SecureTransport

Version 5.4

2 April 2024

Developer's Guide

This documentation describes the following Axway software:

Axway SecureTransport 5.4

Revision history

The following changes are added to the SecureTransport 5.4 Developer's Guide:

SecureTransport version	Document revision number	Topics updated
5.4	5.4.01 – initial version	
5.4	5.4.02	A new *Connection interface implementation version 1.3* on page 24 subsection is added to the *Implement a custom protocol connector* on page 16 topic.
5.4	5.4.03	A new *Authentication SPI versioning* on page 97 subsection is added to the *Pluggable authentication* on page 84 topic.
5.4	5.4.04	A new *Connection interface implementation version 1.4* on page 25 subsection is added to the *Site SPI versioning* on page 17 subtopic.
5.4	5.4.05	A new *Authorization SPI versioning* on page 100 subsection is added to the *Pluggable authorization* on page 99 topic.
5.4	5.4.06	New methods added to the *Pluggable authorization* on page 99 topic.
5.4	5.4.07	A new method added to the *Pluggable authentication* on page 84 topic.
5.4	5.4.08	Two new services - Account Attributes Data service and Transfer Attributes Data service - added to the *Implement a custom protocol connector* on page 16 topic.
5.4	5.4.09	• *Implement a custom protocol connector* on page 16 updated with SPI v1.7 • *Pluggable authentication* on page 84 • *Pluggable authorization* on page 99 • *Custom Advanced Routing step* on page 112 • Expression language support

Contents

Preface

The guide describes the components and tasks associated with customizing the Axway SecureTransport Server. It provides background, conceptual, and procedural information for extending SecureTransport Server functionalities by consuming and implementing available SecureTransport APIs . Before implementing the tasks described in this manual, make sure that SecureTransport is installed as described in the *SecureTransport Installation Guide*.

Who should read this document

This guide is intended for developers who do integrations with SecureTransport. Developers should follow the guide lines in this document in order to successfully create custom connectors and use the SecureTransport REST API. The set of technologies and programming languages that can be used to customize SecureTransport depends on the specific SecureTransport API that will be implemented.

Related documentation

SecureTransport provides the following documentation:

- *SecureTransport Administrator's Guide* – This guide describes how to use the SecureTransport Administration Tool to configure and administer your SecureTransport Server. The content of this guide is also available in the Administration Tool online help.
- *SecureTransport REST API documentation* – The portal published API documentation derived from the API swagger documents. To access the administrator API documentation, go to SecureTransport Administrator API v1.4. To access the end-user API documentation, go to SecureTransport End-User API v1.4.
- *SecureTransport Appliance Guide* - This guide provides the SecureTransport Appliance installation, configuration, and operation instructions. It also provides SecureTransport installation and upgrade instructions for Axway Appliances.
- *SecureTransport Capacity Planning Guide* – This guides provides information useful when planning your production environment for SecureTransport.
- *SecureTransport Developer's Guide* – This guide provides the descriptions and usage of the plug-able information for the SecureTransport Pluggable Transfer Site and how to implement a Pluggable Transfer Site. It also provides Swagger REST API integration instructions and custom Address Book source implementation instructions and custom plugins/exits source implementation instructions.
- *SecureTransport Getting Started Guide* – This guide explains the initial setup and configuration of SecureTransport using the SecureTransport Administrator setup interface.
- *SecureTransport Installation Guide* – This guide explains how to install and uninstall SecureTransport on UNIX-based platforms and Microsoft Windows.

- *SecureTransport Release Notes* – This document contains information about new features and enhancements, information received after the finalization of the rest of the documentation, and a list of known and fixed issues.
- *SecureTransport Security Guide* – This guide provides security information necessary for the secure operation of the SecureTransport product.
- *SecureTransport Software Development Kit (SDK)* – A set of software development tools and examples that allow extending SecureTransport by consuming and implementing available APIs.
- *SecureTransport Upgrade Guide* - This guide explains how to upgrade SecureTransport on UNIX-based platforms and Microsoft Windows.
- *ST Web Client Configuration Guide* – This guide describes how to configure and customize the ST Web Client user interface.
- *ST Web Client User Guide* – This guide describes how to use the ST Web Client.

Go to Axway Support at support.axway.com to view or download documentation. The website requires login credentials and is for customers with active support contracts.

Get more help

Go to Axway Support at support.axway.com to get technical support, download software, documentation and knowledgbase articles. The website requires login credentials and is for customers with active support contracts.

The following support services are available:

- Official documentation
- Product downloads, service packs, and patches
- Information about supported platforms
- Knowledgebase articles
- Access to your cases

When you contact Axway Support with a problem, be prepared to provide the following information for more efficient service:

- Product version and build number
- Database type and version
- Operating system type and version
- Service packs and patches applied
- Description of the sequence of actions and events that led to the problem
- Symptoms of the problem
- Text of any error or warning messages
- Description of any attempts you have made to fix the problem and the results

Training

Axway offers training across the globe, including on-site instructor-led classes and self-paced online learning. For details, go to training.axway.com

Introduction 1

Axway SecureTransport is part of the Axway family of managed file transfer (MFT) products. SecureTransport allows organizations to adeptly control and manage the transfer of files inside and outside of the corporate firewall in support of mission-critical business processes, while satisfying policy and regulatory compliance requirements. SecureTransport serves as a hub and router for moving files between humans, systems and more. SecureTransport also completes tasks related to moving files (push or pull), hosting files in mailboxes or "FTP-like" folders, and provides portal access with configurable workflow for file handling and routing. SecureTransport delivers user friendly governance and configuration capabilities, including delegated administration and predefined and configurable workflows, while providing the highest possible level of security.

Transfer sites 2

This section describes the capability of SecureTransport Server to use custom protocol connectors for executing server initiated transfers and provides instructions on how to implement and configure a Custom Connector.

Starting with SecureTransport 5.3.3, during the installation or upgrade of SecureTransport 5.3.3 or above, one new folder is created:

- `${FILEDRIVE_HOME}/plugins/transferSites` - for custom sites

This folder contains a jar file for each Pluggable Transfer Site that is implemented and configured in SecureTransport. The jar file contains all the information required for the site including the specific configuration metadata. For additional information, refer to *What is a Pluggable Transfer Site* on page 11.

What is a Transfer site?

A transfer site is a location such as a local folder or a protocol server used when sending or receiving files during a server-initiated transfer or for handling files arriving over AS2. Transfer sites are specified and managed on a per-account basis.

When defining a transfer site as part of an account, you need to provide the information required for connecting to the site, authenticating (when applicable) and sending and/or receiving files. The selected transfer protocol dictates what information is required to define the transfer site.

In general, a site is used for AS2 transfers or server-initiated transfers. In the case of a server-initiated transfer, a file is either uploaded to the site when a subscription processes an uploaded file or downloaded from the site using a scheduler in a subscription.

What is a Pluggable Transfer Site

Beginning with SecureTransport 5.3.3, the Pluggable Transfer Site Service Provider Interface (SPI) has been exposed to allow developers to implement custom transfer sites for sending and receiving files using SecureTransport server-initiated transfers. This capability allows customer specific custom protocols to be implemented and used for SecureTransport file transfers.

Generally, a Pluggable Transfer Site is a custom connector that can be plugged into SecureTransport Server and used like a built-in SecureTransport Transfer Site. The SecureTransport Pluggable Transfer Site SPI is a set of Java interfaces and services that can be used to create, configure, and register custom protocol connectors to a SecureTransport Server.

Note The version of the Pluggable Transfer Site SPI described in this SecureTransport Developer's Guide is 1.2.0-4.

When creating a custom connector that can be used by SecureTransport for transferring files, you should generally do the following steps:

- *Implement a user interface* on page 12
- *Implement a custom protocol connector* on page 16
- *Deploy a custom protocol connector* on page 33

Note Take notice of the following Pluggable Transfer Site *limitation*: the Send To Partner step in Advanced Routing cannot overwrite the upload folder of a Pluggable Transfer Site.

The following sections will guide you through these steps to create and deploy a Custom Connector in SecureTransport Server. All you need to implement a Custom Transfer Site is the Pluggable Transfer Site SPI. An example implementation of a custom connector using the FTP protocol will be used to illustrate how to implement a SecureTransport Pluggable Transfer Site SPI.

Implement a user interface

SecureTransport Transfer Sites can be created and modified through the SecureTransport Administration Tool and SecureTransport REST API. In order to create a Pluggable Transfer Site that can be saved and modified by SecureTransport administrators, you must first implement the `com.axway.st.plugins.site.UIBean` interface from the Pluggable Transfer Site SPI.

The UIBean implementation must be a simple Java bean that describes the custom connector properties which will be saved into the SecureTransport database. These properties describe the connector and depending on the protocol; they can contain host, port, username, upload folder and so on. They will be visualized in the Administration Tool when an administrator creates an instance of the Pluggable Transfer Site.

Below is a an example implementation, called `FtpBean` which demonstrates the creation of an FTP Pluggable Transfer Site.

```
public class FtpBean implements UIBean {
    /** Holds the partner hostname. */
    private String mPartnerHost;
    /** Holds the partner port. */
    private Integer mPartnerPort;
    /** Holds the login name. */
    private String mPartnerUserName;
    /** The user's password. */
    private String mUserPassword;
    /** The remote folder to download files from. */
    private String mPartnerDownloadFolder;
    /** The download pattern. */
    private String mPartnerDownloadPattern;
    /** The remote folder to push files to. */
    private String mPartnerUploadFolder;
    /** Holds the network zone name. */
    private String mNetworkZone;
    /** Whether Active or Passive mode should be used. */
    private boolean mIsPassiveMode;
```

```
    /** Whether to use FTP over secure connection. */
    private boolean mUseFtps;
    /** Holds the certificate id, of the private key to authenticate with. */
    private String mClientCertificateIdLocalCertificate;
    /** A flag indicating if sending file with different name is enabled. */
    private boolean mSendFileAsEnabled;
    /** The new file name to be used when sending the file. */
    private String mSendFileAs;
}
```

Note Special symbols like periods (.) or dashes (–) are not allowed in pluggable transfer site property names.

You can use bean validation annotations from the `javax.validation.constraints.*` package (part of Java Bean Validation 1.0 (JSR 303)), if you would like to validate any of the properties defined in the UIBean implementation. For example, you can add the following annotations to validate the `PartnerHost`, `PartnerPort` and `PartnerUsername` properties for incorrect values:

```
/** Holds the partner hostname. */
    @NotNull(message="Host could not be null")
    @Size(min=1, message="Host could not be empty")
    @Pattern(regexp="\\S*", message="Host could not contain whitespaces.")
    private String mPartnerHost;

/** Holds the partner port. */
    @NotNull(message="Port could not be null")
    @Min(value=0, message="Port should not be a negative integer.")
    @Max(value=65535, message="Invalid port number. It should be less or equal
to 65535.")
    private Integer mPartnerPort;

    /** Holds the login name. */
    @NotNull(message="Username could not be null")
    @Size(min=1, message="Username could not be empty")
    @Pattern(regexp="\\S*", message="Username could not contain whitespaces.")
    private String mPartnerUserName;
```

The UIBean implementation must define accessor methods (get and set methods) for every field, so the property can be retrieved and modified. Also, the Pluggable Transfer Site SPI provides `com.axway.st.plugins.site.Encrypted` annotation which indicates that a specific property should be encrypted. This annotation needs to be added to both accessors of the property to indicate that this specific property needs to be encrypted. For example, if you have already defined the bean property `UserPassword` (`private String mUserPassword;`), you must define accessor methods annotated with the encrypted annotation so that the `mUserPassword` is encrypted when stored in the database:

```
    @Encrypted
    public String getUserPassword() {
      return mUserPassword;
```

```
    }
    @Encrypted
    public void setUserPassword(String password) {
      mUserPassword = password;
    }
```

To associate a custom site with a certificate, the certificate custom property should be named with a `LocalCertificate`, `PartnerCertificate` or `LoginCertificate` suffix depending on the certificate usage.

The following is an example of certificate custom property added to the `FtpBean` implementation, which holds login certificate id.

```
/** Holds the certificate id of the private key to authenticate with.*/
private String mClientCertificateIdLocalCertificate;
```

The next step of creating the user interface of the custom connector is to define the HTML file that will be loaded by SecureTransport Administration Tool on the *Transfer Sites* configuration page. The HTML file allows the administrator to create and save custom connectors to the database.

