional areas within Symantec to answer your questions in a timely fashion. For example, the Technical Support group works with Product Engineering and Symantec Security Response to provide alerting services and virus definition updates.

Symantec's support offerings include the following:

- A range of support options that give you the flexibility to select the right amount of service for any size organization
- Telephone and/or web-based support that provides rapid response and up-to-the-minute information
- Upgrade assurance that delivers automatic software upgrades protection
- Global support purchased on a regional business hours or 24 hours a day, 7 days a week basis
- Premium service offerings that include Account Management Services

For information about Symantec's support offerings, you can visit our web site at the following URL:

www.symantec.com/business/support/

All support services will be delivered in accordance with your support agreement and the then-current enterprise technical support policy.

Contacting Technical Support

Customers with a current support agreement may access Technical Support information at the following URL:

www.symantec.com/business/support/

Before contacting Technical Support, make sure you have satisfied the system requirements that are listed in your product documentation. Also, you should be at the computer on which the problem occurred, in case it is necessary to replicate the problem.

When you contact Technical Support, please have the following information available:

- Product release level

- Hardware information
- Available memory, disk space, and NIC information
- Operating system
- Version and patch level
- Network topology
- Router, gateway, and IP address information
- Problem description:
  - Error messages and log files
  - Troubleshooting that was performed before contacting Symantec
  - Recent software configuration changes and network changes

Licensing and registration

If your Symantec product requires registration or a license key, access our technical support web page at the following URL:

www.symantec.com/business/support/

Customer service

Customer service information is available at the following URL:

www.symantec.com/business/support/

Customer Service is available to assist with non-technical questions, such as the following types of issues:

- Questions regarding product licensing or serialization
- Product registration updates, such as address or name changes
- General product information (features, language availability, local dealers)
- Latest information about product updates and upgrades
- Information about upgrade assurance and support contracts
- Information about the Symantec Buying Programs
- Advice about Symantec's technical support options
- Nontechnical presales questions
- Issues that are related to CD-ROMs or manuals

Support agreement resources

If you want to contact Symantec regarding an existing support agreement, please contact the support agreement administration team for your region as follows:

Asia-Pacific and Japan	customercare_apac@symantec.com
Europe, Middle-East, and Africa	semea@symantec.com
North America and Latin America	supportsolutions@symantec.com

Additional enterprise services

Symantec offers a comprehensive set of services that allow you to maximize your investment in Symantec products and to develop your knowledge, expertise, and global insight, which enable you to manage your business risks proactively.

Enterprise services that are available include the following:

Managed Services	These services remove the burden of managing and monitoring security devices and events, ensuring rapid response to real threats.
Consulting Services	Symantec Consulting Services provide on-site technical expertise from Symantec and its trusted partners. Symantec Consulting Services offer a variety of prepackaged and customizable options that include assessment, design, implementation, monitoring, and management capabilities. Each is focused on establishing and maintaining the integrity and availability of your IT resources.
Education Services	Education Services provide a full array of technical training, security education, security certification, and awareness communication programs.

To access more information about enterprise services, please visit our web site at the following URL:

www.symantec.com/business/services/

Select your country or language from the site index.

Contents

Chapter 1

Introducing the DLP Scripting Language

This chapter includes the following topics:

About the scripting language

Symantec Data Loss Prevention provides a basic scripting language that you can use to detect custom file types and to validate custom data identifiers. You can deploy custom scripts for both server and endpoint agent detection. You can extend the language using pre-built functions.

Table 1-1 Detection features supporting scripting

Feature	Description
Custom File Type Signature detection	The DLP Scripting Language lets you write a script that detects the unique bytes of a custom file type. See "Detecting custom file types" on page 31.
Custom Script validators for custom data identifiers	The DLP Scripting Language lets you write a script to validate patterns in a message.

About the scripting language syntax

The Symantec Data Loss Prevention Scripting Language is similar in syntax to the Perl programming language but simplified for ease of use. The Symantec Data Loss Prevention Scripting Language uses statements to perform operations on constant or variable values to check or manipulate data.

The Symantec Data Loss Prevention Scripting Language provides various system variables that you can use to access stored data.

See "System variables" on page 11.

The Symantec Data Loss Prevention Scripting Language provides three types of statements that you can use to check or manipulate data:

- Assert
  See "Assert statement" on page 11.
- If/Else
  See "If/Else statements" on page 12.
- Evaluate
  See "Evaluate statement" on page 13.

The Symantec Data Loss Prevention Scripting Language has the following syntactical characteristics:

- Statements must end in a semicolon.
- Multiple statements can be included on a single line of code.
- Each statement must stand alone; nested statements are not supported.

Note: Symantec Data Loss Prevention may update statement names and may add additional statement functions

System variables

System variables store data that you can check and manipulate. For custom file type detection, the script has access to the entire file by the `$data` variable. For custom validators, the script has access to the raw message, the normalized message, and the 10 bytes preceding and trailing the matched data. For custom validators the script does not have access to the entire message.

Warning: Do not assign values to system variables. These variables already hold system-defined data. Use a local variable such as `$result` to assign values. You should not use system variables with logical, assignment, or arithmetic operations.

