Symantec™ Data Loss Prevention Server FlexResponse Platform Developers Guide

Version 14.0

Symantec Data Loss Prevention Server FlexResponse Platform Developers Guide

Documentation version: 14.0a

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 software upgrades
- 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 website 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, DVDs, 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_apj@symantec.com
Europe, Middle-East, and Africa	semea@symantec.com
North America and Latin America	supportsolutions@symantec.com

Contents

Chapter 1

Introducing the Symantec Data Loss Prevention Server FlexResponse Plug-in

This chapter includes the following topics:

About the Server FlexResponse platform

The Server FlexResponse application programming interface (API) provides a flexible platform for incident remediation. It enables Symantec Data Loss Prevention users to protect data by automatically or manually invoking custom Server FlexResponse actions.

Symantec provides a set of Server FlexResponse plug-ins that perform various remediations such as quarantining sensitive data, copying files, and applying digital rights protection or encryption. Independent developers can also write Server FlexResponse plug-ins to perform custom incident remediation using this API and the Java programming language. The Server FlexResponse API enables developers to build a plug-in that can be used to implement incident responses for use in Automated and Smart Response rules.

The following are example Network Protect actions that you can implement by developing a Server FlexResponse plug-in:

- Change Access Control Lists (ACL) on files. For example, you can remove guest access to selected files.

- Apply Digital Rights Management (DRM). For example, you can apply digital rights to documents so external parties are restricted in their access to sensitive material. These digital rights can include “do not forward” or “do not print.”
- Encrypt files.
- Migrate files to SharePoint. The custom protect action can move files from shares to a SharePoint repository, and then apply DRM and ACLs.
- Perform workflow and automation of remediation responses.
- Use the Symantec Workflow business process automation workflow.

The following steps are involved in building, deploying, and using a Server FlexResponse plug-in:

- Developing a plug-in using the Java API. This stage involves designing and coding the plug-in and remediation action.
  See “Developing a Server FlexResponse plug-in” on page 13.
- Configuring plug-in parameters by creating the configuration properties file for your plug-in.
  See “Creating a properties file to configure a Server FlexResponse plug-in” on page 44.
- Adding your plug-ins to the plug-ins configuration properties file.
  See “Adding a Server FlexResponse plug-in to the plug-ins properties file” on page 42.
- Deploying your custom plug-in on the Enforce Server.
  See “Deploying a Server FlexResponse plug-in” on page 41.
- Loading the plug-in, including the plug-in metadata.
- Creating response rules for incident Smart Response actions.
- Using the plug-in action to remediate an incident.
  See “Using the action of a Server FlexResponse plug-in to remediate an incident manually” on page 50.
- Verifying the results of the Server FlexResponse plug-in action.
  See “Verifying the results of an incident response action” on page 51.

Note: Server FlexResponse plug-ins that were created for Symantec Data Loss Prevention versions 11 and 12 are compatible with Symantec Data Loss Prevention 14.

Components of the Server FlexResponse plug-in

The Server FlexResponse plug-in includes the following components:

- See "Server FlexResponse plug-in API library" on page 11.
- See "Server FlexResponse plug-in Java code examples" on page 11.
- See "Server FlexResponse plug-in Javadoc" on page 11.

The following default installation directory is for Windows:

```
c:\SymantecDLP
```

The following default installation directory is for Linux:

```
/opt/SymantecDLP
```

References to `SymantecDLP` in this documentation are to this location.

Server FlexResponse plug-in API library

The application programming interface library is provided with the Symantec Data Loss Prevention release.

See "About the Server FlexResponse interfaces" on page 55.

The library is in the file `flexresponseapi.jar` in the following directory:

```
SymantecDLP\Protect\tomcat\webapps
    \ProtectManager\WEB-INF\lib\
```

You must add the `flexresponseapi.jar` file to your classpath when you build your Server FlexResponse plug-in.

Server FlexResponse plug-in Java code examples

Code examples contain sample plug-ins for some remediation use cases.

See "About the Server FlexResponse plug-in examples" on page 53.

These are included in the Server FlexResponse development kit in the following directory:

```
SymantecDLP\Protect\tools\flexresponse\examples\
```

Server FlexResponse plug-in Javadoc

The Javadoc describes the Java classes for the Server FlexResponse plug-in.

The Javadoc is in the following directory:

```
SymantecDLP\Protect\tools\flexresponse\javadoc\
```

Open the file `index.html`, as an initial page to view the Javadoc.

Chapter 2

Developing a Symantec Data Loss Prevention Server FlexResponse plug-in

This chapter includes the following topics:

Developing a Server FlexResponse plug-in

Developing a Server FlexResponse plug-in requires knowledge of Java programming. An annotated sample plug-in provides some information about the structure of a Server FlexResponse plug-in.

See “Sample (annotated) Server FlexResponse plug-in “Hello World”” on page 31.

A sample procedure describes the steps for developing, configuring, deploying, and testing a Server FlexResponse plug-in.

See "Developing, deploying, configuring, and testing a sample Server FlexResponse plug-in" on page 36.

A separate properties file for each custom Server FlexResponse plug-in provides the plug-in name, version, and other configuration settings. This properties file can also provide custom settings for your plug-in.

See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.

About the Server FlexResponse plug-in interfaces

The Server FlexResponse plug-in runs on the Enforce Server. The plug-in action is initiated either automatically after an incident is created, or manually when a user pushes a custom button for a Smart Response remediation.

A Server FlexResponse plug-in uses the following interfaces:

- `Plugin`

  A single instance of this interface is constructed when the plug-in is loaded, at the time that the Symantec Data Loss Prevention services start.

  See "Deploying a Server FlexResponse plug-in" on page 41.

- `IncidentResponseAction`

  A new instance of this interface is constructed for each incident to be remediated.

  See "Using the action of a Server FlexResponse plug-in to remediate an incident manually" on page 50.

Other interfaces are provided for specific functions.

See "About the Server FlexResponse interfaces" on page 55.

A Server FlexResponse plug-in can be constructed with no constructor, or with a constructor that takes no parameters.

See "Implementing the plug-in interface and action" on page 14.

A Server FlexResponse plug-in can also be constructed with an optional constructor that takes a `ConfigurationParameters` object.

See "Implementing the plug-in interface with an optional constructor" on page 15.

Implementing the plug-in interface and action

A complete example of a small Server FlexResponse plug-in is available.

See "Sample (annotated) Server FlexResponse plug-in "Hello World"" on page 31.

Your Server FlexResponse plug-in must use the following interfaces:

- `Plugin`

```
public class myPlugin implements Plugin
```

The singleton instance of this class is constructed when the plug-in loads. This class is a factory class that is responsible for constructing instances of your `IncidentResponseAction` implementation.

- `IncidentResponseAction`

  ```
  public class myIncidentResponseAction implements IncidentResponseAction
  ```

  Instances of this class are constructed when remediation actions for this plug-in are selected in the user interface. This class does the actual remediation work.

Implementing the plug-in interface with an optional constructor

A Server FlexResponse plug-in can include an optional constructor with configuration parameters at the time it is constructed.

A separate properties file for each Server FlexResponse plug-in provides configuration settings.