The HTML file should display UIBean properties and should provide JavaScript functions for loading and saving these properties to the database. The JavaScript functions could be defined as a script inside the HTML file or in a separate JavaScript file. An example HTML file can be found in the example implementation of the Custom connector included in SecureTransport Software Development Kit.

You need to follow specific conventions when writing your JavaScript code to enable SecureTransport administrators to create and save your Custom Transfer Site using the SecureTransport Administration Tool. Your JavaScript code must define the specific functions that SecureTransport will call to save and load your custom site properties.

First of all, you must use the `pluginRegister` object, which is the integration point between the SecureTransport Administration tool and your JavaScript code. The `pluginRegister` object is a special object that is exposed by SecureTransport allowing you to register a custom component that implements predefined load and save functions.

For instance, your JavaScript code should look like:

```
pluginRegister.setPluginComponent(new CustomPluginSite());
function CustomPluginSite() {
    function load(accountName, siteId) {
        // implement load custom site properties
    }
    function save(siteData, callback) {
        // implement save custom site properties
    }
    var instance = new PluginComponentInterface();
    instance.load = load;
    instance.save = save;
    return Object.seal(instance);
```

When implementing the `load` function, you must use the SecureTransport REST API to retrieve:

- Custom transfer site data from the database
- Available network zones
- Available certificates (can be used to authenticate to the remote partner)

Note When a custom transfer site is updated with a change in the transfer protocol, the correct default data must be retrieved. Check if the loaded transfer site is with the same type of transfer protocol. Default values like the network zones list must be retrieved even when a change in the protocol is made.

```
if (siteId) {
    $.ajax({
    url : "/api/v1.4/sites/" + siteId,
    type: 'GET',
    dataType : 'json',
    cache: false,
    success : function(data) {
        if (data && data['protocol'] === 'myftp') {
            //check if the loaded transfer site is of custom TS type, otherwise
load default data
            // load custom data for this transfer site
        } else {
            // load default data
        }
    }
});
} else {
```

The custom transfer site is returned by SecureTransport REST API as a key-value map containing all custom site properties as described in the UIBean implementation. After retrieving these properties, they should be populated and displayed in the custom HTML file.

When implementing the `save` function, you must collect all properties filled by the administrator in the custom HTML file and use a POST REST request to save the properties into the SecureTransport database. When you save custom properties as key-value pairs using POST REST request, the keys should be named exactly the same as the properties of the class that implemented the UIBean interface. If you want to implement validation for the site properties, you should add the validation in the save function. In order to propagate any validation errors, you should use the callback function with the desired error message as an argument:

```
callback("Custom validation error");
```

If the save action is successful and there are no errors, the last action of the save function should be to call the function without arguments:

```
callback();
```

SecureTransport REST API resources

In order to save the custom transfer site, you should initiate an authorized POST REST request to the following resource:

```
/api/v1.4/sites
```

The request body must look like:

```
{"sites": [<key-value pairs map>]}
```

In order to load the custom site properties, you should initiate an authorized GET REST request to the following resource:

```
/api/v1.4/sites/<siteId>
```

In case of a successful call, the function will receive a data map with the key - value pairs of the name and the value of the properties for the site.

You can retrieve available certificates by sending a GET REST request to the following resource:

```
/api/v1.4/certificates
```

Available network zones can be retrieved by GET REST request to the following resource:

```
/api/v1.4/zones
```

For more details about implementing the User Interface for the Custom Transfer Site, review the examples included in SecureTransport Software Development Kit.

Implement a custom protocol connector

The SecureTransport Pluggable Transfer Site SPI provides the `com.axway.st.plugins.site.Connection` interface that should be implemented in order to transfer files to and from the partner. The Connection interface version 1.0 declares the following methods:

- `void getFile(DestinationFile file) throws IOException;`
- `void putFile(SourceFile file) throws IOException;`
- `void finalizeExecution() throws IOException;`
- `List<FileItem> list() throws IOException;`

The Connection interface version 1.1 adds the following methods:

```
List<String> getProtocolCommands();
String getAdditionalInfo();
FlowAttributesData getFlowAttributesData();
```

The Connection interface is extended by `com.axway.st.plugins.site.CustomSite` abstract class which is also part of Pluggable Transfer Site SPI. Its main purpose is to hold a UIBean instance containing connection parameters as key-value pairs saved into the database as custom transfer site properties. The parameters are loaded from the database by SecureTransport and populated in the UIBean instance inside the specific `CustomSite` implementation when starting server-initiated transfers.

The following steps should be followed to create the custom protocol connector:

1. Define a class that extends the `com.axway.st.plugins.site.CustomSite`:

```
import com.axway.st.plugins.site.CustomSite
public class FTPSite extends CustomSite {
}
```

2. Define the UIBean instance that will hold the connector parameters:

```
/** The UIBean implementation. */
private FtpBean mFtpBean = new FtpBean();
```

3. Define the default constructor for `FTPSite` class which should set the instantiated `FtpBean` as the UIBean instance in the `CustomSite` parent class.

   When SecureTransport starts the transfer, it will populate the connection properties into UIBean instance defined in the `CustomSite` class:

```
public FTPSite() {
      setUIBean(mFtpBean);
}
```

In addition there is a way to report the download information. This information consist of two attributes: *remote download folder* and *remote download pattern* via the following annotations, that are part of site SPI: `RemoteDownloadFolder` and `RemoteDownloadPattern`.

- `RemoteDownloadFolder` - During execution of the `list()` method, this attribute can be used to distinguish the remote download folder, if any.
- `RemoteDownloadPattern` - During execution of the `list()` method, this attribute can be used to distinguish the remote download pattern, if any.

Note The annotated field must be of type String. Also, this annotation take effect only when is used in implementation instance of `CustomSite`.

The reported information will be used for correct Sentinel reporting during pull actions.

Site SPI versioning

Each Site SPI is versioned. The initial SPI release is version 1.0. Every version extends the previous one and adds additional content. The SPI methods and classes added in a specific version have the **@since** annotation in their Javadoc documentation.

To declare what SPI version is used in Site implementation, place the `Plugin-Info.yaml` file inside the `Meta-Inf` directory. This file contains the plug-in information, similar to the `MANIFEST.MF` file in Site SPI version 1.0. The `Plugin-Info.yaml` file is introduced in Site SPI version 1.1. The old `MANIFEST.MF` file (if any) is with lower priority when there is a `Plugin-Info.yaml` file.

If the version is not specified, the default one (1.0) will be used.

Only the methods and classes available in the specified version can be used.

The latest SPI version is **1.7**.

Connection interface implementation - version 1.0

SecureTransport calls `putFile` (`SourceFile` file) method from the Connection interface when executing server-initiated push. The `SourceFile` instance will contain the `InputStream` to read the local file content that will be sent, the remote file name to write the content to, and the size of the local file (number of bytes the `InputStream` could read). Generally, the implementation of `putFile()` method should connect to the remote partner and write the content from the local file input stream to the remote file. The parameters for the connection can be retrieved from the already defined UIBean instance.

When calling `getInputStream(RemotePartner descriptor)` from the `SourceFile` instance, you must pass a `com.axway.st.plugins.site.RemotePartner` instance as argument. This instance should hold the connection parameters which will be reported by SecureTransport File Tracking. Currently, these parameters are the remote host, the remote folder to which the file will be pushed, the parameter that shows whether the remote connection is secure and the new remote file name which should be reported as Post Transmission Action if such is configured in transfer site. In SPI v1.3 two additional parameters were introduced – one for identifying the network connection port and one for identifying the remote impersonated entity.

Let's assume you have an `FTPConnector` interface and a simple FTP client implementation based on your interface which has the following methods:

- `connect(FtpBean ftpBean)`
- `upload(InputStream inputStream, String remoteDir, String remoteFile)`

The `connect()` method retrieves connection parameters from the `FtpBean` instance and uses them to configure the FTP client and then connects to the remote partner.

The `upload()` method sends the content from the `inputStream` to the remote partner and saves it to `remoteFile` in the `remoteDir`.

Note The `remoteDir` is actually the remote upload folder to which the file will be sent. The Send To Partner step in the Advanced Routing cannot overwrite the upload folder of a Custom Transfer Site.

The Send to Partner step has option to override the upload folder configured in the transfer site only if the transfer site explicitly allows upload folder overriding which is configurable in all built-in transfer sites. For all custom transfer sites, the overriding of the upload folder by the Send to Partner step **is not** allowed and **is not** configurable. If the Send to Partner step is configured to override the upload folder of custom transfer site, the overriding is ignored.

So, in the `putFile` method implementation inside the `FTPSite` class, you should connect to the remote partner and then upload the file.

Note Do not disconnect from the remote partner.

First, you should use the `connect()` method to connect to the FTP server:

```
/** The FTP client implementation. Allows connecting/disconnecting/transferring
files. */
private AbstractFTPConnector mFtpConnection;

/** Connects to a FTP server.
 *
 * @throws IOException on error
 */
public String connect() throws IOException {
    mFtpConnection = new FTPConnectorBuilder().build(mFtpBean);
    mFtpConnection.setCertificateService(mCertificateService);
    mFtpConnection.setProxyService(mProxyService);
    return mFtpConnection.connect();
}
```

Then use it in the `putFile()` method to upload the file:

```
@Override
public void putFile(SourceFile file) throws IOException {

    String resolvedHost = "127.0.0.1";
    if (mFtpConnection == null) {
        resolvedHost = connect();
    }

    RemotePartner descriptor = new RemotePartner(resolvedHost,
    mFtpBean.getPartnerUploadFolder(),
            mFtpBean.isFtps(), getSentFileAs());

    mFtpConnection.upload(file.getInputStream(descriptor),
    descriptor.getRemoteHost(), file.getName());
}
/**
 * Gets the "Send File As" value or
 * return empty string if "Send File As" option is not enabled.
 *
 * @return "Send File As" value or, if no value is found it returns
   empty string
 */
private String getSentFileAs() {
    String resultFileAs = "";
    if (mFtpBean.isSendFileAsEnabled()) {
        resultFileAs = mFtpBean.getSendFileAs();
        }
    return resultFileAs;
}
```

List() method

The `list()` method implementation from the Connection interface should return the list of files that are going to be downloaded from the remote partner". These files should be described using the `com.axway.st.plugins.site.FileItem` class. This class holds the remote file name that should be downloaded and the name which will be used to save the downloaded files.

When SecureTransport starts the server-initiated pull transfer, the `list()` method from the Connection interface is always called before the file pull. The parameters for configuring file pull are `remoteDir` and `remotePattern` where the pattern could be a filename or expression. After the `list()` method returns the list of files that should be downloaded, described using the `FileItem` class, SecureTransport calls the `getFile()` method from Connection interface for every item in the list. In the `list()` method, you can implement logic for listing files in a predefined remote directory only, or recursively listing all files in a specific directory. It is important to remember that the file names (and paths in case of recursive listing) returned by the `list()` method should be relative to the predefined remote directory.

In the following example, the `FtpBean` and `FtpConnector` will be used to implement the `list()` method:

Let's assume you have an `FTPConnector` interface and a simple FTP client implementation based on your interface which has the following methods:

- `connect(FtpBean ftpBean)`
- `listFiles(String remoteDir, String remotePattern)`

The `connect()` method will retrieve connection parameters from the `FtpBean` instance and use them to configure the FTP client and then connect to the remote partner.

The `listFiles()` method returns the list of file names matching the `remotePattern` inside the `remoteDir` folder.

In the `list()` method implementation you must first connect to the remote partner, then execute the `listFiles()` method to get the list of file names that match the specific pattern and then construct the result list of the `FileItem` instances.

First, you should use the `connect()` method to connect to the FTP server:

```
/** The FTP client implementation. Allows connecting/disconnecting/transferring
files. */
private AbstractFTPConnector mFtpConnection;

/** Connects to a FTP server.
 *
 * @throws IOException on error
 */
public String connect() throws IOException {
    mFtpConnection = new FTPConnectorBuilder().build(mFtpBean);
    mFtpConnection.setCertificateService(mCertificateService);
    mFtpConnection.setProxyService(mProxyService);
    return mFtpConnection.connect();
}
```

Then use it to list the files from the remote partner that match the specified criteria (remote folder and file name pattern) defined in the `FtpBean` instance.