Table 1-2 System variables

System variable	Description
`$data`	The script engine creates the byte array $data variable when it reads in a file. The $data variable stores the entire file. In previous releases of the scripting language, this variable was limited to 4 KB. In the version 11 release of the scripting language the entire file is stored in the $data variable.
`$match`	The script engine stores the data matched by the pattern in the `$match` variable before it is normalized.
`$normalizedMatch`	The script engine stores the normalized matched data in the `$normalizedMatch` variable after it is normalized.
`$matchPrefix`	You can use this method to verify if a message starts with a certain pattern. The methods checks 10 bytes before the matched pattern.
`$matchSuffix`	You can use this method to verify if a message ends with a certain pattern. The methods checks 10 bytes after the matched pattern.

Assert statement

The Assert statement evaluates a Boolean expression and asserts the value "true" when the expression returns a match. The Assert statement must end with a semicolon.

The Assert statement supports all regular Boolean expressions:

- == evaluates to

- >= greater than or equal to
- <= less than or equal to
- > greater than
- < less than
- != does not evaluate to

Table 1-3 Assert statement

Statement	Use	Example
`assertTrue`	The assertTrue statement checks if the value of the $variable evaluates to the specified value (in this example, 3). If it does the Boolean expression asserts the value of true.	`assertTrue($variable == 3);`

If/Else statements

You use the If/Else conditional statement to control the flow of program execution. The If/Else statement lets you include conditional logic in your script when you need to evaluate the unique bytes of a complex dataset.

The If/Else condition operates the same as conditional statements that other programming languages provide. The If/Else statement takes a Boolean expression, evaluates it, and alters the execution of the program based on the result of the expression.

The following example shows one way to use the If/Else conditional statement:

```
if ($var1 == 3)
{
  // statement
  // statement
}
else
  // statement
```

The scripting language supports nested execution of the statements that are contained within the conditional statement. To use nested statements, you use brackets within the scope of an If/Else statement to offset the multiple script statements.

If the dataset you want to evaluate requires more advanced conditional logic, you can declare multiple If/Else statements nested within each other.

Evaluate statement

The Evaluate statement provides a number of functions that you can you use to evaluate data. Not all functions are available for each feature.

Table 1-4 Evaluate statement functions per feature

Evaluate statement function	Custom File Type	Custom Script Validator
Addition	X	X
AsciiValue	X	NO
DataLength	X	X
GetAsciiStringAt*	X	X
GetBinaryIntValue*	X	NO
GetBinaryValueAt	X	NO
GetHexStringValueAt	X	NO
GetIntegerAt*	X	X
GetStringValueAt*	NO	X
Modulus	X	X
Multiply	X	X
Subtract	X	X

Key:

- X = Feature supports the statement on server and endpoint.
- NO = Statement not supported by that feature.
- * = New function in Symantec Data Loss Prevention version 11.

Evaluate statement functions

You use Evaluate statements to execute functions on variable or constant data values. You can save the return value of an Evaluate function as a variable or discard the return value. Evaluate statements must end with a semicolon.

Table 1-5 Evaluate statement functions

Function	Description	Example

Addition
`add`
	The Addition function takes two values as arguments and adds them together. The values can be variables or constants. You can save the returned result as a variable or discard the result.	Add two variables together and returns the value in the variable $result.
		`$result = add($var1, $var4);`
		Add two constants together but discards the value.
		`add(1, 2);`

AsciiValue `ascii`	The AsciiValue function takes a single ASCII string as a parameter and assigns it to the specified variable. The length of the ASCII parameter must be from one to four characters. You can use this statement for readability purposes.	`$result = ascii('CFV');` The $result variable is assigned the specified ASCII value.
DataLength `datalength`	The DataLength function counts the length of the variable array. The function takes the variable name of a byte array as a parameter and returns the number of bytes in that array.	`$result = datalength($data);` The engine creates the byte array $data variable when it reads in a file. The $data variable stores up to the first 4 KB of the file.
GetAsciiStringAt `getAsciiStringAt`	The GetAsciiStringAt function treats the data as ASCII characters and converts the data into a string. The data is converted starting from the specified offset for the specified number of digits.	The variable $data is a byte array with the values: 'abcdef'.getBytes(); The result should be abc. `$result = getAsciiStringAt($data, 0x0, 3);`

Table 1-5 Evaluate statement functions *(continued)*

Function	Description	Example
GetBinaryIntValue `getBinaryIntValueAt`	The GetHexStringValue function pulls the byte data as an integer from the specified index of a byte array variable. It also allows a user to specify how many digits to pull from the data. Since the return value is an integer, the number of digits has to be 1 – 4 bytes. This can be used to analyze data at specific offsets of a byte array. The number of digits are combined to form an integer value.	`$result = getBinaryIntValueAt($data, 0x0, 1);` The $data variable is byte array with values {(byte)0x59,(byte) 0xAD,(byte) 0x1C,(byte) 0xDF,(byte) 0x2B,(byte)0x37}. In this example the $result should equal 89.
		`$result = getBinaryIntValueAt($data, 1);` The $data variable is a byte array with values {1, 2, 3}. The $result should equal 2.
GetBinaryValuteAt `getBinaryValueAt`	The GetBinaryValuteAt function pulls the byte data into a new byte array based on the offset and length specified. The new byte array can then be compared to other byte arrays for equality. This function lets you specify how many digits to retrieve from the data (from 1 - 4 bytes). You use this function to analyze data at specific offsets of a byte array.	The variable $data is byte array with values {(byte)0x59,(byte) 0xAD,(byte) 0x1C,(byte) 0xDF,(byte) 0x2B,(byte)0x37}. The $result should be a byte array with the byte 0x59. `$result = getBinaryValueAt($data, 0x0, 1);`
		The $data variable is a byte array with values {1, 2, 3}. The $result should equal a new byte array with the number 2 in it. `$result = getBinaryValueAt($data, 1);`
GetHexStringValue `getHexStringValue`	The GetHexStringValue function takes a hexadecimal string as a parameter and converts it to the byte (binary) representation.	`$result = getHexStringValue('D0CF');`