The Server FlexResponse plug-in should extend the `ConfiguredPlugin` abstract class.

```
public final class myConfiguredPlugin extends ConfiguredPlugin
{
  public myConfiguredPlugin(ConfigurationParameters parameters)
```

See the Javadoc for additional details.

See "Server FlexResponse plug-in Javadoc" on page 11.

The `ConfigurationParameters` object is populated with configuration settings from the plug-in properties file.

These configuration settings can be one of the following types:

- General settings that are stored as strings.
  See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.
- Stored credentials.
  To refer to a stored credential in the properties file, use the following form:

  ```
  name.credential=stored-credential-name
  ```

  The `name` is the name for the credential in your properties file.

  The `stored-credential-name` matches the name of a stored credential that has been defined on the Enforce Server.

See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.

See "About the credential store" on page 17.

When you refer to a stored credential in the plug-in properties file using this form, the Server FlexResponse plug-in framework looks up the stored credential in the database. The framework then adds a reference to that credential in the `ConfigurationParameters`.

See the Javadoc for additional details.

See "Server FlexResponse plug-in Javadoc" on page 11.

Input data for the remediation action

The incident information and the configuration parameters are passed to the `IncidentResponseAction` as input data.

The incident information varies, depending on the Network Discover target type. For example, incident information about a file system target includes the file name, location, and owner. Incident information about an SQL database target includes the row. Interfaces are also provided to obtain specific items in the incident information, for example the file name.

For details about incident information, see the Javadoc for the `Incident` interface.

For details about configuration parameters, see the Javadoc for the `ConfigurationParameters` interface.

See "Server FlexResponse plug-in Javadoc" on page 11.

Output data as action result

The incident response action provides the incident remediation action taken, and provides other status information.

You can also provide error handling and messages.

See "Error handling in the Server FlexResponse plug-in" on page 23.

The incident response also provides the following action results:

- An optional message string that explains what the plug-in did, or what error occurred.
  This message appears in the incident history.
  For international environments, this message should be localized.
- Optional custom attribute values to set on this incident.
- Optional protect or prevent status that indicates whether the Server FlexResponse action has succeeded or failed.

- Optional remediation location. This location is a `String` value with the location where the remediated item is stored.

Authenticating access with scan target and remediation credentials

The Server FlexResponse plug-in can obtain credential information to access targets and to remediate them, for example to quarantine them.

For additional information about setting the credential information in the user interface, see the *Symantec Data Loss Prevention Administration Guide* or the online Help.

To complete remediation, you can get the following information that was entered in the user interface for the Discover target:

- Scan credential
  The scan credential is the read credential needed to access a file.
  Scan credentials are on the Discover target **Scanned Content** tab in the user interface. The permissions can be entered on the line in a file containing the files shares, or from the **Add** option on the **Scanned Content** tab.
  The **Username** and **Password** are passed to the Server FlexResponse action as part of the incident information. The `getScanCredential` method in the `ContentRoot` class has this information.
- Quarantine or Copy Share credential
  This credential is needed to copy files into the quarantine location.
  The **Username** and **Password** credentials from the **Protect** tab in the user interface are available for **Quarantine/Copy Share**.
  The `getRemediationLocation` method in the `RemediatedItem` class has the remediation path. The `getCredential` method in the `RemediationLocation` class has the remediation credentials.
- Protect credential
  The Protect write credential is needed to remove a file from its current location, when it is moved or encrypted.
  The **Username** and **Password** credentials from the **Protect** tab in the user interface, for the **Protect Credential** are available.
  The `getRemediationCredential` method in the `ContentRoot` class has this information.

About the credential store

An authentication credential can be stored as a named credential in a central credential store. It can be defined once, and then referenced by any number of Discover targets. Passwords are encrypted before they are stored.

The credential store simplifies management of user name and password changes.

You can add, delete, or edit stored credentials.

See "Adding new credentials to the credential store" on page 18.

See "Managing credentials in the credential store" on page 18.

The Credential Management screen is accessible to users with the "Credential Management" privilege.

Stored credentials can be used when you edit or create a Discover target.

Adding new credentials to the credential store

You can add new credentials to the credential store. These credentials can later be referenced with the credential name.

To add a stored credential

1. Click **System > Settings > Credentials**, and click **Add Credential**.
2. Enter the following information:

Credential Name	Enter your name for this stored credential. The credential name must be unique within the credential store. The name is used only to identify the credential.
Access Username	Enter the user name for authentication.
Access Password	Enter the password for authentication.
Re-enter Access Password	Re-enter the password.

3. Click **Save**.
4. You can later edit or delete credentials from the credential store.

See "Managing credentials in the credential store" on page 18.

Managing credentials in the credential store

You can delete or edit a stored credential.

To delete a stored credential

1. Click **System > Settings > Credentials**. Locate the name of the stored credential that you want to remove.
2. Click the delete icon to the right of the name. A credential can be deleted only if it is not currently referenced in a Discover target or indexed document profile.

To edit a stored credential

1. Click **System > Settings > Credentials**. Locate the name of the stored credential that you want to edit.
2. Click the edit icon (pencil) to the right of the name.
3. Update the user name or password.
4. Click **Save**.
5. If you change the password for a given credential, the new password is used for all subsequent Discover scans that use that credential.

Authenticating access with a stored credential

The Server FlexResponse plug-in can authenticate access using a named credential in a central credential store. The named credential can be used according to the needs of the Server FlexResponse plug-in. Uses include authentication with a database, a Web service, or a workflow system.

The named credential is defined and the user name and password is set on the Enforce Server.

See "About the credential store" on page 17.

A reference to the named credential is then provided in the Server FlexResponse plug-in properties file.

See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.

Creating the plug-in JAR file

Use a Java compiler and tools to create a JAR file for each Server FlexResponse plug-in. When you build the JAR file for your plug-in, you must include the Server FlexResponse API library in your classpath. This library is in the file `flexresponseapi.jar` in the installation directory:

```
SymantecDLP\Protect\tomcat\webapps
    \ProtectManager\WEB-INF\lib\
```

See "Server FlexResponse plug-in API library" on page 11.

At a minimum, your JAR file should have classes to implement the following interfaces:

- `Plugin`

  or

  `ConfiguredPlugin`

- `IncidentResponseAction`

About the plug-in JAR container and external JAR dependencies

The container in which your JAR file is deployed includes all of the public JRE classes provided by the JVM installed with Symantec Data Loss Prevention. The container also includes all of the FlexResponse API classes described in this document (classes in the `com.symantec.dlpx` package hierarchy).

Your FlexResponse plug-in code may have dependencies on other JAR files that are not provided by the plug-in container. Place any external JAR files that you require in the `\plugins` directory of the Enforce Server where the FlexResponse plug-in is deployed. Then reference the JAR dependency in the `plugin.properties` file using the `com.symantec.dlpx.flexresponse.Plugin.plugins` property.

See "Adding a Server FlexResponse plug-in to the plug-ins properties file" on page 42.

As an alternative, reference the dependent JAR file in the manifest of your plug-in JAR.

Creating your plug-in properties file

Each Server FlexResponse plug-in has its own properties file that specifies a plug-in name and a unique identifier.