Note Do not disconnect from the external server.

```
@Override
public List<FileItem> list() throws IOException {
    if (mFtpConnection == null) {
        connect();
    }

    List<String> names = mFtpConnection.listFiles
(mFtpBean.getPartnerDownloadFolder(),
            mFtpBean.getPartnerDownloadPattern());
    List<FileItem> result = new ArrayList<FileItem>();
    if (names != null && names.size() > 0) {
        for (String name : names) {
            result.add(new FileItem(name));
        }
    }
    return result;
}
```

Note that in the example above, the file will be saved locally with the same name as the name of the remote file when pulled.

If you want to save the file with different name, you can use the other constructor from the `FileItem` class when creating the `FileItem` instance the files:

```
new FileItem(String fileName, String receiveFileAs);
```

getFile(DestinationFile file) method

The next method that should be implemented is `getFile(DestinationFile file)`.

SecureTransport calls the `getFile(DestinationFile file)` method from the Connection interface when executing a server-initiated pull. The `DestinationFile` instance will contain the `OutputStream` to write the remote file content to and the remote file name which will be pulled. Generally, the `getFile()` method should write the content of the remote file to the output stream held by the `DestinationFile` instance. When calling the `getOutputStream(RemotePartner descriptor);` from `DestinationFile` instance, you must pass as argument the `com.axway.st.plugins.site.RemotePartner` instance. This instance should hold the connection parameters which will be reported by SecureTransport File Tracking. Currently, these parameters are the remote host, remote folder from which the file will be downloaded and the parameter that shows whether the remote connection is secure. In SPI v1.3 two additional parameters were introduced – one for identifying the network connection port and one for identifying the remote impersonated entity.

Let's assume you have an `FTPConnector` interface and a simple FTP client implementation based on your interface which has the following methods:

- `connect(FtpBean ftpBean)`
- `download(OutputStream outputStream, String remoteDir, String remoteFile)`

The `connect()` method retrieves the connection parameters from the `FtpBean` instance and uses them to configure the FTP client and then connects to the remote partner.

The `download()` method retrieves the content from the `remoteFile` which is in the `remoteDir` and writes its content to the `outputStream` instance.

So, in the `getFile` method implementation inside the `FTPSite` class, you should first connect to the remote partner and then download the file.

First, you should use the `connect()` method to connect to the FTP server:

```
/** The FTP client implementation. Allows connecting/disconnecting/transferring
files. */
private AbstractFTPConnector mFtpConnection;

/** Connects to a FTP server.
 *
 * @throws IOException on error
 */
public String connect() throws IOException {
    mFtpConnection = new FTPConnectorBuilder().build(mFtpBean);
    mFtpConnection.setCertificateService(mCertificateService);
    mFtpConnection.setProxyService(mProxyService);
    return mFtpConnection.connect();
}
```

Then use it in the `getFile()` method to download the file.

Note Do not disconnect from the external server.

```
@Override
public void getFile(DestinationFile file) throws IOException {
    String resolvedHost = "127.0.0.1";
    if (mFtpConnection == null) {
        resolvedHost = connect();
    }
    mFtpConnection
        .download(
            file.getOutputStream(
                    new RemotePartner(resolvedHost,
               mFtpBean.getPartnerDownloadFolder(), mFtpBean.isFtps())),
                mFtpBean.getPartnerDownloadFolder(), file.getName());
}
```

public void finalizeExecution() throws IOException; method

The last method from Connection interface which you should implement is:

```
public void finalizeExecution() throws IOException;
```

This method is called by SecureTransport when finalizing the server-initiated transfer after executing the `list` method, `getFile` method, and `putFile` method regardless if they are completed successfully or not.

In this method you should release any occupied resource, for instance disconnecting the specific client from the remote partner.

Note If the external server requires an explicit logout, it needs to be implemented either here or in the disconnect implementation.

```
@Override
public void finalizeExecution() throws IOException {
    if (mFtpConnection != null) {
        mFtpConnection.disconnect();
        mFtpConnection = null;
    }
}
```

Connection interface implementation version 1.1

Connection interface version 1.1 includes all functionality, introduced in version 1.0 and adds the following methods:

public List<String> getProtocolCommands(); method

This method is called by SecureTransport after the Server-initiated transfer (SIT) ends. The purpose of this method is to return the protocol commands logged (or captured) by the client during the transfer process. This method must be used along with `CommandLoggingService`, and more correctly method `getProtocolCommands()`.

For more information, see *Command Logging Service* on page 29.

SecureTransport logs the protocol commands as normal protocol commands for the current transfer.

Note The restrictions for custom logged protocol commands are the same as the ones logged by SecureTransport.

```
@Override

public List<String> getProtocolCommands() {
    return mCommandLoggingService.getProtocolCommands();
}
```

public String getAdditionalInfo(); method

You can use this method to log any additional information about the transfer. This method is called on Server-initiated transfers end. The logged information is shown in SecureTransport File Tracking page under the **Additional Information** section. This method must be used along with `AdditionalInfoLoggingService`, and more correctly method `getAdditionalInfo()`.

For more information, see *Additional Info Logging Service* on page 29.

Note The total size of the message that can be logged should not exceed 4096 characters. If bigger, the exceeding part will be truncated.

```
@Override
public String getAdditionalInfo() {
    return mAdditionalInfoLogService.getAdditionalInfo();
}
```

public FlowAttributesData getFlowAttributesData(); method

Gets the flow attributes for the currently transferred file. This serves two purposes. First it is populated with existing flow attributes for the file being transferred by the server. Called second time when transfer is finished to persist the attributes stored in it.

For more information about flow attributes, see the SecureTransport Administrator's guide, Mail template commands and variables section Flow attributes subsection.

Connection interface implementation version 1.3

Connection interface version 1.3 includes all functionality, introduced in version 1.1. It also provides:

- extended `RemotePartner` class with two additional properties and a builder utility
- ability to notify SecureTransport for executed post-transmission actions (PTAs) to report them in SecureTransport File Tracking.

integer RemotePartner#remotePort; field

This object variable is meant to contain the partner's port, used for the remote connection.

string RemotePartner#remoteContainer; field

This object variable is meant to identify the remote impersonated entity. Could be a user, business unit, email address, group, resource, etc.

static class RemotePartner.Builder; utility

This utility is meant to construct RemotePartner objects using any set of RemotePartner properties. Typical usage:

```
RemotePartner remotePartner = new RemotePartner.Builder()
     .host("192.168.10.2")
     .port(21)
     .container("user2")
     .folder("/download")
     .secureConnection(true)
     .build();
```

void CustomSite#notifyPTAExecuted(PTAInfo); utility method

This method should be called by CustomSite implementations after a PTA is executed to report it in SecureTransport File Tracking. Post-transmission actions are usually executed after an actual transfer, so this method could be effectively called after calling `SourceFile#getInputStream(…)` or `DestinationFile#getOutputStream(…)` in case of upload or download respectfully.

class PTAInfo; bean

Objects of type `PTAInfo` describe an executed post-transmission actions. The object fields are:

- `String actionType` – the type of the executed operation; required; typical actions are: MOVE, DELETE, SHARE, etc.; types, longer than 20 chars, are being trimmed by SecureTransport
- `PTACompletionStatus completionStatus – SUCCESS/FAILURE;` required; indicates if the execution of the PTA succeeded or failed
- `String result` – the product of the executed action; typically, a file-name
- `String comment` – human-readable information about the executed action; comments, longer than 1024 bytes are being trimmed by SecureTransport

Typical usage:

```
notifyPTAExecuted(
   new PTAInfo(
       "MOVE",
       PTACompletionStatus.SUCCESS,
       "/download/success/file.txt",
       "File 'file.txt' has been successfully moved to
'/download/success/'."));
```

Connection interface implementation version 1.4

Connection interface version 1.4 includes all functionality, introduced in version 1.3. It also provides:

- Extended proxy functionality, where SecureTransport can deny or blacklist proxies

ProxyInfo getProxyInfo(); method

This method is meant to hold and return the proxy instance (if any) used when performing Server-initiated transfers. SecureTransport needs it to be able to determine which proxy to deny or blacklist if transfers via this proxy are failing often in a predefined period of time. If a proxy is not used, this method can simply return null. The `ProxyInfo` object can be easily obtained using the Proxy Service.

Connection interface implementation version 1.7

Connection interface version 1.7 includes all functionality, introduced in version 1.6. It also provides:

- New `SSLContextService` which constructs `SSLContext`, `SSLSocketFactory` objects, to be used when connecting over secure connection to a remote partner.

Exception handling

SecureTransport will only retry the transfer by default if some of the Connection interface methods throw an `IOException` instance if there are retry attempts remaining. The SecureTransport Pluggable Transfer Site SPI provides a special exception named `TransferFailedException`, which is of type `java.io.IOException`. This exception indicates a transfer failure; if the error is permanent (for example, an authentication failure) and is not expect to be resolved on retry, an `isPermanentFailure` flag should be raised. Otherwise, it should be left set to false, (for example, on connection timeout or any other transient error), allowing the transfer to succeed on the next retry. Generally, any exception caused by configuration or miss configuration is an exception that should not be retried. If an error occurs while implementing your Custom Connector, a `TransferFailedException` instance should be thrown with the flag set to indicate if the exception is permanent or temporary.

There are two available constructors for instantiating `com.axway.st.plugins.site.TransferFailedException`:

- `public TransferFailedException(String message, Throwable cause);`
- `public TransferFailedException(String message, Throwable cause, boolean isPermanentFailure);`

The first one will set the `isPermanetFailure` flag to false and will force the SecureTransport to retry the transfer if retry attempts are available. The number of retries and the interval between retries for a transient failure of transfer can be configured by a SecureTransport administrator. For more information about configuring retry parameters for server-initiated transfers, refer to the *SecureTransport Administrator's Guide*.

Note Only adminstrator configurable retries should be attempted. Internal retries should not be attempted. Internal retries are hidden from SecureTransport and no file tracking information will be generated.

SecureTransport exposed services

The SecureTransport Pluggable Transfer Site SPI also exposes services from SecureTransport Server that can be consumed by the Custom Connector implementation. The implementation can `@Inject` the service interface in its extension of `CustomSite` class and then consume the service via provided interface.

Certificate Service

To use the certificate service, declare a variable with `@Inject` annotation (`javax.inject.Inject`) to the interface of the certificate service provided in the API:

```
@Inject
private CertificateService certificateService;
```

The certificate service provides capabilities for:

- `CertificateStatus getCertificateStatus(X509Certificate certificate);`
- `Subject getCertificateSubject(X509Certificate certificate) throws CertificateException;`
- `KeyPair getKeyPairByAlias(String certificateAlias) throws CertificateException;`
- `KeyPair getKeyPairById(String certId) throws SecurityException;`
- `X509Certificate getCertificateByAlias(String certificateAlias) throws CertificateException;`
- `X509Certificate getCertificateById(String certificateId) throws SecurityException;`
- `X509Certificate getIssuer(X509Certificate certificate) throws CertificateException;`
- `List<X509Certificate> getCertificateChain(X509Certificate certificate) throws CertificateException;`
- `Collection<X509Certificate> getCertificates(String subjectDN) throws CertificateException;`

Proxy Service

To use the proxy service, declare a variable with `@Inject` annotation to the interface of the proxy service provided in the API:

```
@Inject
private ProxyService proxyService;
```

The proxy service provides capabilities for:

- `ProxyInfo getProxy(ProxyType proxyProtocol, String dmzName) throws NoSuchZoneException;`
- `boolean isRemoteDnsResolutionEnabled(String zoneName) throws NoSuchZoneException;`
- `String getPublicURLPrefix(String zoneName) throws NoSuchZoneException`

The `getProxy` method of the proxy service will return a "null" object, if there is no proxy with the specified type defined in the selected DMZ zone. The specific custom connector implementation decides whether to use this proxy for the transfer or not.

The default SecureTransport zone is **Private**.

SSL Context Service

To use the SSL context service, declare a variable with @Inject annotation (javax.inject.Inject) to the interface of the certificate service provided in the API:

```
@Inject
private SSLContextService sslContextService;
```

The SSL context service provides capabilities for:

- `SSLContext createSSLContext(String certId, boolean verifyPeer) throws SecurityException;`
- `SSLContext createSSLContext(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm) throws SecurityException;`
- `SSLSocketFactory configSSLContextFactory(SSLContext sslContext, String[] protocols, String[] cipherSuites);`
- `SSLSocketFactory configSSLContextFactory(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm, String[] protocols, String[] cipherSuites) throws SecurityException;`

Logging Service

You can use this service to log messages on different levels in the SecureTransport Server Log.