Table 1-5 Evaluate statement functions *(continued)*

Function	Description	Example
GetIntegerAt `getIntegerAt`	The GetIntegerAt function is similar to GetAsciiStringAt, except GetIntegerAt parses the ASCII characters and converts them to an integer value. The data is converted starting from the specified offset for the specified number of digits. If a character is not a numerical value, the script throws an exception.	The $data variable is a byte array with the values: '12345'.getBytes(); The result should be 34. `$result = getIntegerAt($data, 0x2, 2);`
GetStringValueAt `getStringValueAt`	The GetStringValueAt function pulls the data into a new character array based on the offset and length specified. The new byte array can then be compared to other byte arrays for equality.	The variable $data is a byte array with values {(byte)0x59,(byte) 0xAD,(byte) 0x1C,(byte) 0xDF,(byte) 0x2B,(byte)0x37}. `$result = getStringValueAt($data, 0x0, 1);` The result should be byte 0x59
		The variable $data is a byte array with values {1, 2, 3}. `$result = getStringValueAt($data, 1);` The result should equal a new byte array with the number 2 in it. (Offset starts at zero.)
Modulus `mod`	The Modulus (mod) function returns the mod of two parameters. The mod is the remainder of the first parameter divided by the second.	The result should be 2. `$result = mod(5, 3);`
Multiply `multiply`	The Multiply function takes two arguments and multiplies their values.	`$result = multiply(2, 4);` The result should be 8.

Table 1-5 Evaluate statement functions *(continued)*

Function	Description	Example
Subtract `sub`	The Subtract function subtracts the second parameter from the first.	`$result = sub(10, 4);` The result should be 6.

Example scripts for custom file type detection

Listed here are several example script solutions that detect custom file types. These examples can be used as reference for writing your own custom scripts and for detecting the indicated custom file type.

The following script example detects the Microsoft Word file type:

```
$Int1 = getHexStringValue('D0CF');
$Int2 = getBinaryValueAt($data, 0x0, 2);
assertTrue($Int1 == $Int2);
$Int3 = getHexStringValue('ECA5');
$Int4 = getBinaryValueAt($data, 0x200, 2);
assertTrue($Int3 == $Int4);
```

The following script example detects the CDD file type:

```
$Int1 = getBinaryValueAt($data, 0x0, 4);
$Int2 = getBinaryValueAt($data, 0x8, 4);
assertTrue($Int1 == $Int2);
$Int3 = getBinaryValueAt($data, 0x0, 2);
$Int4 = getBinaryValueAt($data, 0x2, 2);
assertTrue($Int3 != $Int4);
$Last = getBinaryValueAt($data, 0x27, 1);
$RecSep = getHexStringValue('1e');
assertTrue($Last == $RecSep);
```

The following script example detects the CATIA file type:

```
$Int1 = ascii('V');
$Int2 = getBinaryValueAt($data, 0x0, 1);
assertTrue($Int1 == $Int2);
$Int3 = ascii('CFV');
$Int4 = getBinaryValueAt($data, 0x3, 3);
assertTrue($Int3 == $Int4);
```

The following script example detects the EPUB file type.

```
$slash=getHexStringValue('2f');
$epub1=ascii('epub');
$epub2=ascii('zip');
$slash1=getBinaryValueAt($data, 0xb, 1);
assertTrue($slash == $slash1);
$word1=getBinaryValueAt($data, 0xc, 4);
assertTrue($word1 == $epub1);
$word2=getBinaryValueAt($data, 0x11, 3);
assertTrue($word2 == $epub2);
```

Note: EPUB files are in the open book format (XML) encapsulated in a ZIP file format. You cannot test this script using the File Type Analyzer utility because the script detects the "application/epub+zip" string contained in the manifest file (named "mimetype"). The utility cannot crack the ZIP to read the manifest. However, the detection engine can crack the ZIP file and read the manifest. You can implement this script in an instance of the Custom File Type Signature detection rule and detect EPUB files.

The following script example detects the Amazon Kindle file type:

```
$book=ascii('BOOK');
$mobi=ascii('MOBI');
$word1=getBinaryValueAt($data, 0x3c, 4);
$word2=getBinaryValueAt($data, 0x40, 4);
assertTrue($book == $word1);
assertTrue($mobi == $word2);
$null=getBinaryValueAt($data, 0x3b, 1);
assertTrue($null == 0);
$nullx=getBinaryValueAt($data, 0x44, 1);
assertTrue($nullx == 0);
```

The following script example detects the Oracle IRM file type, which is used for Digital Rights Management (DRM):

```
$soft=ascii('Soft');
$seal=ascii('SEAL');
$word1=getBinaryValueAt($data, 0x0, 4);
$word2=getBinaryValueAt($data, 0x4, 4);
assertTrue($soft == $word1);
assertTrue($seal == $word2);
```

In addition, the following two tutorials offer additional examples of the scripting language:

- Java class files
  See "Tutorial 1: Detecting Java class files" on page 32.
- Password encrypted ZIP files
  See "Tutorial 2: Detecting an encrypted ZIP file format" on page 35.