You can also add additional custom properties to this file. When you create your Server FlexResponse plug-in, you can plan for any custom properties that are available to your plug-in `IncidentResponseAction`.

See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.

Editing the plug-ins properties file

The `Plugins.properties` file must be edited to specify the plug-ins that are installed and should be loaded on the Enforce Server.

See "Adding a Server FlexResponse plug-in to the plug-ins properties file" on page 42.

Best practices for developing a Server FlexResponse plug-in

You can report errors and messages in the logs and in the incident history detail.

You should update the protect or prevent status.

For international environments, localize the messages.

About logs and messages

The incident history detail displays the plug-in output. If any errors occur, the error messages are also displayed.

For international environments, these messages should be localized.

The plug-in can update the protect or prevent status, remediation location, and custom attributes for an incident.

See "Output data as action result" on page 16.

See "Set the plug-in action execution status and message" on page 26.

See "Set the protect or prevent status" on page 26.

See "Write custom attributes" on page 28.

You can view the **Incident Detail** from the Discover reports to view the current values.

See the *Symantec Data Loss Prevention Administration Guide* or the online Help for information about how to view the **Incident Detail**.

The protect or prevent status has several possible values. See the Javadoc for `PreventOrProtectStatus`.

The following events are logged on the Enforce Server in the Manager operational log:

- Plug-in environment loaded.
- Plug-in loaded.
- Plug-in protect action invoked.
- Plug-in response.
- Timeout that is reached on plug-in invocation.

For more information about operational logs, see appendix A in the *Symantec Data Loss Prevention Administration Guide*.

The Enforce Server operational logs are stored in the *SymantecDLP*`\Protect\logs\` directory.

About localization

Java enables localization and internationalization.

Text data that is passed to the plug-in is all in Unicode, as Java `String` objects.

Values in the plug-in properties file can be in Unicode. Any of the values that are passed to the plug-in preserve the localized characters.

The values of custom attributes are also in Unicode.

Error messages from the Enforce Server are localized.

To support multiple locales, you can externalize the error messages in your Server FlexResponse plug-in into resource bundles in the Java platform. For additional information about resource bundles, see the Java platform documentation.

http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles/

The quarantine example in the release has an example of externalizing your messages.

See "Server FlexResponse plug-in Java code examples" on page 11.

Testing the Server FlexResponse plug-in

You should test a new Server FlexResponse plug-in before deploying it in a production environment.

To test a Server FlexResponse plug-in

1. Develop, configure, and deploy your plug-in.

   See "Developing a Server FlexResponse plug-in" on page 13.

   See "Adding a Server FlexResponse plug-in to the plug-ins properties file" on page 42.

   See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.

   See "Deploying a Server FlexResponse plug-in" on page 41.

2. Write a unit test for your plug-in. You can stub out the classes that the Server FlexResponse API provides.

   You can also use online mocking frameworks, such as the following:

   - Mokito, which can be obtained from the Web site http://mockito.org/
   - EasyMock, which can be obtained from the Web site http://easymock.org/

   After the unit test runs correctly, you can test your plug-in with the Symantec Data Loss Prevention system.

   Set up your Symantec Data Loss Prevention system, with test policies, Discover targets, and Automated or Smart Response Rules.

See the *Symantec Data Loss Prevention Administration Guide*.

3 Run some scans with your test Discover targets. Make sure that you see some incidents in the reports.

See the *Symantec Data Loss Prevention Administration Guide*.

4 Navigate to one of the Discover test incidents to verify that automated FlexResponse actions were completed. Click the incident to display the incident detail.

5 For executing the action as a Smart Response, a button should be displayed above the incident number with the rule name of your plug-in.

6 To test manual execution of a FlexResponse action, click the button with your rule name. Wait until the response action has completed. Refresh the Enforce Server page to update the status.

7 View the incident history (**History** tab) in the incident snapshot and note any messages from the plug-in response action.

8 Verify that the remediation actions have been performed.

Error handling in the Server FlexResponse plug-in

When a Server FlexResponse plug-in action is executed as part of a Smart Response rule, you can report failures to the user.

You should always validate the incident type at the beginning of your action.

See "Validate the incident type in a Server FlexResponse action" on page 24.

If a plugin executes as part of a Smart Response action, the plugin should communicate the result of its execution back to the user. This information can be communicated in the following ways:

- Set the protect or prevent status.
  This status is visible as an icon when you view a list of incidents. The status tells the user what has happened to apply prevention for Network incidents or protection for Discover incidents. The status indicates whether the file was quarantined, blocked, or if the attempt to protect or prevent failed. Depending on what happens in the plug-in action, you may or may not want to change this status.
  See "Set the protect or prevent status" on page 26.
- Set the plug-in action execution status and status message.
  This status indicates if the action executed successfully or not. The status is written as an entry into the incident history.

When you return from an action, you can include a message that communicates what happened when the action executed.

See "Set the plug-in action execution status and message" on page 26.

- Throw an exception.

  The plug-in action can throw an exception, and this exception is communicated both in the incident history and in the operational log.

  See "Throw an exception" on page 27.

- Write custom attributes.

  The plug-in action can write to one or more custom attributes. You can then run any reports that filter or sort using these custom attributes.

  See "Write custom attributes" on page 28.

When a Server FlexResponse plug-in action is executed as part of an automated response rule, the Incident Persister process on the Enforce Server administration console executes the plug-in. You can view information about the execution of plug-ins in the `VontuIncidentPersister.log` debug log file. See the *Symantec Data Loss Prevention Administration Guide*.

Validate the incident type in a Server FlexResponse action

The action for your Server FlexResponse plug-in needs to handle the possibility that you get some incidents that are not appropriate for your action. Rather than trying to process any incident, you need to first check the type of incident. Verify that the incident type is the kind you can handle and are interested in.

Make sure that input to your Server FlexResponse plug-in is valid at the beginning of the action.

Check the content type. Each incident has a `ContentItem` associated with it that describes the item. For example, the `ContentItem` can be a file, an SQL database row, or an email message that caused the incident. Verify that the item is of the right type for your plug-in action.

The list of all possible items is in the Javadoc. Go to the entry for `ContentItem` and look at all the interfaces that extend that interface.

See "Server FlexResponse plug-in Javadoc" on page 11.

For example, a quarantine action only applies to items that are files. A quarantine action should make sure that the item is of type `FileServerItem`. If it is not of this type, then the action throws an `UnsupportedIncidentTypeException`. This exception is provided as part of the Server FlexResponse API and is intended to be used specifically when you have the wrong kind of incident.

```
ContentItem item = incident.getContentItem();
if (! (item instanceof FileServerItem))
{
   throw new UnsupportedIncidentTypeException
         ("This plugin only supports incidents on files.");
}
```

Note: When the content item is of type `FileServerItem`, the target type can only be of type `FILESYSTEM_SERVER`, so no further testing is needed.

In some use cases, the plug-in may have to take different actions or reject the remediation request based on the URL type that is returned from the `getLocation()` method of the `FileServerItem` class.