The log levels are DEBUG, INFO, WARN, ERROR and FATAL.

Every level has two methods: one with a logged message only and another with a logged message with an exception.

For more information, see the documentation for the associated methods.

The logged message appears in the SecureTransport Server Log. It starts with the following pattern `[TRANSFER_SITE]:[plugin_name OR protocol_name].`

For more information about the third-party libraries logging, see Third-party libraries logging.

Additional Info Logging Service

To use the additional information logging service, declare a variable with an `@Inject` annotation to the interface of the additional information logging service provided in the API:

```
@Inject
private AdditionalInfoLoggingService
mAdditionalInfoLogService;
```

The additional information logging service capabilities are:

`void append(String info)` - adds/appends an info message. Every appended message is delimited by the particular system line separator.

`String getAdditionalInfo();` - gets the additional information.

You can add additional information by calling the `append` method with the information you want to be logged. You can call the `append` method multiple times. The different messages will be separated by a new line.

Command Logging Service

The service provides protocol commands logging capabilities in SecureTransport.

To use the command logging service, declare a variable with `@Inject` annotation to the interface of the command logging service provided in the API:

```
@Inject
private CommandLoggingService mCommandLoggingService;
```

The basic capabilities of this service are:

`void append(String command);` – appends the command to the buffer of commands.

`List<String>getProtocolCommands();` – gets the protocol commands.

You can use the `append` method to log a single protocol command. You can call the append method multiple times. The logged protocol commands appear in SecureTransport Transfer Log as an ordinary protocol command.

Flow Attributes Data service

FlowAttributesData

Serves as a container for flow attributes. This container can be used for reading/writing flow attributes for the currently transferred file.

To use the flow attributes container, declare a variable with `@Inject` annotation to the interface of the flow attributes data service provided in the API:

```
@Inject
public FlowAttributesData mFlowAttributesData;
```

The `FlowAttributesData` object has the following capabilities:

- `Map<String, Object> getFlowAttributes();` - gets the flow attributes populated by the server.

  Return the key-value pairs of flow attributes for the currently transferred file.

- **void** `setFlowAttributes(Map<String, Object> flowAttributes);` - sets the flow attributes to be persisted by the server.

  Sets the flow attributes for the currently transferred file. If the attribute already exists, it will be overwritten.

Expression Evaluator Service

You can use expression evaluator service to evaluate and validate the expressions used in site connection implementation.

To use the expression evaluator service, declare a variable with `@Inject` annotation to the interface of the expression evaluator service provided in the API:

```
@Inject
privateExpressionEvaluatorServicemExpressionEvaluatorService;
```

The expression evaluator service has the following capabilities:

- `void  loadExpressionService(Map<String, Object> context);` - loads the expression service with an appropriate context.
- `String evaluateString(String expression);` - evaluates the Expression and returns the string result.

  This is a convenience method that will simply evaluate the expression with a return class of *String.class*. This is by far the most common usage of the `evaluateString` method.

- `public <T> T evaluateObject(String expression, Class<T> expectedType)` **`throws`** `InvalidExpressionException;`: evaluates expression to specific type of object. If the expression is not valid, an `InvalidExpressionException` will be thrown.
- `void validateExpression(String expression)` **`throws`** `InvalidExpressionException;`: validate the given expression. If the expression is not valid, an `InvalidExpressionException` will be thrown.
- `public String evaluatePathString(String expression)` **`throws`** `InvalidExpressionException;`: evaluates the expression and return the string result.

According to the JSR specification for Expression Language symbol '\' escapes both `#{}` and `${}` expressions. So, in Windows, all '\' are replaced by '/' as we might have problems with such paths: `c:\smth\${smth}`.

After evaluation symbols '\' are returned. Expression escaping is not supported. If the expression is not valid an `InvalidExpressionException` will be thrown.

If you want to load in expression evaluator the evaluation context, use the `loadExpressionService` method and pass the evaluation context. This context will be accessible using the following variable `${plugin['<key>']}` or `${plugin.<key>}`, where `<key>` is existing key from the provided evaluation context. If `<key>` is one word, use `${plugin.<key>}` . Otherwise, use `${plugin['<key>']}`.

You can use a collection of already built-in functions in expression evaluator. For a full list with examples about the built-in functions, see Expression language support.

Note The expression evaluator service is not applicable for the Account Templates sites.

Transfer Attributes Data service

TransferAttributesData

This service is available in SecureTransport Pluggable Transfer Site SPI version 1.6 and serves as a container for transfer attributes. This container can be used for reading transfer attributes for the currently transferred file.

The transfer attributes contain properties, submitted through the REST API when triggering a SIT PULL, as well as:

- `fulltarget` - the absolute path of the transferred file. When used in a route, it points to the file inside the sandbox folder.
- `route_source_full_target` - the absolute path of the transferred file when used in a route.

Note All transfer attributes submitted through the REST API are available with their actual name in lower case.

Note `fulltarget` and `route_source_full_target` are only available for SIT Push.

To use the transfer attributes container, declare a variable with `@Inject` annotation to the interface of the transfer attributes data service provided in the API:

```
@Inject
public TransferAttributesData mTransferAttributesData;
```

The `TransferAttributesData` object has the following capabilities:

`Map<String, Object> getTransferAttributes();` - gets the transfer attributes populated by the server.

Returns the key-value pairs of transfer attributes for the currently transferred file.

Account Attributes Data service

AccountAttributesData

This service is available in SecureTransport Pluggable Transfer Site SPI version 1.6 and serves as a container for account-related attributes. This container can be used for reading account-related attributes for the currently transferred file.

The account attributes contain the following properties:

- `account_name` – the name of account transferring the file
- `account_id` – ID of the account transferring the file
- `account_email` – email address of the account transferring the file
- `account_notes` – notes of the account transferring the file
- `account_phone` – phone number of the account transferring the file
- `account_type` – type of account transferring the file, e.g. user, template, service
- `homedir` – home folder of the account transferring the file
- `useruid` – UID of the account transferring the file
- `usergid` – GID of the account transferring the file
- `loginname` – the name of logged in user transferring the file
- `site_name` – the name of site executing the transfer
- `business_unit_name` – the name of business unit the account that performs the file transfer belongs to
- `userVars.<variable>` - additional attributes of the account transferring the file

To use the account attributes container, declare a variable with `@Inject` annotation to the interface of the account attributes data service provided in the API:

```
@Inject
```

```
public AccountAttributesData mAccountAttributesData;
```

The `AccountAttributesData` object has the following capabilities:

- `Map<String, Object> getAccountAttributes();` - gets the account attributes populated by the server.
- Return the key-value pairs of account attributes for the currently transferred file.

Deploy a custom protocol connector

In order to configure and deploy a Custom Connector implementation into SecureTransport Server, follow these steps:

1. After implementing Custom Connector and the User Interface for this connector you should add them to a single jar file. For Site SPI version 1.0, you must create a `MANIFEST.MF` file in the jar file that describes the integration points between SecureTransport and the Custom Connector.

   The `META-INF/MANIFEST.MF` file must be created with the following properties:

Property	Description
Custom-Protocol	The name of the protocol as displayed in SecureTransport File Tracking. File Tracking also logs the protocol. Should be unique. Maximum of 255 characters.
Protocol-Label	The human readable name that is displayed in the drop-box menu of the Pluggable Transfer Site. Maximum of 255 characters.
Custom-UI	The HTML file, which is injected, when the Pluggable Transfer Site is selected from the drop down menu on the Transfer Sites page.
Custom-Site	The full name of the class that extends the `com.axway.st.plugins.site.CustomSite` abstract class and implements the `com.axway.st.plugins.site.Connection` interface. This is the integration point between SecureTransport and the Custom Transfer Site.
Class-Path	This property is optional. The Class-Path property is useful for specifying the list of the external libraries your custom connector needs to successfully run. The value of this property must be a space or tab separated list of the external libraries.
SPI-Version	This property is optional. Specify the used SPI version by the plug-in. Default value is 1.0.

   This metadata file is used by SecureTransport to register and use the Custom Connector implementation like a regular Transfer Site.

If your Custom Connector implementation depends on third party libraries, you have two implementation approaches:

- You can use a fat jar file that stores your classes and the classes from all dependent jars into a one single jar file.

  or

- Add a Class-Path attribute to the `MANIFEST.MF` file, in which you should enumerate your dependent jars separated by spaces or tabs. SecureTransport will add the enumerated libraries to the Custom Connector implementation classpath.
- For Site SPI v1.1 or higher use instead the `Plugin-Info.yaml` file.
  The META-INF/Plugin-Info.yaml must be created with the same properties as described above.
  Example:

```
Custom-Protocol: myProtocol
SPI-Version: '1.0':
Custom-Site: com.axway.st.plugins.site.MyProtocol_1_0
Protocol-Label: My Sample Transfer Site (1.0)
Custom-UI: html/mySite-1-0.html
Class-Path: MySite/lib/lib1.jar MySite/lib/lib2.jar MySite/lib_1_0/lib3_1_
0.jar
'1.1':
Custom-Site: com.axway.st.plugins.site.MyProtocol_1_1
Protocol-Label: My Sample Transfer Site (1.1)
Custom-UI: html/mySite-1-1.html
Class-Path: MySite/lib/lib1.jar MySite/lib/lib2.jar MySite/lib_1_1/lib3_1_
1.jar
```

Note SecureTransport will look for the latest Site SPI version if more than one is specify.

2. Deploy Custom Connector into SecureTransport.

When you have produced/created a jar file containing the Custom Connector implementation and a `MANIFEST.MF` file, you should place it into your SecureTransport Server. In the installation folder of SecureTransport 5.3.6 version and above, you will find the following directory:

- `${FILEDRIVE_HOME}/plugins/transferSites`, where `${FILEDRIVE_HOME}` represents SecureTransport installation directory.

Note In a cluster environment, you must add the Custom Connector jar file to all nodes of the cluster.

The Custom Connector jar file should be placed in the `${FILEDRIVE_HOME}/plugins/transferSites` directory. This directory is scanned by SecureTransport for jar files with the specific `MANIFEST.MF` file describing the integration points between the custom connector and the SecureTransport Server. For each custom connector, there should be a jar file containing the SecureTransport Pluggable Transfer Site SPI implementation and the `MANIFEST.MF` file. Also, if there are any third party resources that are required by your custom connector implementation and are not included in the Custom Connector jar, they should be placed in a separate directory inside the `${FILEDRIVE_HOME}/plugins/transferSites` directory and be listed in the `MANIFEST.MF` file under Class-Path attribute.

Note External libraries should be placed in separate directory inside the `${FILEDRIVE_HOME}/plugins/transferSites` directory.

Let's assume you have already implemented the simple FTP Custom Connector with a third party dependencies to external library named `sftp.jar`. Then the `MANIFEST.MF` file should look like this:

```
Custom-Protocol: myftp
Protocol-Label: FTP(S) Sample Transfer Site
Custom-UI: ftpSampleSite.html
Custom-Site: com.axway.st.plugins.site.sdk.ftp.FTPSite
Class-Path: myftp/lib/sftp.jar
```

Add your Custom Connector implementation jar to the `${FILEDRIVE_HOME}/plugins/transferSites/` directory and the `sftp.jar` file to the `${FILEDRIVE_HOME}/plugins/transferSites/lib/myftp/lib/` directory inside the SecureTransport installation directory. The final directory structure should look like this:

```
${FILEDRIVE_HOME}/plugins/transferSites/myftp.jar
${FILEDRIVE_HOME}/plugins/transferSites/myftp/lib/sftp.jar
```

After your custom connector jar file, including the third party libraries, is placed in the `${FILEDRIVE_HOME}/plugins/transferSites` directory, the SecureTransport Administration Service should be restarted in order to load and register your Custom Connector implementation as a SecureTransport Transfer Site.

Note When you are deploying your Custom Connector implementation, you should only include third party dependencies used by your implementation. You should not add APIs provided run-time by SecureTransport. This includes the SecureTransport Pluggable Transfer Site SPI, Bean Validation API, and JSR-330 (`javax.inject`) API.

For more details about implementing a Custom Connector, review the examples included in SecureTransport Software Development Kit.