Example scripts for custom validators

Listed here are some examples of custom script validators.

The following script is a basic custom validator that validates a 5-digit data identifier by retrieving the 5th digit and the first 4 digits, assigning them to variables, and comparing these values against the expected datalength.

Pattern

```
\d{5}
```

Normalizer

Do Nothing

Custom Script

```
$s1 = getStringValueAt($normalizedMatch, 0x0, 5);
$s2 = getStringValueAt($normalizedMatch, 0x5, 4);
$size1 = datalength($s1);
$size2 = datalength($s2);
assertTrue($size1 == 5);
assertFalse($size2 != 4);
```

The following custom script validates a 10-character string in the form of LL/MM/DD/YYYY. The first two characters are the initials of the person and are excluded from validation. The remaining digits are saved into separate variables, computed by a multiplier, added, and then compared to ensure that they conform to a proper day (less than 32), month (less than 13), and year (less than 2051).

Pattern

```
\l{2}\d{8}
```

Normalizer

Digits and Letters

Custom Script

```
$m1 = getIntegerAt($normalizedMatch, 0x2, 1);
$m2 = getIntegerAt($normalizedMatch, 0x3, 1);
$d1 = getIntegerAt($normalizedMatch, 0x4, 1);
$d2 = getIntegerAt($normalizedMatch, 0x5, 1);
$y1 = getIntegerAt($normalizedMatch, 0x6, 1);
$y2 = getIntegerAt($normalizedMatch, 0x7, 1);
$y3 = getIntegerAt($normalizedMatch, 0x8, 1);
$y4 = getIntegerAt($normalizedMatch, 0x9, 1);

$m1 = multiply($m1, 10);
$d1 = multiply($d1, 10);
$y1 = multiply($y1, 1000);
$y2 = multiply($y2, 100);
$y3 = multiply($y3, 10);


$Day = Add($d1, $d2);
$Month = Add($m1, $m2);
$Year = Add($y1, $y2, $y3, $y4);

assertTrue($Day > 0);
assertTrue($Day <= 31);
assertTrue($Month > 0);
assertTrue($Month <= 12);
assertTrue($Year >= 1910);
assertTrue($Year <= 2050);
```

The following custom script validator can be used to verify the match of a Turkish ID number. A Turkish ID is an 11-digit number. The first digit cannot be zero. The 10th and 11th digits are check digits for error detection.

Pattern `\d{11}`

Normalizer Digits Only

Custom Script

```
$k1 = getIntegerAt($normalizedMatch, 0x0, 1);
$k2 = getIntegerAt($normalizedMatch, 0x1, 1);
$k3 = getIntegerAt($normalizedMatch, 0x2, 1);
$k4 = getIntegerAt($normalizedMatch, 0x3, 1);
$k5 = getIntegerAt($normalizedMatch, 0x4, 1);
$k6 = getIntegerAt($normalizedMatch, 0x5, 1);
$k7 = getIntegerAt($normalizedMatch, 0x6, 1);
$k8 = getIntegerAt($normalizedMatch, 0x7, 1);
$k9 = getIntegerAt($normalizedMatch, 0x8, 1);
$c1 = getIntegerAt($normalizedMatch, 0x9, 1);
$c2 = getIntegerAt($normalizedMatch, 0xA, 1);

$iOdds = add($k1, $k3, $k5, $k7, $k9);
$iEvens = add($k2, $k4, $k6, $k8);
$iOddsMltSeven = multiply($iOdds, 7);
$iEvensMltNine = multiply($iEvens, 9);
$iOddsMltEight = multiply($iOdds, 8);
$iMidSum = add($iOddsMltSeven, $iEvensMltNine);
$iCheck1 = mod($iMidSum, 10);
assertTrue($iCheck1 == $c1);
$iCheck2 = mod($iOddsMltEight, 10);
assertTrue($iCheck2 == $c2);
```

Chapter 2

Using the File Type Analyzer utility for custom file type detection

This chapter includes the following topics:

About the File Type Analyzer utility

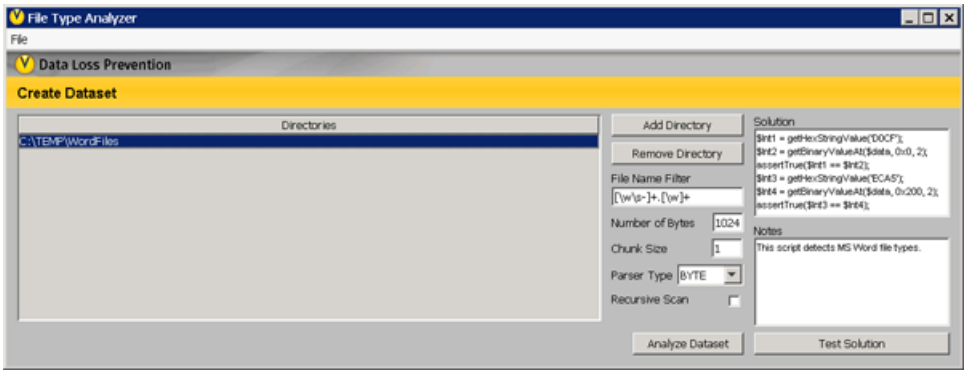
To assist you with analyzing custom file types, you can use the Symantec Data Loss Prevention File Type Analyzer utility.