The following possible URL types are differentiated by their formats:

- Shares accessible through the Windows redirector
  For example:

  ```
  \\server\share\...
  smb:\\server\share\...
  ```

- PST file email items
  For example:

  ```
  \\server\share\...\*.pst:folder:email subject
  smb:\\directory\...\*.pst:folder:email subject
  drive letter\directory\...\*.pst:folder:email subject
  ```

- Share accessible though a specific protocol that Windows redirector does not support
  For example:

  ```
  nfs://server/directory/...
  sftp://server/directory/...
  ```

- Windows local file system
  For example:

  ```
  drive letter\directory\...
  ```

- Linux local file system
  For example:

```
/directory/...
```

Note: For parsing file paths, the characters '/' and '\' must be treated interchangeably.

Set the protect or prevent status

A plug-in action can specify the Server FlexResponse completion status by calling `setPreventOrProtectStatus()` when you build an `ActionResult`. To indicate that the Server FlexResponse succeeded (for example, the file that caused the incident was deleted), write the following code when you return from your action:

```
return new ActionResultBuilder()
  .setMessage("The document with the policy violation was removed")
  .setPreventOrProtectStatus
       (PreventOrProtectStatus.FLEX_RESPONSE_EXECUTED)
  .getActionResult();
```

See the Javadoc for `PreventOrProtectStatus` for the full list of status values that are available. Except for very specific use cases, only `FLEX_RESPONSE_EXECUTED` or `FLEX_RESPONSE_ERROR` should be used.

In some cases you may not want to set the protect or prevent status at all, for example if the requested remediation was previously successfully executed. If you do not want to set this status, omit the call to `setPreventOrProtectStatus()` when you build your `ActionResult`.

Set the plug-in action execution status and message

In some cases when the action execution fails, you normally do not want to update the protect or prevent status. For example, if a file for a given incident has been quarantined, your action can attempt to unquarantine the file. If that action fails, you do not want to eliminate the status that the file is quarantined.

However, you can still record the execution status in the incident history. To indicate an execution failure without changing the protect or prevent status, use the `setActionSuccessful()` method on the `ActionResultBuilder`, as in the following example:

```
return new ActionResultBuilder()
      .setMessage("Unquarantine failed: specified file not found")
      .setActionSuccessful(false)
      .getActionResult();
```

When you follow this example, the incident history records the fact that the Server FlexResponse execution failed, and displays the message you provide.

In this example code, if your plugin display name is `MyPlugin` then you see the following message in the incident history:

```
[MyPlugin] FlexResponse Action Failed with message
     Unquarantine failed: specified file not found
```

By default if you do not specify an action execution status, your plug-in is recorded in the incident history as a successful execution.

For example you can return an `ActionResult`, as in the following example:

```
return new ActionResultBuilder()
      .setMessage("No errors encountered")
      .getActionResult();
```

Then you see the following message in the incident history:

```
[MyPlugin] FlexResponse Action Successful with message
     No errors encountered.
```

If you have no message that you want to communicate on success, you can build an `ActionResult` with no message:

```
return new ActionResultBuilder()
      .getActionResult();
```

Then the incident history displays the following message:

```
[MyPlugin] with message
```

Throw an exception

If an exception occurs in your plug-in action, in most cases you should catch the exception and log it. You would then set the Server FlexResponse execution status and the protect or prevent status appropriately. The following example shows this sequence:

```
Logger logger = Logger.getLogger(MyPluginAction.class.getName());

try
{
    doAction();
}
catch (IOException ioe)
```

```
{
    logger.log(Level.SEVERE, ioe.getMessage(), ioe);

    // Don't change the prevent/protect status on failure
    return new ActionResultBuilder()
         .setMessage("Error execution action: " + ioe.getMessage())
         .setActionSuccessful(false)
         .getActionResult();
}

return new ActionResultBuilder()
     .setMessage("Action succeeded")
     .setPreventOrProtectStatus
          (PreventOrProtectStatus.FLEX_RESPONSE_EXECUTED)
     .getActionResult();
```

If your plug-in does throw an exception, the plug-in invocation framework logs the exception to the operational log. It also marks the execution status in the incident history as failed, along with the message of the exception.

Note: The protect or prevent status is reset back to its original value. If this reset is not what you want, you should catch any exceptions and return an `ActionResult` with the desired protect or prevent status.

Your action can throw the specific exception `UnsupportedIncidentTypeException`. You should throw this exception if your action does not support the incident type for the incident that is associated with this action. For example, if your action quarantines a file, and the incident is associated with an email message, you should throw an `UnsupportedIncidentTypeException`.

The following example shows this exception:

```
ContentItem contentItem = incident.getContentItem();
if (! (contentItem instanceof FileServerItem))
{
    throw new UnsupportedIncidentTypeException("I only do file items");
}
```

Write custom attributes

You may want to set a custom attribute in your action. One use case is when you want to filter your incidents based on whether the execution of a Server

FlexResponse action has succeeded or failed, and you do not want to modify the protect or prevent status.

If you set the specific execution status, the history is updated, but you cannot filter incidents based on this history status entry.

However, you can create a custom attribute and set this attribute to reflect execution status. Then you can filter incidents based on the custom attribute.

For example, you can create a custom attribute called FlexResponse Action Status. Then you can set this attribute to the value "SUCCEED" or the value "FAIL" when you return from your action.

You can filter incidents in an incident report based on the Server FlexResponse actions that failed, and can take further action.

The following example sets custom attributes to indicate the execution status. Note that the `getCustomAttribute()` method may throw an `IllegalArgumentException` if the custom attribute does not exist, but you can allow the action execution framework to log that correctly.

```
public ActionResult execute(Incident incident,
            ConfigurationParameters configurationParameters)
            throws UnsupportedIncidentTypeException,
               InterruptedException
{
    CustomAttribute actionStatusAttribute =
        incident.getCustomAttribute("FlexResponse Action Status");

    try
    {
        doAction();
    }
    catch (Exception e)
    {
        return new ActionResultBuilder()
             .setMessage("Action failed with message " + e.getMessage())
             .addCustomAttribute(actionStatusAttribute, "FAIL")
             .setActionSuccessful(false)
             .getActionResult();
    }

    return new ActionResultBuilder()
         .addCustomAttribute(actionStatusAttribute, "SUCCEED")
         .setMessage("Action executed successfully")
```

```
            .getActionResult();
}
```

Troubleshooting a Server FlexResponse plug-in

Table 2-1 has troubleshooting issues and suggestions for diagnosing Server FlexResponse problems.

Table 2-1 Troubleshooting suggestions

Issue	Suggestions
During creation of a Smart Response Rule, the drop-down menu does not display the action **All: Server FlexResponse**. During creation of an automated Response Rule, the drop-down menu does not display the action **All: Server FlexResponse**. If you have multiple plug-ins, your plug-in name does not display in the **All: Server FlexResponse** drop-down menu.	This issue happens because your plug-in did not load. At the end of the file `Plugins.properties`, enter the name of your plug-in JAR file on the list of plug-ins. Make sure that this line is not commented out. Restart both the Vontu Incident Persister and Vontu Manager services to load your plug-in. Your plug-in properties file and plug-in code may not match appropriately. Look at the Tomcat log for errors. The log file is `localhost.`*`date`*`.log`. This log file is in `SymantecDLP\Protect\logs\tomcat`. To verify that your plug-in is loaded, look for Enforce system event (2122). This event lists all the plug-ins that are loaded.
Your plug-in does not execute successfully.	Check the incident snapshot history for messages from your plug-in and the plug-in framework. For Smart Responses, look at the Tomcat log for errors. This log is in `SymantecDLP\Protect\logs\tomcat`. The log file is `localhost.`*`date`*`.log`. For automated responses, look at the `VontuIncidentPersister.log` debug log file.