Third-party libraries logging

To enable the logging, produced by third-party libraries, add a logger for a specific package in `${FILEDRIVE_HOME}/conf/tm-log4j.xml`.

For example:

```
<logger name="com.amazonaws" additivity="false">
    <level value="debug" />
    <appender-ref ref="ServerLog" />
</logger>
```

Third-party libraries loggers don't need any additional inclusion of logging libraries like `SLF4J, Log4j or JUL (java.util.logging)`. They will be loaded if needed, like any existing library in SecureTransport (`${FILEDRIVE_HOME}/lib/jar`). All SecureTransport libraries will be loaded with priority in case of any duplicated libraries in plug-ins directory.

For more information about the deployment of custom site connectors, refer to *Deploy a custom protocol connector* on page 33.

Expression language support

Expression evaluator service will evaluate expressions that follow the supported expression language (EL). More detailed information about the Expression Language support in SecureTransport can be found in SecureTransport Administrator's Guide (*Appendix H: Expression Language*). The following sub-sections is applicable for the expression evaluator service:

1. Expression Language operators
2. Predefined variables
   - `${uploadFolderOverwrite}` – this variable will hold value from the Overwrite Upload Folder field in the Send To Partner step. If the value is not present, an empty string will be returned.
   - `${timestamp}` – the timestamp variable in Unix Epoch time
3. Predefined functions
4. Match and replace functions.

Enable Logger service

In order to enable logging when using the Logger service, edit the `com.axway.st.plugins` loggers in the `tm-log4j.xml` file. For example, with Pluggable Transfer Sites:

```
<logger name="com.axway.st.plugins.site" additivity="false">
        <level value="info" />
        <appender-ref ref="ServerLog" />
</logger>
```

For fine-tuning debug logging, add the transfer site name in conjunction with the `com.axway.st.plugins` logger.

In this case, use `com.axway.st.plugins.site.<transfer_site_name>` as shown on the example:

```
<logger name="com.axway.st.plugins.site.ts" additivity="false">
        <level value="debug" />
```

```
        <appender-ref ref="ServerLog" />
</logger>
```

SecureTransport REST APIs 3

The current version of REST API for SecureTransport 5.4 is version 1.4. It includes new resources and all existing resources (backward compatible), which are still available in the previous versions.

Administrative credentials

With administrator credentials, you can use the SecureTransport REST API to query, create, modify, and delete user accounts and related SecureTransport configuration information including applications, business units, certificates, sites, subscriptions, and transfer profiles. You can also use the REST API to perform the following actions:

- Query and kill FTP and HTTP sessions.
- Query and export logs.
- Query, create, modify, and delete network zones to configure the SecureTransport Edge protocol.

For more information about network zones, refer to the *SecureTransport Administrator's Guide*.

User credentials

With user credentials, you can use the SecureTransport REST API to perform the following actions:

- Perform and manage server-initiated transfers using your transfer sites.
- Upload files to your home folder.
- List your home folder.
- Change your password.
- Trigger an email from SecureTransport Server with instructions for resetting you password.

Base REST API URI

The base URI for the REST is `https://<host>:<port>/api/<version>/` where:

- *<host>* is the hostname or the IP address of the SecureTransport server.
- *<port>* is the one of the following:
  - Administrator resources – Administration Tool port (default 444).
  - User resources – HTTP or HTTPS port (default 80 or 443).

- *<version>* is the SecureTransport REST API version (vx.y where x is the major version and y is the minor version, v1.4 for SecureTransport 5.4).

Useful URIs

The following are useful REST API URIs:

- https://*<host>*:*<port>*/api/ – lists all available versions
- https://*<host>*:*<port>*/api/v*<1.4 or lower>*/ – resource descriptions
- https://*<host>*:*<port>*/api/v*<1.4 or lower>*/docs/index.html – API reference
- https://*<host>*:*<port>*/api/v1.4/ – GET request returns yaml or json Swagger specification:
  - Returns Swagger yaml specification with no "Accept:" HTTP header or with "Accept: application/x-yaml" header.
  - Returns Swagger json specification with "Accept: application/json" HTTP header.
- https://*<host>*:*<port>*/api/application.wadl – WADL application description
- https://*<host>*:*<port>*/api/v1.4/docs/index.html – Swagger-UI API reference

HTTP methods

The HTTP methods used by the API are GET, PUT, POST, and DELETE. These methods and the provided resources allow access to a single element or a collection of elements.

Internet media data types

The Internet media data types supported by the REST API are application/json and application/xml. For Java use of the REST API, the SecureTransport Java API provides a library called ws-representation, which includes Java classes that represent the data that can be accessed using the REST API. The ws-representation library depends on the jersey-core and jersey-server-linking libraries version 1.13 which are available from http://jersey.java.net/. In addition to them, Java programs require the jackson libraries version 1.9.2 and validation-api version 1.0.0 GA from J2EE 6. Representations are annotated according to JSR 303. Validation is done on the server side.

SecureTransport Swagger REST API integration 4

The SecureTransport Swagger REST API can be integrated into API Manager. Once the Swagger REST API is integrated into API Manager, the Frontend API and its Swagger definition from API Manager can be used to acquire a Java Client on API Gateway.

SecureTransport REST API Swagger integration in API Manager

This topic provides the prerequisites and instructions for integrating SecureTransport Swagger REST API into API Manager.

Prerequisites

The following prerequisites must be met prior to Swagger REST API being integrated in API Manager:

- SecureTransport Server must be installed, configured, and running.
  - The HTTPS and/or the Admin daemons (for the Client and the Admin APIs respectively) must be configured with certificates with CN - the IP address of the SecureTransport and no empty fields. API Gateway requires the fields to be completed.
- API Gateway and API Manager must be installed, configured, and running on a machine other than the SecureTransport server machine.

Limitation

When a Single Sign-On end-user logout is performed (**DELETE** request to `~/myself` resource), the server returns an **Error**.

This is caused by Swagger sending a standardized cURL request : `curl -X DELETE ~`.

The SSO logout uses **Redirects**, with a standardized **Request** from Swagger using **Redirects** is not possible. However, cURL has means to control redirect behavior through an additional option `-L`. The `-L` option instructs cURL to use the redirect method instead of re-using the initial request method: **DELETE** in this use case.

Integration

The take the following steps to integrate Swagger REST API into API Manager:

1. Log into API Manager with an administrator's account.
2. Navigate to **API Registration > Backend API > New API -> Import Swagger API**.
3. Compete the following fields:
   - *Source:* Swagger definition URL
   - *URL:* `https://<st_host>/api/v1.4/docs/swagger.json`

     Note `<st_host>` is the host or IP of the machine, on which SecureTransport Server is installed. Specifying port 444 will point to the Admin daemon – used for the Admin API.
   - *API name:* Enter the desired name.
   - *Organization:* Pick an existing one or create a new one. To create a new one, navigate to **Client Registry > Organizations > New Organization**.
   - *Authentication:* Enter the User credentials or Admin credentials for the Client API and Admin API respectively.

     Note The current version of API Manager only supports swagger definitions in JSON format.
4. Click **Import**.
5. Navigate to **API Registration > Frontend API > New API > New API from Backend API** and select the alias of the imported API.

   If you receive an "Internal server error", refresh the page (F5) and the new Frontend API will appear in the list. The error is probably caused by the missing optional *Host* field in the Swagger definition.
6. On the **Frontend API** tab click on the alias of created API to configure it as follows:
   - *Inbound:*
     - *Security:* **Pass Through**
     - *Resource path:* The path which will be exposed to the user for the API. The path must be unique.
   - *Outbound:*
     - *Authentication profile:* **No authentication**
     - *Backend service URL:* `https://<st_host>/api/v1.4`

       Note Setting **Pass Through** inbound and **No authentication** outbound means that API Gateway will allow the SecureTransport Server to process authentication requests. If you want to implement custom authentication in API Manager, refer to *API authentication* on page 42.
   - *Trusted Certificates:* Click the **Add** (+) icon.
     - *Source:* **URL**
     - *URL:* `https://<st_host>`
     - *Use for outbound:* **Yes**

- *Use for inbound:* **No**
- This will import the certificate from a daemon in SecureTransport – HTTPS or Admin. This certificate should have CN: IP of the SecureTransport and no empty fields. Empty fields will cause run-time internal server errors since API Gateway is sensitive to them.

7. Click **Save**.
8. On the **Frontend API** tab, select the alias of new API and then click **Manage selected > Publish**.

   The Frontend API can only be configured only if not published. To unpublish the Frontend API, select the alias of the new API and then click **Manage selected > Unpublish**.

API description

API Manager provides Swagger definitions for Frontend APIs. To access one:

1. Navigate to **API Catalog**.
2. Click on the alias a Frontend API.
3. Click **Download Swagger 2.0**.

A text file containing the Swagger definition in JSON format should start downloading.

API authentication

This section provides information how to setup basic authentication on the API Gateway for SecureTransport.

1. Navigate to **API Registration > Frontend API >** Select the API to edit **> Outbound**.
2. Set *Authentication profile* to **HTTP Basic** and provide a valid SecureTransport username and password.
3. Navigate to **Inbound** and set *Security* to **HTTP Basic**.
4. Navigate to **Client registry > Applications**.
5. Create a new application:
   a. Specify the desired API's organization.
   b. Under *API ACCESS*, click on **Add API** and select the desired API.
   c. Click **Create**.
   d. Under **Authentication > API KEYS**, create as many API keys as you need.

      Each API Key and its corresponding secret can be used as username/password pair for authentication on a Frontend API.

This setup represents a simple scenario. If you would like to setup advanced authentication and authorization methods, refer to the API Gateway documentation.

Acquiring a Java Client for a Frontend API on API Gateway

This topic provides the prerequisites and instructions for acquiring a Java client for a Frontend API on API Gateway.

Prerequisites

The following prerequisites must be met prior to acquiring a JAVA client for a Frontend API on API Gateway:

- Registered and published Frontend API in API Manager and its Swagger definition.
- JRE 8 with installed Java Cryptographic Extension (JCE) 8.

  Note API Gateway is configured to be accessed via a TLS secured connection only using a cipher suite that is available in JCE 8.

Generating client source code from Swagger definition

The take the following steps to generate client source code from a Swagger definition:

1. Go to http://editor.swagger.io/#/.
2. Click **File > Paste JSON**.
3. Paste the Swagger definition from API Manager and click **Import**.
4. Click **Generate Client > Java**.

An archive containing the source code should start downloading.

There is a test in the `io.swagger.client.api` package that shows a general usage of the generated client code. `DefaultApi` class is the entry point to the API.

Note Missing libraries can be acquired by importing the code into a maven project.

Configuring the code for SecureTransport API in API Gateway

The API Gateway trust and authentication must be configured for the SecureTransport API in API Gateway.

API Gateway trust

The client code is transferred to the API Gateway via TLS secure protocol. In addition to the uncommon cipher suite, the client code communication also uses a self-signed certificate and a not-trusted host. The certificate can be imported in the JVM's key store, while host trust is acquired using the OK HTTP (http://square.github.io/okhttp/) framework code generated for HTTP request-response modeled communication.

Importing API Gateway certificate

To import the API Gateway certificate:

1. Download or extract the API Gateway certificate.

   API Manager uses the same certificate for HTTPS. It can be extracted from a browser.

2. Locate the JRE keytool application.

   The default location is `<jre8_home>/bin`.

3. Execute the following command:

   `keytool -import -v -trustcacerts -alias <alias> -file <certificate> -keystore <store_name> -storepass <password>`, where:

   - `<alias>` - (string) - Alias for the imported certificate. The alias must be unique for the store.
   - `<certificate>` - The file containing the certificate of API Gateway.
   - `<store_name>` - The key store used by your JVM. The default key store is `cacerts.jks`.
   - `<password>` - The password for the keystore. The default password for the `cacerts.jks` data key store is **changeit**.

   Note On Microsoft Windows, execute the command as an administrator.