The File Type Analyzer utility helps you find the commonalities and attributes that describe a custom file type. This data is often referred to as "magic bytes" because they are unique characters that positively identify the file type.

The File Type Analyzer utility is a standalone Java application that features a graphical user interface. This utility enables you to perform the following operations:

- Read in a collection of files from a directory or directories (the "dataset").
- View and compare the unique bytes for each file in the dataset.
- Analyze the bytes across files and determine those that are consistent.
- Test a custom file type detection script.
- Save and open dataset configurations and test scripts.

Figure 2-1 Symantec Data Loss Prevention File Type Analyzer utility interface



See "Installing the File Type Analyzer utility" on page 24.

Installing the File Type Analyzer utility

The Symantec Data Loss Prevention File Type Analyzer utility is available for Windows. The software is available for download from https://fileconnect.symantec.com.

To install the File Type Analyzer utility on Windows

1. Double-click the `fileanalyzer_windows_4_0_1.exe` executable.
2. At the "Welcome" screen, click **Next**.
3. Accept the default Destination Directory C:\Program Files\File Analyzer.

   Or, you can change the Destination Directory to one you prefer.
4. Click **Next** to install the utility.
5. Click **Finish** to complete the installation process.

See "About the File Type Analyzer utility" on page 23.

Launching the File Type Analyzer utility

You can run the File Type Analyzer utility in GUI mode or from the command line. The GUI mode of operation is recommended because it lets you test your script against a configured dataset.

Launching the File Type Analyzer utility (GUI version)

1. Obtain the File Type Analyzer utility and install it.

   See "Installing the File Type Analyzer utility" on page 24.

2. Navigate to the installation folder where you installed the utility.

   For example: `C:\Program Files\File Analyzer`

3. Double click the `analyzer_gui.exe` executable.

   The File Type Analyzer utility interface should appear.

Creating the dataset

The File Type Analyzer utility offers several parameters for configuring the dataset in preparation for analyzing file type byte data.

Table 2-1 Parameters for configuring the dataset

Parameter	Use
Add Directory	This option lets you choose which directories to include in the file analysis. You can add multiple directories to a single dataset. Each directory you select should contain samples of the file type you want to analyze and ultimately detect. To have a useful dataset, include several samples of the file type, including different versions of the product with different features enabled and disabled. **Note:** To achieve the best results, the recommended minimum sample size is 15 files of the same file type.
Remove Directory	This option lets you remove a directory you have added to the dataset. You can select multiple directories to remove. When a directory is removed, it is no longer scanned as part of the dataset.

Table 2-1 Parameters for configuring the dataset *(continued)*

Parameter	Use

File Name Filter	This field contains a regular expression pattern that tells the utility what files from each directory to include in the dataset. A regular expression is used because it provides flexibility for filtering the files you want to include in your dataset.
	
	The following regular expression reads in all ASCII file names from a directory (or directories) to a dataset:
	
	`[\w\s]+.[\w]+`
	
	The following regular expression lets you filter file names that use non-ASCII characters:
	
	`[^0x00]+.[\w]+`
	
	Note: For assistance with using regular expressions for file name filtering, see the topic "About writing regular expressions" in the *Symantec Data Loss Prevention Administration Guide* or in the online Help.

Number of Bytes	This field specifies the number of bytes per file to display for analysis. The default maximum value for this field is **1024** bytes. See "Increasing the number of bytes that are analyzed" on page 30.
Chunk Size	This field represents the size of the group of bytes to be displayed in a column. For example, if you enter **2** in this field, the utility displays two bytes of data in each column (offset).
Parser Type	This option defines how the data is displayed for analysis from the scanned dataset. ■ The **BYTE** option displays the analysis results in hexadecimal format representing the corresponding byte value. ■ The **ASCII** option displays the analysis results as ASCII characters. ■ The **NUMBER** option displays the analysis results in integer format.
Recursive Scan	If this box is checked, the utility scans each directory and any subdirectories that are included in the dataset. If a directory contains subdirectories where files you want to scan are located, choose this option. **Note:** Recursive scanning is memory intensive. If you want to analyze either a large or a recursive dataset, consider increasing the Java heap size to improve performance. See "Increasing the Java heap size for large or recursive datasets" on page 29.

Table 2-1 Parameters for configuring the dataset *(continued)*

Parameter	Use
Analyze Dataset	Click this option when you have completed configuring the dataset. The File Type Analyzer utility validates the input and initiates the file analysis process. The utility reads in all the necessary data and displays the results in the "Analyze Dataset" screen.

Analyzing dataset results

The Analyze Dataset screen displays the results of the dataset based on the criteria you specified.

Once the utility filters the files in your dataset, it sorts and displays the data by tabs according to file extension. You can further sort the data by clicking the column names. You can also delete columns or rows of irrelevant data by selecting the row or column and performing a right-click.

Note: The File Type Analyzer utility uses the file extension to organize by tab the files in your dataset. However, the file extension is not a reliable means of detecting a file type because the file extension can easily be changed. Symantec Data Loss Prevention detects file type based on uniquely-identifying specific bytes.