See "Best practices for developing a Server FlexResponse plug-in" on page 20.

Sample (annotated) Server FlexResponse plug-in “Hello World”

Use this example to help develop your Server FlexResponse plug-in.

Some examples are also provided with the Symantec Data Loss Prevention release.

See “Server FlexResponse plug-in Java code examples” on page 11.

The following example code of a Server FlexResponse plug-in and action is annotated.

Table 2-2 is the example code for the plug-in, in the file `HelloWorldPlugin.java`.

Table 2-3 has the annotations of the example code for the plug-in.

Set and access the plug-in metadata in the Java code in the file `MyPluginMetadata.java`.

Table 2-4 is the example code for the plug-in action, in the file `HelloWorldAction.java`.

Table 2-5 has the annotations of the example code for the plug-in action.

Table 2-6 has the example plug-in properties file.

Table 2-2 Example code for a plug-in

Line number	Java code
1	`package com.symantec.dlpx.flexresponse.examples.helloworld;`
2	
3	`import com.symantec.dlpx.flexresponse.Plugin;`
4	`import com.symantec.dlpx.flexresponse.PluginMetadata;`
5	`import com.symantec.dlpx.flexresponse.action.`
6	`   IncidentResponseAction;`
7	
8	`/**`
9	`  * An example of a simple FlexResponse plugin.`
10	`  *`
11	`  * This plugin provides an action that simply returns the action`
12	`  *   result message`
13	`  *   "Hello World! This item has been remediated."`
14	`  *   for every incident.`
15	`  */`
16	`public class HelloWorldPlugin implements Plugin`
17	`{`
18	`   public PluginMetadata getMetadata()`
19	`   {`
20	`      return new MyPluginMetadata();`
21	`   }`
22	
23	`   public IncidentResponseAction newIncidentResponseAction()`
24	`   {`
25	`      return new HelloWorldAction();`
26	`   }`
27	`}`
28	

Table 2-3 Description for the plug-in code

Line numbers	Description
16	Declaration of the class for your custom plug-in. Note that your class declaration implements the Plugin interface from the Server FlexResponse plug-in API library.
18-21	Method to set the metadata.
23	The Server FlexResponse plugin framework uses this method to get a new instance of your plugin action.

Table 2-3 Description for the plug-in code *(continued)*

Line numbers	Description
25	Return a new instance of your plugin action.

Set and access the plug-in metadata in the Java code in the file `MyPluginMetadata.java`.

```
package com.symantec.dlpx.flexresponse.examples.helloworld;

import com.symantec.dlpx.flexresponse.PluginMetadata;

import java.util.Map;
import java.util.Collections;

/**
 * This class sets the plug-in metadata.
 * This method is not used in this release,
 * but is a placeholder for future releases.
 */
public class MyPluginMetadata implements PluginMetadata
{
   public String getVendor()
   {
      return "<Company>";
   }

   public String getVersion()
   {
      return "2.0";
   }

   /**
    * @return null, which requires the display name to be
    *    set in the plugin configuration file. Alternatively,
    *    this method can return a String value. This value is
    *    used by the plug-in framework unless it is overwritten
    *    by the value provided in the configuration file.
    */
   public String getDefaultDisplayName()
   {
      return null;
   }
```

```
   /**
    * @return null, which requires the plugin identifier to be
    *    set in the plugin configuration file. Alternatively,
    *    this method can return a String value. This value is
    *    used by the plug-in framework unless it is overwritten
    *    by the value provided in the configuration file.
    */
   public String getDefaultIdentifier()
   {
      return null;
   }


/**
 * This method is not used in this release,
 * but is a placeholder for future releases.
 */
   public Map<String, Object> getAttributes()
   {
      return Collections.emptyMap();
   }
}
```

Table 2-4 Example code for a plug-in action

Line number	Java code
1	`package com.symantec.dlpx.flexresponse.examples.helloworld;`
2	
3	`import com.symantec.dlpx.flexresponse.action.ActionResult;`
4	`import com.symantec.dlpx.flexresponse.action.ActionResultBuilder;`
5	`import com.symantec.dlpx.flexresponse.action.IncidentResponseAction;`
6	`import com.symantec.dlpx.flexresponse.configuration.`
7	`   ConfigurationParameters;`
8	`import com.symantec.dlpx.flexresponse.incident.Incident;`
9	`import com.symantec.dlpx.flexresponse.incident.PreventOrProtectStatus;`
10	
11	`/**`
12	` * An example of a simple FlexResponse IncidentResponseAction.`
13	` *`
14	` * This action simply returns the action result message`
15	` *   "Hello World! This item has been remediated."`
16	` *   for every incident.`
17	` */`
18	`public class HelloWorldAction implements IncidentResponseAction`
19	`{`
20	`   private static final String ACTION_RESULT_MESSAGE =`
21	`      "Hello World! This item has been remediated.";`
22	
23	`   public ActionResult execute(Incident incident,`
24	`                       ConfigurationParameters parameters)`
25	`   {`
26	`      return new ActionResultBuilder()`
27	`             .setMessage(ACTION_RESULT_MESSAGE)`
28	`             .setPreventOrProtectStatus`
29	`               (PreventOrProtectStatus.FLEX_RESPONSE_EXECUTED)`
30	`             .getActionResult();`
31	`   }`
32	`}`
33	

Table 2-5 Description for the plug-in action code

Line numbers	Description
18	Declaration of the class for your custom plug-in. Note that your class declaration implements the `IncidentResponseAction` interface from the Server FlexResponse plug-in API library.

Table 2-5 Description for the plug-in action code *(continued)*

Line numbers	Description
23-24	Declaration of the method to execute your remediation action. The input parameters for the incident information and configuration parameters are passed in. This example does not use the information in these parameters. A more complex action can use the information in these input parameters. See "Input data for the remediation action" on page 16.
26-30	Call for the plug-in action and return of the result. Note that this example uses the `ActionResultBuilder` class from the Server FlexResponse plug-in API library. It also uses the `getActionResult` method from this class in the library. The result of the action must be returned. See "Output data as action result" on page 16. The status and a status message are also returned. See "Error handling in the Server FlexResponse plug-in" on page 23.

For this Server FlexResponse plug-in, a properties file is needed to set the display name and identifier, if the plug-in metadata class does not provide these values.

Table 2-6 illustrates the properties file.

Table 2-6 Plug-in properties file

Line number	Properties file

1
2
3
4
	`# Primary plugin metadata`
	
	`display-name=Hello World FlexResponse Plugin`
	`plugin-identifier=MyCompany-hello-world-plugin`

Developing, deploying, configuring, and testing a sample Server FlexResponse plug-in

The following example provides the complete process for implementing, configuring, and testing a Server FlexResponse plug-in.

This example assumes that Symantec Data Loss Prevention has been installed on a test server. It also assumes that you have administrative access to the server (to

create files), and to an account on the Enforce Server administration console. The example plug-in is called `HelloWorld`.