Trusting API Manager host

Trusting the API Manager host can be done by providing a custom host verification:

```
final DefaultApi api = new DefaultApi();
final ApiClient apiClient = api.getApiClient();
// allow only the API Gateway host
apiClient.getHttpClient().setHostnameVerifier(new HostnameVerifier()
{
    @Override
    public boolean verify(String hostname, SSLSession session) {
        // check if hostname is the IP of API Gateway
        return "10.232.3.104".equals(hostname);
    }
});
```

Authentication

Authentication is completed by the SecureTransport Server , but API Gateway does not know what authentication method is being used and as a result the generated code assumes that there is no authentication. SecureTransport REST API uses Basic HTTP authentication – username and password credentials. The Basic HTTP authentication has to be manually implemented in the code as follows:

1. The `authentications` property in the `io.swagger.client.ApiClient` class has to contain the following mapping:

   `"HTTPBasic" -> new HttpBasicAuth()`

2. In the `io.swagger.client.api.DefaultApi` class, several methods make a call to `apiClient.buildCall()` method, providing a string array as a parameter called `authNames`. This array must contain the string `"HTTPBasic"`.

3. Credentials must be also provided using instructions similar to the following:

```
final DefaultApi api = new DefaultApi();
final ApiClient apiClient = api.getApiClient();
apiClient.setUsername("admin");
apiClient.setPassword("admin");
```

Note The credentials are provided as an example. Ensure that you are using a valid admin account or Admin API. For the Client API, ensure that you use the credentials of a valid user account.

Single Sign-On with REST API (Enhanced Client or Proxy) 5

Note This example assumes both the Identity Provider and Service Provider are properly configured for Enhanced Client or Proxy (ECP) using basic authentication. For details, refer to the Shibboleth documentation.

Operating system and tools used

The following operating system and tools were used in this example:

- Red Hat 4.8.5-11
- curl - A tool for client-side URL transfers.
- tempfile or mktemp - Utility for creating a temporary file to store cookies.

Single Sign-On (SSO) authentication

1. REST client – Will request authentication to the Service Provider > SecureTransport (The Service Provider).

   Headers needed for ECP:

   - `Accept:text/html; application/vnd.paos+xml`
   - `PAOS:ver=\"urn:liberty:paos:2003-08\";\"urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp\"`

   Create a temp file where you can store the cookie.

   a. mktemp

   b. /cookieTemp- The created temp file where the cookies will be stored.

   CURL GET request to Service Provider:

   ```
   curl -v -k -L -c /cookieTemp -b /cookieTemp -H "Accept:text/html;
   application/vnd.paos+xml" -H "PAOS:ver=\"urn:liberty:paos:2003-
   08\";\"urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp\""
   https://<ServiceProvider IP>:<Service Provider PORT>
   ```

   The Service Provider will return a SAML Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap11:Envelope xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/">
  <soap11:Header>
    <paos:Request xmlns:paos="urn:liberty:paos:2003-08"
responseConsumerURL="<The Response Consumer URL>"
service="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"
soap11:actor="http://schemas.xmlsoap.org/soap/actor/next"
soap11:mustUnderstand="1"/>
    <ecp:Request xmlns:ecp="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"
soap11:actor="http://schemas.xmlsoap.org/soap/actor/next"
soap11:mustUnderstand="1">
      <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"><The
SAML Issuer></saml2:Issuer>
    </ecp:Request>
  </soap11:Header>
  <soap11:Body>
    <saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
Destination="<IDP ENDPOINT>" ForceAuthn="false"
ID="id-30735cecb29da6728ca33a52b3428846550069245d37a43d3cf562ef5e7f21df-
1491497240411-b5e88519-3536-4294-b06f-df7deeec376a" IsPassive="false"
IssueInstant="2017-04-06T16:47:20.431Z"
ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:PAOS" Version="2.0">
      <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"><The
SAML Issuer></saml2:Issuer>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ds:SignatureMethod
Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
          <ds:Reference URI="#id-30735cecb29da6728ca33a52b3428846550069245d37
a43d3cf562ef5e7f21df-1491497240411-b5e88519-3536-4294-b06f-df7deeec376a">
            <ds:Transforms>
              <ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
              <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue/>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue/>
        <ds:KeyInfo>
          <ds:X509Data>
            <ds:X509Certificate>
MIIDtzCCAp+gAwIBAgIBAjANBgkqhkiG9w0BAQsFADCBmTELMAkGA1UEAwwCY2ExCzAJBgNVBAYT
AmNhMQswCQYDVQQKDAJjYTELMAkGA1UECwwCY2ExCzAJBgNVBAcMAmNhMQswCQYDVQQIDAJjYTFJ
MEcGA1UEBRNANTU1MDQxZTZlZTcyMjY5OTRiZTQyMTI4YTE2ZjM5NjIwYWQ2OWEwZTFlNWFlYjY4
NDQ1M2ZmNThjZTEyZDhlZTAeFw0xNzA0MDYxMjM5MzJaFw0xODA0MDYxMjM5MzJaMFIxDzANBgNV
BAMMBnNzb2tleTELMAkGA1UEBhMCY2ExCzAJBgNVBAoMAmNhMQswCQYDVQQLDAJjYTELMAkGA1UE
BwwCY2ExCzAJBgNVBAgMAmNhMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2PIui/zG
t6l2xm2uqhjdvFkJK1yELUKXYXbLI+uAQglgNfCfGg2qEmk6lulCytKu8UYB+4M4mkMlshwkxTQQ
EUB+o5TATQKNrc6cT7PahdISnDF9AeGqpCr9OgndeMTFgA7LNAGBJZT/v5LgvvGiPbN/N31t9uEw
jyFNFB6Ywpu0+9pBPfWb4++dske9b+leXS6N/nm3cyLRvYhggScnKcr+OU1AxQLJetKTWy7lDZcr
jEGZiEsZc2pm8M5fFc0bUbaPvHjbqB2qNg9+nAVI2i0gHkUwFyd2cHER7p0HR9WdPOou+970SNCb
```

```
SEDtF/yQ9Pk4HuzdVIAvxHX1IV4fWQIDAQABo1AwTjAMBgNVHRMBAf8EAjAAMB0GA1UdDgQWBBSo
Srl4IyBmi053EURE3cN6YBgjozAfBgNVHSMEGDAWgBQnL3mdt9LybLMJRC8nrMcYi8NPgjANBgkq
hkiG9w0BAQsFAAOCAQEAfL+GLxrULu3Qv6zzg7TeRZyikClgGVeEFn3rpckDov2aFKexvQq9ArsR
CCygSvkfFS2EwvXrp9Q1i6prxDQB/cZJtQFdHYfxFT4nWFlmo28oMFWETzJnpMtANA/kt5WVg6Cb
fR0+QZbC5gQhR4F2Pr/61CSzstm25YRHOhkEylqnLnxiil7dpJfntAYOrWqxdrlrmiGS/287H02v
CAJWZ2em3jtSA1963Mi/ByvZZ1ZYUNNnzMwFnfEYFKdmJ4uEeTNc8+K6mjXZRZcoIwiT15v2rVwB
cRm+iSo6mJw+Ah+zUASX+30tNj/+v75Y8NADzW6iS7ilfdmD7IKz27eLag==
          </ds:X509Certificate>
         </ds:X509Data>
        </ds:KeyInfo>
      </ds:Signature>
      <saml2:Conditions
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
NotBefore="2017-04-06T16:47:20.431Z"
NotOnOrAfter="2017-04-06T16:48:20.431Z"/>
    </saml2p:AuthnRequest>
  </soap11:Body>
</soap11:Envelope>
```

We can save the Request in a temp file (`tmp.samlrequest`) and pass it as a parameter for the **POST** Request to the `IDP_ENDPOINT`:

2. The SAML Request then needs to be sent to the Identity Provider so it can challenge the user for authentication.

   The Identity Provider endpoint to which we need to send the SAML can be found in the SAML Request:

```
<saml2p:AuthnRequest
xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
Destination="<IDP_ENDPOINT>" ForceAuthn="false" ID="id-
30735cecb29da6728ca33a52b3428846550069245d37a43d3cf562ef5e7f21df-
1491497240411-b5e88519-3536-4294-b06f-df7deeec376a"
IsPassive="false" IssueInstant="2017-04-06T16:47:20.431Z"
ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:PAOS"
Version="2.0">
```

   The **POST** Request to the Identity Provider Endpoint

```
curl -v -k --fail -X POST -H 'Content-Type: text/xml; charset=utf-
8' -c /cookieTemp -b /cookieTemp --user <USERNAME> -d
"@/tmp.samlrequest" <IDP_ENDPOINT>
```

   The Identity Provider will challenge for authentication. After successful authentication, the Identity Provider will return a SOAP envelope, which then needs to be send to the Service Provider:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ecp="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
  <SOAP-ENV:Header>
```

```
    <ecp:Response AssertionConsumerServiceURL="<Assertion ConsumerService
URL>" SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next" SOAP-
ENV:mustUnderstand="1"/>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>

    <samlp:Response Destination="<Response Destination URL>"
ID="ID_5f5b3f52-7440-49ad-b9c2-d1513239e054"
InResponseTo="id-
30735cecb29da6728ca33a52b3428846550069245d37a43d3cf562ef5e7f21df-
1491497240411-b5e88519-3536-4294-b06f-df7deeec376a"
IssueInstant="2017-04-06T16:47:19.813Z" Version="2.0">
      <saml:Issuer><The SAML issuer></saml:Issuer>

      <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <dsig:SignedInfo>
          <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <dsig:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha256"/>
          <dsig:Reference URI="#ID_5f5b3f52-7440-49ad-b9c2-d1513239e054">
            <dsig:Transforms>
              <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
              <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </dsig:Transforms>
            <dsig:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <dsig:DigestValue>TMW8z+lkNsLH17DpOVDpVswbGwCdoa4DXSWEm5ERrNg=
</dsig:DigestValue>
          </dsig:Reference>

        </dsig:SignedInfo>
         <dsig:SignatureValue>
RWmMleH2nLausNmC4L9UN+lUiT7vYTMNgzBXXnDBukZTubISwCqrY/DSsrjlLS7tZvaodaGlSLM
+eJVimn0EgGHwj/3Vr6S8ohsOk+IzB5fHYpG8N7uoDadp6ILGwc7vU4bOK4L7zgorgaVN0Ofjmr
5sn5F1Rvh/sSmYUpbfX+3lhQzldXI0B7VpsISZAV9eum/c8HJGE/fvWz3kMneKxMB1b2p4wRNvy
ncuekoiKQ4YN/No3S7PKKFDcdaJV10m6CQUn8TsfJhmpDwot8PRkyqZmFUJXhoyT62oIBviC1Vo
oFxVBiJ7tWBeXKeJ1E15MuFtR8T2rLSll1Hvgwq1Nw==
         </dsig:SignatureValue>
        <dsig:KeyInfo>
          <dsig:KeyName>tmx7e0gwvdLC4G9HTPegtqq9-elg2rhwg-sM7r7_RfM</dsig:KeyName>

          <dsig:X509Data>
            <dsig:X509Certificate>
MIICtTCCAZ0CBgFZx7DxnzANBgkqhkiG9w0BAQsFADAeMRwwGgYDVQQDDBNTZWN1cmVUcmFuc3Bv
cnQtc3NvMB4XDTE3MDEyMjE5Mzc1OVoXDTI3MDEyMjE5MzkzOVowHjEcMBoGA1UEAwwTU2VjdXJl
VHJhbnNwb3J0LXNzbzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJ9gyQLXAYXz9vBL
rxpX+Ti3yTbJ8CTy4H5B3dwWk9X1voaL9a3uEV0UR4AdsDq+6ExYroYEUgCjVJVywK7eNitZgxrG
bnEKjYsEt/ZXTOaW0V3VV+CVzGZqOBdRbqC4+BFXoH3Jba1fy3frZomPE1xHGiotceUlz7m7Etnf
fnCzblJhYQY4QPu94wTaVykULWA5CkgiRjDC+5OWsKcQ6C/kmkMgI9S4RCS7N6Apj7ILL3jCm/df
+hVlND3ZpYPdRZSavjTiX0L2hglXyn91Zju8ZIm6w8tQDE0vFwjF+9A+w0mnjp+0gtV5ISiCW/0s
K35xcgIsqF84XWQZyACG8VcCAwEAATANBgkqhkiG9w0BAQsFAAOCAQEAbbttR21ubHush5EsuVQj
LAIrsJyJTdodgFMUaOfUolFb7AHIlUWhUALjgWd/6S6KXwSdfPinE3NC9dmUY+S+bxkyDHrHzzib
cpsVaP8TK031PhjbTvH22RgGl77nqq3cRKfte+3Eqi45CqBKfEFMt8YsrAsIzW/T9oeMkLP0M8mw
RO9x280UID+rqSsKseAIFUE2Pxc/YBGID2qbIDKM5NI97ALbxhts3b2GW1PiCyDo8EIq2DlJXWEs
vzCkavfmN/lmx5YvjviXGdzKw7r5KWyKPVoCePtlZc/fcBRk4dZPUFWW6NlA7liRqzAEi7KJvPGL
yfD5rbAwatPnbOW40Q==
           </dsig:X509Certificate>
          </dsig:X509Data>
          <dsig:KeyValue>
```

```
            <dsig:RSAKeyValue>
              <dsig:Modulus>
n2DJAtcBhfP28EuvGlf5OLfJNsnwJPLgfkHd3BaT1fW+hov1re4RXRRHgB2wOr7oTFiuhgRSAKNU
lXLArt42K1mDGsZucQqNiwS39ldM5pbRXdVX4JXMZmo4F1FuoLj4EVegfcltrV/Ld+tmiY8TXEca
Ki1x5SXPubsS2d9+cLNuUmFhBjhA+73jBNpXKRQtYDkKSCJGMML7k5awpxDoL+SaQyAj1LhEJLs3
oCmPsgsveMKb91/6FWU0Pdmlg91FlJq+NOJfQvaGCVfKf3VmO7xkibrDy1AMTS8XCMX70D7DSaeO
n7SC1XkhKIJb/SwrfnFyAiyoXzhdZBnIAIbxVw==
              </dsig:Modulus>
              <dsig:Exponent>AQAB</dsig:Exponent>
            </dsig:RSAKeyValue>
          </dsig:KeyValue>
        </dsig:KeyInfo>
      </dsig:Signature>
      <samlp:Status>
        <samlp:StatusCode
Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
      </samlp:Status>
      <saml:Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
ID="ID_f36812f5-66ce-4b30-88fc-f8e0071b1587"
IssueInstant="2017-04-06T16:47:19.812Z" Version="2.0">
        <saml:Issuer><The SAML issuer></saml:Issuer>
        <saml:Subject>
          <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:
unspecified">kmanchev</saml:NameID>
          <saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml:SubjectConfirmationData
InResponseTo="id-30735cecb29da6728ca33a52b3428846550069245d37a43d3cf562ef5e7f
21df-1491497240411-b5e88519-3536-4294-b06f-df7deeec376a" NotOnOrAfter="2017-
04-06T16:52:17.812Z"
Recipient="<Response Destination URL>"/>
          </saml:SubjectConfirmation>
        </saml:Subject>
        <saml:Conditions NotBefore="2017-04-06T16:47:17.812Z"
NotOnOrAfter="2017-04-06T16:48:17.812Z">
          <saml:AudienceRestriction>
            <saml:Audience>koce-admin-10.134.64.151</saml:Audience>
          </saml:AudienceRestriction>
        </saml:Conditions>
        <saml:AuthnStatement AuthnInstant="2017-04-06T16:47:19.813Z"
SessionIndex="f414258a-df9b-43c7-a8cf-11edfe78a423">
          <saml:AuthnContext>
            <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:
classes:unspecified
</saml:AuthnContextClassRef>
          </saml:AuthnContext>
        </saml:AuthnStatement>
        <saml:AttributeStatement>
          <saml:Attribute Name="email"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <saml:AttributeValue
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">kmanchev@st1.lab.sofi.axway.int</saml:AttributeValue>
          </saml:Attribute>
```

```
          <saml:Attribute Name="Role"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <saml:AttributeValue
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:type="xs:string">manage-account</saml:AttributeValue>
          </saml:Attribute>
          <saml:Attribute Name="Role"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">uma_authorization</saml:AttributeValue>
          </saml:Attribute>
          <saml:Attribute Name="Role"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <saml:AttributeValue
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">view-profile</saml:AttributeValue>
          </saml:Attribute>
        </saml:AttributeStatement>
      </saml:Assertion>
    </samlp:Response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

We can save the Response in a temp file (`tmp.idpreponse`) and pass it as a parameter for the **POST** Request to the Service Provider.

Note We need to compare the `responseConsumerURL` from the Service Provider to the `assertionConsumerServiceURL` from the Identity Provider. If they are not identical, we need to send a SOAP fault to the Service Provider.

Payload:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<S:Fault>
<faultcode>
S:Server</faultcode>
<faultstring>responseConsumerURL from SP and
assertionConsumerServiceURL from IdP do not match</faultstring>
</S:Fault>
</S:Body>
</S:Envelope>
```