When you analyze dataset results, your goal is to locate the unique bytes that are consistent for each instance of the file type. These unique bytes are the "magic bytes" for the analyzed file type. You must determine what the magic bytes are to write a script that detects the custom file type. For example, the first two bytes of a Microsoft Word file (*.doc) are DO CF (in hexadecimal format).

To help you assess the results and find the magic bytes for the custom file type, click the **Analyze Table Data** option. With the default option COLUMN_MATCH selected, the File Type Analyzer utility highlights the columns that are the same across all files in the selected tab.

The ROW_OFFSET_MATCH option looks for byte matches within the same file (row). The offsets (columns) that match in the same row are highlighted; those that match the same offset in another row are not. This option is useful for a few file types that use unique bytes within the same file to indicate file type. For example, the CADAM file type (*.cdd) uses the same values for bytes 0 – 3 and bytes 8 – 11 within each file, but these values are different across files.

Once you have analyzed the results and determined the magic bytes, the next step is to write a script to detect the file type.

See "About the scripting language syntax" on page 10.

Refer to the tutorials for instructions on creating the dataset, analyzing the results, and writing a script to detect a custom file type. These tutorials demonstrate how the File Type Analyzer utility works and should help you get started scripting solutions to detect custom file types.

See "Tutorial 1: Detecting Java class files" on page 32.

See "Tutorial 2: Detecting an encrypted ZIP file format" on page 35.

Testing the script solution

The File Type Analyzer utility provides fields for entering your custom script solution, annotating it, and testing the solution against the dataset.

Table 2-2 Parameters for testing the script solution against the dataset

Parameter	Use
Solution	This field is where you enter the script text you want to use to detect the custom file type. See "About the scripting language syntax" on page 10.
Notes	This field provides a mechanism for annotating the dataset you have configured and your script solution. See Figure 2-1 on page 24. This field is useful for saving your dataset configurations and script solutions. See "Saving, opening, editing a dataset" on page 29.
Test Solution	Click this option to verify that your script accurately detects the custom file type.

When you test your solution, the utility takes the data from the dataset table and filters the files based on the dataset criteria. Once the dataset is built, the script engine runs the solution against the dataset and displays the results in the "Test Dataset Results" screen. The displayed results give you an indication of how well your script has worked to detect the custom file type.

The "Test Dataset Results" screen displays the results of the test in two tabbed panes:

- **Matched Files** – The top pane lists all the files in the configured dataset that your script detected.

- **Mismatched Files** – The bottom pane displays all the files in the configured dataset that your script did not detect.

This bifurcated display lets you quickly assess the accuracy of your script. You can easily see files matched that should not (false positives). You can also see the files that failed to match but should have (false negatives). Finally, you can see if there is any discrepancy between a file extension and the actual file type based on its unique bytes.

Saving, opening, editing a dataset

The File Type Analyzer utility lets you save configured datasets for subsequent reuse. You can open a saved dataset configuration and reanalyze the data at anytime. You can also edit a configured dataset, change its configuration parameters, and update your script solution.

Table 2-3 Options for saving, opening, editing a dataset

Parameter	Use
Save	You can perform a File > Save action to save your dataset configuration and script solution. The file is saved as a *.fgi file type.
Open	You can perform a File > Open action to open a saved dataset. Browse to the *.fgi file and open it.
Edit Dataset	Use this option to change the configuration parameters of an active dataset. You can add directories to or remove directories from the dataset, change configuration parameters, or update the script solution.

Increasing the Java heap size for large or recursive datasets

If you analyze a large or a recursive dataset, you may have to wait to analyze or test the files in the dataset. The File Type Analyzer utility needs to scan each directory in the dataset and perform I/O operations on each file that meets the dataset criteria.

If the utility runs out of memory before it processes the files, it freezes and does not move on to the expected screen.

If you analyze a large dataset (100,000+ files) or use recursive scanning to create the dataset, increase the maximum Java heap size.

To increase the Java heap size for the File Type Analyzer utility (GUI version)

1. Open a command line interface (Windows) or a console interface (Linux).
2. Launch the File Type Analyzer utility from the command line using the following command:

   ```
   analyzer_gui.exe -Xmx1024m
   ```

3. The interface should launch with the Java heap size increased accordingly.

   You should now be able to analyze or test a large or a recursive dataset without error or significant delay.

Increasing the number of bytes that are analyzed

The maximum value for the number of bytes that the Symantec Data Loss Prevention File Analyzer utility allows for a dataset is 1024. Generally this value is sufficient to analyze custom file types since the signature bytes typically exist at the beginning of the file.

To analyze more than the first 1024 bytes of data, modify the File Analyzer utility as follows.

Increasing the number of bytes that are analyzed

1. Using WinRAR, open the file `C:\Program Files\File Analyzer\lib\filegenie.jar`.
2. Within the `filegenie.jar` file, open the file `create-dataset-form-context.xml`.
3. In the first bean, locate the property element `maxNumByteSize`.
4. Change the value from "**1024**" to the desired number of bytes.
5. Save the XML file and update the JAR.
6. Run the File Analyzer and verify that the additional bytes are read and displayed in the user interface.

Chapter 3

Tutorials

This chapter includes the following topics:

Detecting custom file types

Symantec Data Loss Prevention detects more than 300 file types. However, if the type of file you want to detect is not supported, you can detect it using a custom script. To do this, you use the Symantec Data Loss Prevention Scripting Language to write a script that detects the binary signature of the particular file format you want to detect.