In this example, *SymantecDLP* represents the Symantec Data Loss Prevention installation directory. If the default settings are used, this directory is `c:\SymantecDLP` on the Windows platform and `/opt/SymantecDLP` on the Linux platform.

To deploy, configure, and test an example Server FlexResponse plug-in

1. Create a Server FlexResponse plug-in by entering the example Java code.

   See "Sample (annotated) Server FlexResponse plug-in "Hello World"" on page 31.

   You can also copy one of the Java source examples from the release.

   See "Server FlexResponse plug-in Java code examples" on page 11.

2. Create a JAR file `HelloWorld.jar` from the Java source files. At compile time and at run time, the Server FlexResponse API library must be in the classpath.

   The library is in the file `flexresponseapi.jar` in the following directory:

   ```
   SymantecDLP\Protect\tomcat\webapps
       \ProtectManager\WEB-INF\lib\
   ```

3. Copy your JAR file to the plug-ins directory in the installation at *SymantecDLP*`\Protect\plugins\`.

4. Create a properties text file `HelloWorld.properties` for the test plug-in.

   The properties file contains the following contents:

   ```
   display-name=Hello World FlexResponse Plug-in
   plugin-identifier=MyCompany-hello-world-plugin
   ```

5. Copy the properties file to the directory *SymantecDLP*`\Protect\plugins\`.

6. Make sure that the Symantec Data Loss Prevention protect user has read and execute access to the files `HelloWorld.jar` and `HelloWorld.properties`.

7. Edit the file *SymantecDLP*`\Protect\config\Plugins.properties`.

8. At the end of this file, place the name of your plug-in JAR file on the list of plug-ins. Remove the comment mark from the beginning of this line.

   ```
   com.symantec.dlpx.flexresponse.Plugin.plugins=HelloWorld.jar
   ```

9. Save this file, and exit the editor.

10 Stop the Windows Vontu Incident Persister and Vontu Manager services, and then restart them. This loads your plug-in.

11 Locate or create a directory on the Enforce Server or on a file share with a word that is repeated a few times.

12 Log on to the Enforce Server administration console.

Refer to the *Symantec Data Loss Prevention Administration Guide* for details of the steps in the Enforce Server administration console.

13 Click **System > Servers > Overview**. Verify that a Network Discover Server is running.

14 Create a policy group by performing the following actions:

Click **System > Policy Groups**.

On the **Policy Group List** screen that appears, click **Add Policy Group**.

Enter a name and description for your new policy group.

Click **Save**.

15 Create a policy by performing the following actions:

Click **Policies > Policy List**.

Click **Add Policy**.

Select **Add a blank policy**.

Click **Next**.

Enter a name and description.

Select your policy group from the drop-down menu.

16 Add a rule to the policy by performing the following actions:

Click **Add Rule**.

Select **Content Matches Regular Expression**.

Click **Next**.

Enter a name for your rule.

Enter a word that is repeated in the file content in your selected file share or directory.

Click **OK**.

Click **Save**.

17 Create a Network Protect response rule for the remediation that uses your Server FlexResponse plug-in by performing the following actions:

Click **Policies > Response Rules**.

Click **Add Response Rule**.

Select **Smart Response**.

Click **Next**.

Enter a name for your test rule. This name is the label on the button that can be selected during the remediation.

Enter an optional description for your test rule.

In the drop-down menu **choose action type** select the action **All: Server FlexResponse**.

Click **Add Action**.

In the drop-down menu **choose a plugin**, select your test plug-in. The name in this drop-down menu is the name in the `display-name` property from either the configuration properties file or the plug-in metadata class.

Click **Save**.

18 Create a Network Discover target by performing the following actions:

Click **Policies > Discover Scanning > Discover Targets**.

Click **New Target > Server > File System**.

Enter a name for this target.

Select a policy group and a detection server.

19 Enter the details for this target by performing the following actions:

Click the **Scanned Content** tab.

Enter a user name and password that has read access to the files to scan.

Click **Add**.

Enter the complete path of the directory or path of the file share to scan.

Click **OK**.

Click **Save**.

20 Run a scan by performing the following actions:

Find your named target in the list.

Click the play icon.

To refresh the page, click the refresh icon until the scan is complete.

When the scan is complete, the time of the scan, the scan information, and the number of incidents are displayed.

21 Click on the number in the incidents column.

22 In the list of incidents, click one of them.

23 In the **Incident Detail** screen above the incident number, your test remediation button displays, showing the name of your rule.

24 Click your Server FlexResponse test remediation button to perform the action in your sample plug-in.

25 Verify the information about your Server FlexResponse test plug-in, and click **OK**.

26 Wait for the remediation to complete.

27 To verify the remediation, click the **History** tab and view the remediation messages from your plug-in. You should see a message that your plug-in was invoked, and another message with the success or failure.

Chapter 3

Deploying and configuring the Symantec Data Loss Prevention Server FlexResponse plug-in

This chapter includes the following topics:

Deploying a Server FlexResponse plug-in

Enable a plug-in for the Server FlexResponse API.

To deploy a Server FlexResponse plug-in

1. Copy the completed Server FlexResponse plug-in JAR file to the plug-ins directory:

   `SymantecDLP\Protect\plugins\`

   See "Creating the plug-in JAR file" on page 19.

2. Configure the plug-in with a properties file.

   See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.

3 Copy the properties file for each plug-in into the directory where you placed your JAR file:

```
SymantecDLP\Protect\plugins\
```

4 In the file *SymantecDLP*`\Protect\config\Plugins.properties`, add the plug-in to the list, and enter the properties for your plug-in.

See "Adding a Server FlexResponse plug-in to the plug-ins properties file" on page 42.

5 Make sure that the Symantec Data Loss Prevention protect user has read and execute access to both the plug-in JAR file and the plug-in properties file.

6 To load the plug-in, stop the Vontu Incident Persister and Vontu Manager services, and then restart them.

Adding a Server FlexResponse plug-in to the plug-ins properties file

Add a Server FlexResponse plug-in to the `Plugins.properties` file. Also, modify any parameters that are necessary for the plug-in.

To add a Server FlexResponse plug-in to the properties file

1 Edit the `Plugins.properties` file.

General values are in this file for all plug-ins, plus a list of all the plug-ins that are implemented.

See Table 3-1 on page 43.

This file is in the following directory:

```
SymantecDLP\Protect\config
```

2 Locate the following line in the file, which specifies the JAR files of the plug-ins to construct at load time:

```
# Incident Response Action configuration parameters.

com.symantec.dlpx.flexresponse.Plugin.plugins =
  plugin1.jar,plugin2.jar
```

Remove the comment mark from the beginning of the line, if necessary, and replace `plugin1.jar,plugin2.jar` with the names of the plug-in JAR files you want to deploy. Separate multiple JAR files with commas.

3 Edit any additional parameters in this file.

Table 3-1 describes the additional properties for the Server FlexResponse API in the `Plugins.properties` file.

4 Stop the Vontu Incident Persister and Vontu Manager services, and then restart them. This loads the new plug-in and the other parameters in this file.