The payload can be saved to a file (`fault_payload`) and used in the **POST** request:

```
curl -v -k -X POST -c /cookieTemp -b /cookieTemp -d "@/fault_
payload" -H "Content-Type: application/vnd.paos+xml" <The Response
Consumer URL>
```

3. After successful authentication and check if `responseConsumerURL` from the Service Provideer and the `assertionConsumerServiceURL` from the Identity Provide are identical, we are ready to send the Identity Provider Payload to the Service Provider:

```
curl -v -k -s -c /cookieTemp -b /cookieTemp -X POST -d
"@/tmp.idpreponse" -H "Content-Type: application/vnd.paos+xml"
<Assertion ConsumerService URL>
```

   The session should now be established.

4. You can try to access the original resource to check if the authentication was successful:

```
curl -k -v -L -e "https://<ServiceProvider IP>:<Service Provider
PORT>" -c /cookieTemp -b /cookieTemp -H "Accept: application/json"
https://<ServiceProvider IP>:<Service Provider PORT>
```

Single Sign-Off (SSO) logout

Administrator logout:

```
curl -v -k -L -c /cookieTemp -b /cookieTemp -D - -X GET
https://<ServiceProvider IP>:<Service Provider PORT>/coreadmin/logout
```

End-user logout:

```
curl -v -k -L -c /cookieTemp -b /cookieTemp -D - -H "Referer:*" -X
DELETE https://<ServiceProvider IP>:<Service Provider
PORT>/api/v1.4/myself
```

After the logout is performed, the session and session cookies are expired.

Note For logout , SecureTransport uses Redirect Bindings, so in order for the logout to work:

- Single Logout (SLO) needs to be configured with **Redirect** binding. In the SecureTransport SSO configuration, the **Redirect** binding must be before the **POST** binding.
- The *Force POST Binding* option must be disabled for Keycloak.

Login recap

CLIENT - **Requests** the / of the SecureTransport (Service Provider) → **SecureTransport (Service Provider)**

CLIENT ← **Responds** with SAML Request envelope - **SecureTransport (Service Provider)**

CLIENT - **Sends** the SAML Request from the Service provider to the Identity Provider → **Identity Provider (Keycloak/Shibboleth)**

CLIENT ← **Challenges** the Client for authentication - **Identity Provider (Keycloak/Shibboleth)**

CLIENT - **Authenticates** correctly → **Identity Provider (Keycloak/Shibboleth)**

CLIENT ← **Responds** with Identity Provider Response - **Identity Provider (Keycloak/Shibboleth)**

CLIENT - **Sends** the IDP Response to the Service Provider → **SecureTransport (Service Provider)**

The Service provider sets the Session cookies and the session is established.

Logout recap

CLIENT - **Requests** logout (if EndUser - will perform **DELETE**) → **SecureTransport (Service Provider)**

CLIENT ← **Responds** with Logout SAML Request- **SecureTransport (Service Provider)**

CLIENT - **Sends** Logout SAML Request → **Identity Provider (Keycloak/Shibboleth)**

CLIENT ← **Responds** with Logout SAML Response - **Identity Provider (Keycloak/Shibboleth)**

CLIENT - **Sends** the Logout SAML Response from the IDP to confirm that the logout was successul → **SecureTransport (Service Provider)**

Custom Address Book implementation 6

Custom Address Book sources can be implemented in the Address Book feature. Implementing a custom Address Book consists of creating provider, criteria, and configuration classes and registering the Address Book provider.

Note The version of Custom Address Book SPI this Developer's Guide describes is 1.0.0-3.

Custom Address Book implementation steps

To implement a custom Address Book (AB), execute the following steps:

Step	Description
Create a provider class on page 54	Used to hold the main logic for an address book. It is responsible for returning Address Book entries specified by search criteria.
Create a criteria class on page 61	Used to define the Address Book search criteria for entries.
Create a configuration class on page 63	Used as the configuration settings for an Address Book.
Register Address Book provider on page 64	Every Address Book provider must be packaged in a specific way in order to be registered correctly into SecureTransport system.

Create a provider class

The provider class is responsible for generating a list of Address Book (AB) entries based on specified criteria. Every entry generated by the provider must have a unique ID and a unique name across all entries generated by the provider. The search of AB entries is defined by search criteria and a subject. The subject object represents a user account which has requested the provider for a set of AB entries. It can be used to distinguish which user is sending the request and to return a set of address book entries unique to the requesting user. If the implementation does not have a specific requirement, it can be ignored.

The `AddressBookProvider` interface must be implemented in order to create a new address book provider into SecureTransport. Prior to implementing the `AddressBookProvider` interface, the `addressbook.txt` file must be placed in `<FILEDRIVEHOME>` for the example provider class. The location of the file that will be used is specified in the `loadAddressBook()` method.

Note If more than one custom Address Book source is created, the `mName` must be unique for each custom Address Book source.

The structure of the `addressbook.txt` file is:

```
name,email,parentGroup,organization
```

Example `addressbook.txt` file format and content:

```
John Doe,jdoe@company.com,R&D Team,organization=Company
Jane Smith,jsmith@company.com,Tactical Team,organization=Company
```