In addition, you can use the design-time Symantec Data Loss Prevention File Type Analyzer utility to help you determine the unique bytes of the custom file type you want to detect.

To detect custom file types

1. Create a sample archive or directory containing several instances of the custom file or document type you want to detect.

   Create different samples of the document, with different features turned on and off, and based on different software versions.

2. Use the Symantec Data Loss Prevention File Type Analyzer utility to read in the bytes of the dataset.

   Look for patterns among the file bytes to determine file type recognition characters (also known as "magic bytes"). Refine the sample and run more scans as necessary.

   See “About the File Type Analyzer utility” on page 23.

3 Use the Symantec Data Loss Prevention Scripting Language to write a script that detects the custom file type. Use the File Type Analyzer utility to test and refine your script.

See "Example scripts for custom file type detection" on page 17.

See "Testing the script solution" on page 28.

4 Enable the Custom File Type Signature detection rule so it appears in the Enforce Server policy builder interface.

See the topic "Enabling custom file type detection" in the online Help and the *Symantec Data Loss Prevention Adminstration Guide*.

5 Deploy an instance of the Custom File Type Signature condition in one or more detection rules or exceptions.

See the topic "Configuring the Custom File Type Signature condition" in the online Help and the *Symantec Data Loss Prevention Adminstration Guide*

6 Author a policy that uses the detection rule or exception. Test and refine the policy as necessary.

Tutorial 1: Detecting Java class files

This tutorial provides instructions for using the File Type Analyzer utility to analyze a dataset and determine the magic bytes. It also demonstrates how to use the scripting language to author and test a solution.

In this first tutorial you analyze and detect Java class files. This tutorial assumes that you use the Windows-based GUI version of the File Type Analyzer utility.

Tutorial 1: Detecting Java class files

1 Install the File Type Analyzer utility.

See "Installing the File Type Analyzer utility" on page 24.

2 Launch the File Type Analyzer utility.

See "Launching the File Type Analyzer utility" on page 25.

3 Prepare the dataset for this example.

Copy several (15 or more) Java class files (*.class) to a directory on your file system. (For the purposes of this tutorial, the directory that is used is C:\temp\JavaClassFiles.)

In addition, to ensure that your script matches only Java class files, add a few non-Java class files to the same directory.

4 Add the dataset directory to the File Type Analyzer utility.

In the File Type Analyzer utility, click **Add Directory**. Browse to and select the directory where you copied the files and click **Open**.

5 In the **File Name Filter** field enter a regular expression to filter the files.

For example, the following regular expression screens all files in the selected directory: `[\w\s]+.[\w]+`

This regular expression matches:

- **(\w)** Any alphanumeric character, digit, or underscore
- **(\s)** Any whitespace
- **(+)** One or more of the previous characters must match
- **(.)** Any single character, including itself

You may need to adjust this expression to find the files you want to analyze in the specified directory. For example, if a file name contains a dash (-), adjust the expression as follows: `[\w\s-]+.[\w]+`

See "Creating the dataset" on page 25.

6 In the **Number of Bytes** field, enter **1024**.

The magic bytes of a file are almost always contained within the first 1024 bytes of a file. If you want to analyze more than the first 1024 bytes of data, you must increase the number of bytes that the File Type Analyzer utility can read and display.

See "Increasing the number of bytes that are analyzed" on page 30.

7 For the **Chunk Size** enter **1**.

8 For the **Parser Type** choose **BYTE**.

9 If the files you want to screen are in nested directories, choose the **Recursive Scan** option.

Note: If you choose the Recursive Scan option, or you have a large dataset, increase the Java heap size allocated to the File Type Analyzer utility.

See "Increasing the Java heap size for large or recursive datasets" on page 29.

---

10 Click **Analyze Dataset**. The utility analyzes all files in the directory and displays the results. The utility organizes each file by tabs according to its extension. In the **All** tab the utility displays all screened files. In the **.class** tab the utility displays only the Java class files.

11 Click **Analyze Table Data** again. This time the utility highlights the bytes within each file that match across all files.

As you can see, for Java class files there are several bytes in common, including the first four (0 through 3): **CA FE BA BE**. These bytes are the magic bytes for Java class files.

In the drop-down menu at the bottom you can change how the utility analyzes table data. The default option is COLUMN_MATCH, which generally provides the most accurate matching. If you switch to this analysis mode you need to click **Analyze Table Data** again to see the matching bytes by row.

12 Now that you know what the magic bytes are for Java class files, you can author a script to detect this file type. You can then test your script using the File Type Analyzer utility.

In the **Solution** field, enter the following script to detect Java class files:

```
$Int1 = getHexStringValue('CAFE');
$Int2 = getBinaryValueAt($data, 0x0, 2);
assertTrue($Int1 == $Int2);
$Int3 = getHexStringValue('BABE');
$Int4 = getBinaryValueAt($data, 0x2, 2);
assertTrue($Int3 == $Int4);
```

13 Click **Test Solution**. At the top of the interface you see the **Matched Files**. Only those files containing the CAFE BABE magic bytes appear in the "Matched Files" section of the interface. Files that do not contain these magic bytes appear in the **Mismatched Files** section at the lower-half of the interface.