If you later change the `Plugins.properties` file, you must restart both the Vontu Incident Persister and Vontu Manager services to apply the change.

In Table 3-1 *plugin-id* is a unique identifier of the plugin within this properties file, for example `test1`.

Table 3-1 Parameters in the Plugins.properties file

Property name	Description
protect.plugins.directory	The directory under which all Symantec Data Loss Prevention plug-ins are installed.
com.symantec.dlpx.flexresponse.Plugin.plugins	A comma-separated list of JAR files (or JAR titles) to be loaded in the Server FlexResponse plug-in container. Each plug-in in this list will correspond to a response rule action in the Enforce Server administration console. The container in which your JAR file is deployed includes all of the public JRE classes provided by the JVM installed with Symantec Data Loss Prevention. The container also includes all of the FlexResponse API classes described in this document (classes in the com.symantec.dlpx package hierarchy). Your FlexResponse plug-in code may have dependencies on other JAR files that are not provided by the plug-in container. Place any external JAR files that you require in the `\plugins` directory of the Enforce Server where the FlexResponse plug-in is deployed. Then reference the JAR in this property.
com.vontu.enforce.incidentresponseaction. IncidentResponseActionInvocationService. maximum-incident-batch-size	The maximum number of incidents that can be selected from the incident list report for one Server FlexResponse Smart Response rule invocation. The default is 100. In this release, the maximum value of this parameter cannot exceed 1000.
com.vontu.enforce.incidentresponseaction. IncidentResponseActionInvocationService. keep-alive-time	Do not change the value of this parameter. This parameter is reserved for development and debugging. Use the `timeout` property in the individual plug-in properties file to set the timeout for the execution threads for your plug-in.

Table 3-1 Parameters in the Plugins.properties file *(continued)*

Property name	Description
com.vontu.enforce.incidentresponseaction. IncidentResponseActionInvocationService. serial-timeout	The execution thread timeout for the serial thread executor (global). See the `is-serialized` property in the individual plug-in property file for details.

Creating a properties file to configure a Server FlexResponse plug-in

Specific information and parameters for each Server FlexResponse plug-in are in the *`plug-in-name`*`.properties` file.

Each plug-in must have a separate properties file.

An individual plug-in properties file is not necessary if the plug-in satisfies the following conditions:

- Does not need custom properties.
- Provides the display name and the plug-in identifier in the implementation of the plug-in metadata class.
- Does not need a stored credential.
  See "Authenticating access with a stored credential" on page 19.

To configure a Server FlexResponse plug-in

1 Create a text file that contains the properties for each Server FlexResponse plug-in.

Each JAR file has an optional associated properties file with the same base name as the JAR file. These files are located in the `SymantecDLP\Protect\plugins` directory.

For example, if you have a `plugin1.jar` file, you should create a `plugin1.properties` file.

See "Creating the plug-in JAR file" on page 19.

2 In this file, enter the keys and values of all the parameters for the plug-in:

```
display-name=plugin 1
plugin-identifier=IncidentResponseAction1
```

To update the properties, you must stop the Vontu Manager and Vontu Incident Persister services, and then restart them to load in the new values.

See Table 3-2 on page 45.

3 Make sure that the Symantec Data Loss Prevention protect user has read and execute access to the plug-in properties file.

Table 3-2 describes the properties in the *`plug-in-name`*`.properties` file.

Table 3-2 Parameters in the custom plug-in properties file

Property name	Description
display-name	The name of this plug-in. This name is displayed in the **choose a plugin** drop-down menu when you select an **All: Server FlexResponse** action in a Smart Response rule or an automated response rule. A best practice is to define this property in the plug-in properties file. If you change the value of this name in the properties file after the plug-in is loaded, you must restart the Vontu Incident Persister and Vontu Manager services to load in the new name. Alternatively, this value can be specified in the metadata class. This value is mandatory and it must be specified in at least one place, either in the configuration properties file, or the plug-in metadata class. For international environments, this display name can be in the local language.

Table 3-2 Parameters in the custom plug-in properties file *(continued)*

Property name	Description
plugin-identifier	The identifier for this plug-in. This identifier should be unique for all Server FlexResponse plug-ins on this Enforce Server. A best practice is to define this property in the plug-in properties file. Alternatively, this value can be specified in the metadata class. This value is mandatory and it must be specified in at least one place, either in the configuration properties file, or the plug-in metadata class. If any response rule is assigned to this Server FlexResponse plug-in, do not change this identifier in your properties file.

credential-reference.credential	Specifies a reference to a named credential to authenticate access, for example to an inventory database. The value of this property must refer to a named credential that was defined on the Enforce Server. The credential-reference in the property name provides a method to differentiate between multiple credentials in the properties file.
	
	`inventory-credential.credential=`
	`InventoryDB1`
	
	See "Authenticating access with a stored credential" on page 19.

custom name Example: test1.value.1 test1.value.2	These optional custom parameters are required to pass information to your plug-in. These parameters are passed to each invocation of the plug-in and can optionally be made available at the time this plug-in is constructed.
timeout	Optional parameter with the timeout in milliseconds for the execution threads for this plug-in. The default is 60000 (one minute). If the timeout value is reached, the user interface shows the Server FlexResponse plug-in status as failed, and the incident history is updated with a timeout message. If you change the value of this property in the properties file after the plug-in is loaded, you must stop the Vontu Incident Persister and Vontu Manager services, and then restart them.

Table 3-2 Parameters in the custom plug-in properties file *(continued)*

Property name	Description
maximum-thread-count	Optional parameter with the number of parallel threads available for execution of this plug-in. This parameter is ignored if `is-serialized` is set. The default is 2. If you change the value of this property in the properties file after the plug-in is loaded, you must stop the Vontu Incident Persister and Vontu Manager services, and then restart them.
is-serialized	The value of this parameter can be true or false. Set this optional parameter to true if this plug-in execution must be serialized (one thread at a time). All serialized plug-ins share a single execution thread. If this parameter is set, then `timeout` and `maximum-thread-count` are ignored. The default is false. If you change the value of this property in the properties file after the plug-in is loaded, you must stop the Vontu Incident Persister and Vontu Manager services, and then restart them.

Chapter 4

Using the Symantec Data Loss Prevention Server FlexResponse plug-in to remediate an incident

This chapter includes the following topics:

Configuring the Server FlexResponse action

The **All: Server FlexResponse** action enables you to remediate any incident type using a custom, server-side FlexResponse plug-in. You can configure a Server FlexResponse response action for either automated response rules or smart response rules.

The **All: Server FlexResponse** action is available only if you have licensed Network Protect and you have deployed one or more Server FlexResponse plug-ins to Symantec Data Loss Prevention.

See "Deploying a Server FlexResponse plug-in" on page 41.

To configure a Server FlexResponse action

1. Log on to the Enforce Server administration console.
2. Create a new Response Rule for each custom Server FlexResponse plug-in.

   Click **Manage > Policies > Response Rules**.
3. Click **Add Response Rule**.
4. Select either **Automated Response** or **Smart Response**. Click **Next**.
5. Enter a name for the rule in the **Rule Name** field. (For Smart Response rules, this name appears as the label on the button that incident responders select during remediation.)
6. Enter an optional description for the rule in the **Description** field.
7. In the **Actions (executed in the order shown)** menu, select the action **All: Server FlexResponse**.
8. Click **Add Action**.
9. In the **FlexResponse Plugin** menu, select a deployed Server FlexResponse plug-in to execute with this Response Rule action.

   The name that appears in this drop-down menu is the value specified in the `display-name` property from either the configuration properties file or the plug-in metadata class.

   See "Deploying a Server FlexResponse plug-in" on page 41.
10. Click **Save**.
11. Repeat this procedure, adding a Response Rule for any additional Server FlexResponse plug-ins that you have deployed.

Locating incidents for manual remediation

To manually execute the plug-in action configured in a Smart Response Rule, use the reports on the Enforce Server to select incidents for remediation.

To locate incidents for manual remediation

1. Log on to the Enforce Server administration console.
2. Click **Incidents > Discover**.
3. Select an incident (or multiple incidents) for remediation. You can use the standard reports or report filters to narrow the list of incidents.
4. You can select either a group of incidents, or one incident for remediation:

- From the list of incidents, check the box to the left of each incident to select that incident for remediation. You can select multiple incidents.
- From the list of incidents, select all incidents on this page by clicking the check box on the left of the report header.
- From the list of incidents, select all incidents in the report by clicking the **Select All** option on the upper-right side of the report.
- Click one incident to display the **Incident Detail**, and select that one incident for possible remediation.

After you have selected the incidents for remediation, you can manually remediate them.

See "Using the action of a Server FlexResponse plug-in to remediate an incident manually" on page 50.

Using the action of a Server FlexResponse plug-in to remediate an incident manually

After you have selected an incident, or group of incidents to remediate, you can invoke the action of a Smart Response rule. This action uses your custom Server FlexResponse plug-in to remediate the incidents manually.

To remediate a single incident

1. Be familiar with the response rules that are available to manually remediate an incident.

   Click **Policies > Response Rules**.

   The **Conditions** column indicates which rules can be executed manually.

2. Select a single incident, and display the **Incident Detail**.

   See "Locating incidents for manual remediation" on page 49.

3. In the **Incident Detail** screen above the incident number, your remediation options display. These options show the names of your response rules.
4. Click a Server FlexResponse plug-in remediation button to perform the remediation action.

5 View the remediation action. Click **OK**.

6 Verify that the remediation is complete. Some remediation actions may take a long time, for example encryption of a large file. To see user interface updates, click the refresh icon in the upper-right corner of the report. Refresh the page until you see the green success or red failure icon in the incident details.

See "Verifying the results of an incident response action" on page 51.

To remediate a selected group of incidents

1 Select incidents from an incident list report. Check the box at the left of the selected incidents.

Alternatively, you can select all incidents on a page or on a report.

See "Locating incidents for manual remediation" on page 49.

2 **Incident Actions** becomes a drop-down menu.

3 From the **Incident Actions** drop-down menu, select **Run Smart Response** and then select your custom Server FlexResponse.

4 View the remediation action. Click **OK**.

5 Verify that the remediation is complete. Some remediation actions may take a long time, particularly if several incidents were selected. To see user interface updates, click the refresh icon in the upper-right corner of the report. Refresh the page until you see the green success or red failure icon in the incident details.

See "Verifying the results of an incident response action" on page 51.

Verifying the results of an incident response action

You can verify that a remediation action has been completed by using the **History** tab of an incident.

To verify the results of an incident response action for a single incident

1 Log on to the Enforce Server administration console.

2 Click **Incidents > Discover**.

Look for the green success or red failure icons in the incident report.

3 For additional information about the results, click one incident to display the **Incident Detail**.

4 Click the **History** tab.

5 View the remediation messages from your plug-in. A message that your plug-in was invoked, and another message with the success or failure should display. Other messages may also display, with the status result or remediation result.

To verify the results of an incident response action for a group of incidents

1 Log on to the Enforce Server administration console.

2 Click **Incidents > Discover**.

3 Use report filters and summaries to display the protect or prevent status of the incidents.

Custom reports can also be created to show the protect or prevent status, or the values of custom attributes.

Appendix A

Symantec Data Loss Prevention Server FlexResponse plug-in examples

This appendix includes the following topics:

About the Server FlexResponse plug-in examples

The following examples of the Server FlexResponse plug-in have Java code in the release.

See “Server FlexResponse plug-in Java code examples” on page 11.

- “Hello World”
  See “Sample code for the “Hello World” example” on page 54.
- Quarantine
  See “Sample code to quarantine files” on page 54.

Sample code for the “Hello World” example

The “Hello World” example is a simple plug-in. This example is annotated to assist you in developing your first Server FlexResponse plug-in.

See “Sample (annotated) Server FlexResponse plug-in “Hello World”” on page 31.

Refer to the release examples for the code.

See “Server FlexResponse plug-in Java code examples” on page 11.

Sample code to quarantine files

This Server FlexResponse plug-in example Java code provides all the plug-in code for a plug-in that can potentially quarantine files. The code for doing the actual work to quarantine the file is stubbed out in this example.

This plug-in provides an action that gets information about the incident and then updates its custom attributes and remediation location.

The quarantine example has sample code for externalizing your localized messages.

Refer to the release examples for the code.

See “Server FlexResponse plug-in Java code examples” on page 11.

Appendix B

Symantec Data Loss Prevention Server FlexResponse interfaces

This appendix includes the following topics:

About the Server FlexResponse interfaces

The API library in this release provides many interfaces to implement custom Server FlexResponse plug-ins.

See "Components of the Server FlexResponse plug-in" on page 11.

The Javadoc in the release directory describes all the classes and interfaces.

The Javadoc is in the following directory:

SymantecDLP\Protect\tools\flexresponse\javadoc\

Open the file `index.html`, as an initial page to view the Javadoc.

Summary of required interfaces

The following classes are required:

- Plugin

```
public class myPlugin implements Plugin
```

A class responsible for instantiating the plug-in metadata and the plug-in actions.

- IncidentResponseAction

  ```
  public class MyAction implements IncidentResponseAction
  ```

  The class containing the code to perform the plug-in response action.
  This class implements the execute method, a command for executing custom incident response actions.

  ```
  public ActionResult execute(Incident incident,
                  ConfigurationParameters parameters)
  ```

  The following parameters are passed:

  - `Incident`
    The incident that the user selected for this incident response action.
  - `ConfigurationParameters`
    Additional configuration parameters for this response from the properties file for this plug-in.
    This method returns a success message to store in the incident history. If the action was unsuccessful, then it returns an error result.
    See "Error handling in the Server FlexResponse plug-in" on page 23.

Summary of other interfaces

Other optional interfaces are documented in the Javadoc.

The following interfaces are commonly used:

- `ActionResult`
  This interface assists in building the result of the remediation action.
- `ConfigurationParameters`
  This interface contains the structure of the parameters from the custom plug-in properties file.
  See "Creating a properties file to configure a Server FlexResponse plug-in" on page 44.
- `ContentItem`
  This interface contains several subinterfaces that describe the structure of the incident information for the different types of Network Discover targets.
- Incident

This interface contains the structure of the incident information that is common to all Symantec Data Loss Prevention incidents. It also contains the `ContentItem`.