Analysis of the script solution:

- When you analyze the dataset, the File Type Analyzer utility indicates that the first two bytes of a Java class file are CA FE. So, in the first statement of the script you assign that value as a hexadecimal string to the variable **$Int1**.
- In the second statement of the script you get the firsts two bytes of each file and assign that value to the variable **$Int2**. The "0x0, 2" portion of the statement tells the script engine to start at the first byte and get the first two.
- In the third statement you compare the values of the two variables and check for a match.
- The process is repeated for the third and the fourth bytes ("0x2, 2"), looking for a match on BA BE. Files that match both evaluations are detectable by the script and appear in the "Matched Files" portion of the interface.

14 In the **Note** section enter a comment about the solution, such as "**Custom script for detecting Java class files**."

15 In the File Type Analyzer interface, select **File > Save**. Give the file a name and save it to a local directory, such as `C:\temp\JavaClassFiles.fgi`.

16 Close the File Type Analyzer interface and relaunch it. Choose **File > Open** then browse to and select the `JavaClassFiles.fgi` file.

   The dataset parameters and script solution appear in the interface. From here you can reanalyze the dataset and refine your solution as necessary. Click **Edit Dataset** to add or remove directories containing files you want to analyze. You can also right-click a row and remove an individual file from the dataset.

17 Once you have debugged your solution, deploy your script to an instance of the Custom File Type Signature rule. You can then author and deploy new policies that use this rule to detect the custom file type.

Tutorial 2: Detecting an encrypted ZIP file format

This tutorial demonstrates how to write a custom script to detect password protected (encrypted) ZIP files. While a Symantec Data Loss Prevention detection server can detect encrypted ZIP files, an endpoint agent cannot. The solution that is provided here lets you work around this issue.

Note: This script detects if a ZIP file is encrypted by checking if the encryption bit is enabled on the first file entry. Because ZIP allows encryption on a per file basis, this script only works if all files or the first file in the ZIP are encrypted.

This tutorial assumes that you completed the first tutorial.

To detect an encrypted ZIP file format

1 Create several (15 or more) password-protected ZIP files and put them in a directory such as `c:\temp\files\ZIP`.

2 Create a second set of ZIP files (5 or more) that are not encrypted so that you have both matched and mismatched results.

   Place these non-encrypted ZIP files in a second directory such as `c:\temp\files\ZIP2`.

3 Launch the File Type Analyzer utility (`analyzer_gui.exe`).

4 Add the `c:\temp\files\ZIP` directory as the dataset.

   Also add the `c:\temp\files\ZIP2` directory to the dataset.

5 Enter and select the required dataset parameters:

- **File Name Filter**: **[\w\s]+.[\w]+**
- **Number of Bytes**: **1024**
- **Chunk Size**: **1**
- **Parser Types**: BYTE

6 Click **Analyze Dataset**.

7 With COLUMN_MATCH selected, click **Analyze Table Data**.

The utility highlights the byte matches across all files. Note the exact matches for the first 6 bytes of all files. Note also that the 7th byte is zero for the ZIP files that are not encrypted. The 7th bit is the encryption bit.

8 In the **Solution** field, enter the following script:

```
$pktag=ascii('PK');
$frecord=getHexStringValue('0304');
$pkbytes=getBinaryValueAt($data, 0x0, 2);
assertTrue($pktag == $pkbytes);
$recordbytes=getBinaryValueAt($data, 0x2, 2);
assertTrue($frecord == $recordbytes);
$cryptByte=getBinaryValueAt($data, 0x6, 1);
$encrypted=mod($cryptByte, 2);
assertTrue($encrypted == 1);
```

9 The solution should match only those ZIP files in the dataset that are encrypted. The ZIP files that are not encrypted should appear in the "Mismatched Files" pane.

Analysis of this script solution:

- **$pktag=ascii('PK');**
  The first statement assigns the "$pktag" variable the value "PK." If you switch the Parser Type to **ASCII**, you see that the first two bytes of all ZIP files are "P" and "K".
- **$frecord=getHexStringValue('0304');**
  The second statement assigns the "$frecord" variable the value of "0304", which are the 3rd and 4th bytes of the ZIP files. (Switch back to BYTE for the Parser Type to confirm this value.)
- **$pkbytes=getBinaryValueAt($data, 0x0, 2);**
  The third statement gets the binary value of the first two bytes.
- **assertTrue($pktag == $pkbytes);**

The fourth statement compares the values of the "$pktag" and "$pkbytes" variables, looking for an exact match of "P" and "K". If the values match, the assertTrue value is achieved.

- **$recordbytes=getBinaryValueAt($data, 0x2, 2);**
  The fifth statement checks the binary value of the 3rd and 4th bytes (start at the 3rd byte and count 2). Here the values (in BYTE mode) are "03" and "04".
- **assertTrue($frecord == $recordbytes);**
  The sixth statement compares the values of the "$frecord" and the "$recordbytes" variables. If the returned value ("$recordbytes") matches the value assigned to the "$frecord" variable ("03" and "04"), the assertTrue value is achieved.
- **$cryptByte=getBinaryValueAt($data, 0x6, 1);**
  The seventh statement gets the binary value at the 7th byte (column 6).
- **$encrypted=mod($cryptByte, 2);**
  The eighth statement divides the value of the 7th byte (as assigned to the "$cryptByte" variable) by "2." It then assigns this remainder to the "$encrypted" variable.
- **assertTrue($encrypted == 1);**
  The ninth statement checks the value of the "$encrypted" variable. If the value is zero (no remainder), then the ZIP file is not encrypted. If there is a remainder then the ZIP file is encrypted.

Index