The following code illustrates a custom file provider implementation, which reads its entries from a local file:

```
package com.axway.st.plugins.abcustomsource.sdk.file.provider;

import com.axway.st.plugins.absource.AddressBookCriteria;
import com.axway.st.plugins.absource.AddressBookEntry;
import com.axway.st.plugins.absource.AddressBookEntry.EntryId;
import com.axway.st.plugins.absource.AddressBookEntryIterator;
import com.axway.st.plugins.absource.AddressBookGlobalConstraintException;
import com.axway.st.plugins.absource.AddressBookPermissionException;
import com.axway.st.plugins.absource.AddressBookProvider;
import com.axway.st.plugins.absource.AddressBookProviderConfiguration;
import com.axway.st.plugins.absource.Subject;

import
com.axway.st.plugins.abcustomsource.sdk.file.configuration.AddressBookFile
ProviderConfiguration;
import
com.axway.st.plugins.abcustomsource.sdk.file.criterion.AddressBookFileCriteria;
import
com.axway.st.plugins.abcustomsource.sdk.file.criterion.MyAddressBookEntryBean;

/**
 * Address Book File Provider.
 *
 */
public class AddressBookFileProvider implements AddressBookProvider {

    /**
     * The address book provider configuration instance.
     */
    private static AddressBookProviderConfiguration sConfiguration;

    /**
     * The address book name.
```

```
     */
    private String mName;

    /**
     * Holds all address book entries.
     */
    private Collection<AddressBookEntry> mEntries;

    /**
     * The index of the display name when parsing the AB file.
     */
    private static final int DISPLAY_NAME = 0;

    /**
     * The index of the parent group when parsing the AB file.
     */
    private static final int PARENT_GROUP = 2;

    /**
     * The index of the custom properties when parsing the AB file.
     */
    private static final int CUSTOM_PROPS = 3;

    /**
     * The index of the email when parsing the AB file.
     */
    private static final int EMAIL = 1;

    /**
     * The file name of the address book.
     */
    private static final String ADDRESS_BOOK = "addressbook.txt";

    static {
        sConfiguration = new AddressBookFileProviderConfiguration();
    }
    /**
     * Default constructor which loads the address book.
     * The mName must have a different value for custom address book.
     */
    public AddressBookFileProvider() {
        super();
        mName = "File Address Book";
        mEntries = new ArrayList<>();
        loadAddressBook();
    }

    @Override
    public String getAddressBookName() {
        return mName;
    }
```

```
    private void loadAddressBook(){
        String home = System.getProperty("FILEDRIVEHOME");
        File addressBook = new File(home, ADDRESS_BOOK);

        BufferedReader input = null;
        try {
            input = new BufferedReader(new FileReader(addressBook));
            String line = input.readLine();
            int linNum = 1;
            while(line != null){
                String []contact = line.split(",");
                MyAddressBookEntryBean entry = new MyAddressBookEntryBean();
                entry.setDisplayName(contact[DISPLAY_NAME].trim());
                entry.setEmail(contact[EMAIL].trim());
                entry.setId(newEntryId(Integer.toString(linNum)));
                entry.setType(AddressBookEntry.Type.User);
                entry.setParentGroup(contact[PARENT_GROUP].trim());

                HashMap<String, String> customProperties = new HashMap<>();
                String propStr = contact[CUSTOM_PROPS];
                if(propStr != null && propStr.trim().length() > 0){
                    String [] properties = propStr.split(";");
                    for(String propEntry : properties){
                        String key = propEntry.split("=")[0];
                        String value = propEntry.split("=")[1];
                        customProperties.put(key, value);
                    }
                    entry.setCustomProperties(customProperties);
                }
                mEntries.add(entry);
                line = input.readLine();
                linNum++;
            }
        } catch (IOException e) {
            // TODO Handle exception
        } finally {
            if(input != null){
                try {
                    input.close();
                } catch (IOException e) {
                }
            }
        }
    }

    @Override
    public List<AddressBookEntry> getAddressBookEntries(Subject subject,
Integer first, Integer count) {
```

```
        TreeMap<String, AddressBookEntry> sortedByDisplayName = new TreeMap<>
();
        for(AddressBookEntry entry : mEntries){
            sortedByDisplayName.put(entry.getDisplayName(), entry);
        }

        int counter = 0;
        int added = 0;
        ArrayList<AddressBookEntry> paginatedView = new ArrayList<>();
        for(Map.Entry<String, AddressBookEntry> entry :
sortedByDisplayName.entrySet()){
            if(added > 0 && added < count){
                paginatedView.add(entry.getValue());
                added++;
            }

            if(counter == first){
                paginatedView.add(entry.getValue());
                added++;
            }

            counter ++;
        }
        return paginatedView;
    }

    @Override
    public List<AddressBookEntry> getAddressBookEntries(Subject subject,
AddressBookCriteria criteria,
            Integer first, Integer count) {
        if(criteria instanceof AddressBookFileCriteria){
            AddressBookFileCriteria c = (AddressBookFileCriteria)criteria;
            return c.applyCriterion(mEntries, first, count);
        }

        return null;
    }

    @Override
    public AddressBookEntry getAddressBookEntry(Subject subject, EntryId
entryId)
            throws AddressBookPermissionException {
        for (AddressBookEntry entry : mEntries) {
            if (entry.getEntryId().getUniqueId().equals(entryId.getUniqueId()))
{
                return entry;
```

```
            }
        }
        return null;
    }

    @Override
    public AddressBookEntry createAddressBookEntry(Subject subject,
AddressBookEntry entry)
            throws AddressBookGlobalConstraintException,
AddressBookPermissionException {
        throw new AddressBookPermissionException("Operation not allowed.");
    }

    @Override
    public AddressBookEntry updateAddressBookEntry(Subject subject,
AddressBookEntry entry)
            throws AddressBookGlobalConstraintException,
AddressBookPermissionException {
        throw new AddressBookPermissionException("Operation not allowed.");
    }

    @Override
    public void deleteAddressBookEntry(Subject subject, EntryId entryId)
            throws AddressBookPermissionException {
        throw new AddressBookPermissionException("Operation not allowed.");
    }

    @Override
    public AddressBookEntry newAddressBookEntry() {
        return new MyAddressBookEntryBean();
    }

    @Override
    public AddressBookCriteria newAddressBookCriteria() {
        return new AddressBookFileCriteria();
    }

    @Override
    public AddressBookProviderConfiguration getConfiguration() {
        return sConfiguration;
    }

    @Override
    public List<AddressBookEntry> getAddressBookEntries(Subject subject,
List<EntryId> entryIdList)
            throws AddressBookPermissionException {
        List<AddressBookEntry> entries = new ArrayList<AddressBookEntry>();
        for (AddressBookEntry entry : mEntries) {
            for (EntryId id : entryIdList) {
                String uniqueId = entry.getEntryId().getUniqueId();
```

```
iteria, 0, Integer.MAX_VALUE);
        return new MyAddressBookEntryIteratorImpl(entries);
    }

    @Override
    public EntryId newEntryId(String id) {
        return new MyAddressBookEntryBean.Id(id);
    }

    @Override
    public boolean isSingleton() {
        return false;
    }
}
```

If an Address Book provider cannot support create, update, and delete operations over its data store it will throw an appropriate exception when an unsupported operation is requested.

```
@Override
    public AddressBookEntry createAddressBookEntry(Subject subject,
AddressBookEntry entry)
            throws AddressBookConstraintException,
AddressBookPermissionException {
        throw new AddressBookPermissionException("Operation not allowed.");
    }

@Override
    public AddressBookEntry updateAddressBookEntry(Subject subject,
AddressBookEntry entry)
            throws AddressBookConstraintException,
AddressBookPermissionException {
        throw new AddressBookPermissionException("Operation not allowed.");
    }

@Override
    public void deleteAddressBookEntry(Subject subject, Id entryId)
            throws AddressBookPermissionException {
        throw new AddressBookPermissionException("Operation not allowed.");
    }
```

Once defined the provider implementation must return a new instance of its criteria object.

```
@Override
    public AddressBookCriteria newAddressBookCriteria() {
        return new AddressBookFileCriteria();
    }
@Override
    public Configuration getConfiguration() {
        return sConfiguration;
    }
```

The provider class must also return an instance with its configuration. The configuration object stores the default parent group name and adds a built-in parent group entry if SecureTransport needs it.

The parent group name is a SecureTransport virtual group, which is used as a parent for all entries coming from the provider. When an Address Book provider is initially registered into SecureTransport, the default parent group property will be read from the Address Book provider configuration.

Address Book group entries must be generated with unique names. Duplicate group names will not be properly resolved when a user sends a message to a group since SecureTransport resolves address groups based on the display name property.

If the Address Book provider implementation requires the creation of a single instance, it must return true as a result of a `isSingleton` method call:

```
@Override
    public boolean isSingleton() {
        return true;
    }
```

Create a criteria class

The Address Book (AB) criteria object will be passed to each provider method which needs to apply filtering over the result set of its own entries. In order to add the custom implementation of the AB criteria, the `AddressBookCriteria` interface must be implemented.

Note The Address Book framework use the provider specific criteria object when getting entries from it. How they are filtered depends on the provider itself. Address Book entries are not passed to the provider, only combinations of the criteria, user info, and pagination settings.

In the sample code, the `AddressBookFileCriteria` class extends the system base class which provides implementation for all Gets and Sets of the Address Book entry properties.

The `AddressBookFileCriteria` class defines a new method which handles the filtering:

```
package com.axway.st.plugins.abcustomsource.sdk.file.criterion;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;

import com.axway.st.plugins.absource.AddressBookEntry;

/**
```

```
 * Address Book File Criteria.
 *
 */
public class AddressBookFileCriteria extends MyAddressBookCriteriaBean {

    /**
     *
     */
    private static final long serialVersionUID = -9087309532695089048L;

    /**
     * Apply the search criterion.
     *
     * @param entries the input entries
     * @param start the offset
     * @param count the max size
     * @return filtered and ordered set of address book entries
     */
    public List<AddressBookEntry> applyCriterion(Collection<AddressBookEntry>
entries, int start, int count){
        List<AddressBookEntry> resultSet = new ArrayList<>();
        for(AddressBookEntry entry : entries){
            if(getDisplayName() != null){
                if(!getDisplayName().equals(entry.getDisplayName())){
                    continue;
                }
            }

            if(getEmail() != null){
                if(!getEmail().equals(entry.getEmail())){
                    continue;
                }
            }

            if(getId() != null){
                if(!getId().getUniqueId().equals(entry.getEntryId().getUniqueId
())){
                    continue;
                }
            }

            if(getParentGroup() != null){
                if(!getParentGroup().equals(entry.getParentGroup())){
                    continue;
                }
            }

            if(getSearchFor() != null){
                if(!entry.getParentGroup().toLowerCase().
                        contains(getSearchFor().toLowerCase())
                    && !entry.getDisplayName().toLowerCase().
                        contains(getSearchFor().toLowerCase())
```

```
                        contains(getSearchFor().toLowerCase())){
                    continue;
                }
            }

            if (getType() != null) {
                if (getType() == AddressBookEntry.Type.Group) {
                    continue;
                }
            }

            resultSet.add(entry);
        }

        Collections.sort(resultSet, new MyAddressBookContactComparator(getOrder
()));

        List<AddressBookEntry> paginatedView = new LinkedList<>();
        if (start > resultSet.size() - 1) {
            return paginatedView;
        }
        ListIterator<AddressBookEntry> iterator = resultSet.listIterator
(start);
        while (iterator.hasNext()) {
            AddressBookEntry entry = iterator.next();
            paginatedView.add(entry);
            if (paginatedView.size() == count) {
                break;
            }
        }
        return paginatedView;
    }
}
```

Create a configuration class

A new file configuration class is created in the sample code. The `isParentGroupAdded` method specifies if the provider requires the SecureTransport server to generate a system Address Book entry which will be used as a parent group for all Address Book entries returned by the provider.

```
package com.axway.st.plugins.abcustomsource.sdk.file.configuration;

import com.axway.st.plugins.absource.AddressBookProviderConfiguration;

/**
 * File Configuration.
```

```
 *
 */
public class AddressBookFileProviderConfiguration implements
AddressBookProviderConfiguration {

    @Override
    public String getDefaultParentGroupName() {
        return "file-based";
    }

    @Override
    public boolean isParentGroupAdded() {
        return true;
    }
}
```

Register Address Book provider

Once an Address Book provider is implemented it needs to be registered into the SecureTransport system. To register an Address Book provider, a new file with the name `com.axway.st.plugins.absource.AddressBookProvider` must be created and placed inside the provider `jar` file in the `META-INF/services` folder. The file must contain the full class name of the Address Book provider.

Example:

```
com.axway.st.plugins.abcustomsource.sdk.file.provider.
AddressBookFileProvider
```

Once the provider is properly packaged, it must be placed inside the `<FILEDRIVEHOME>/lib/jars` folder. On Transaction Manager startup, the provider class will be dynamically loaded and the new Address Book will be registered into the SecureTransport system.

Note Transaction Manager must be restarted for the changes to be successfully applied.

Additional classes

This topic includes the additional SecureTransport classes needed to create a new custom provider. The javadoc for the provided classes is part of the SecureTransport SDK package.

MyAddressBookEntryIteratorImpl class

```
package com.axway.st.plugins.abcustomsource.sdk.file.provider;
```

```
import java.io.IOException;
import java.util.Collection;
import java.util.Collections;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.NoSuchElementException;


import com.axway.st.plugins.absource.AddressBookEntry;
import com.axway.st.plugins.absource.AddressBookEntryIterator;


import
com.axway.st.plugins.abcustomsource.sdk.file.criterion.MyAddressBookEntryBean;


/**
 * AddressBookEntryIteratorImpl.
 */
public class MyAddressBookEntryIteratorImpl implements AddressBookEntryIterator
{


    /**
     * Enumeration instance to be wrapped by this specific iterator
implementation
     * One of the possible AddressBookEntry containers.
     * Available AddressBookEntry containers are: {@code mNamingEnumeration},
{@code mEnumeration} of {@code mIterator}
     * One of them must not be null
     */
    private Enumeration<AddressBookEntry> mEnumeration;


    /**
     * Iterator instance to be wrapped by this specific iterator implementation
     * One of the possible AddressBookEntry containers.
     * Available AddressBookEntry containers are: {@code mNamingEnumeration},
{@code mEnumeration} of {@code mIterator}
     * One of them must not be null
     */
    private Iterator<AddressBookEntry> mIterator;


    /**
     *
     */
    private AddressBookEntry mDefaultEntry = null;


    /**
```

```
     *
     */
    private String mSourceId;


    /**
     * The default parent group name to be set.
     */
    private String mDefaultGroupName;


    /**
     *
     */
    private boolean mIsSorted = false;


    /**
     *
     */
    private boolean mIsPaginated = false;


    /**
     * Empty AddressBookEntryIterator constructor.
     */
    protected MyAddressBookEntryIteratorImpl() {
        super();
    }


    /**
     * @return Empty iterator object
     */
    public static AddressBookEntryIterator getEmpty() {
        return new MyAddressBookEntryIteratorImpl();
    }


    /**
     * AddressBookEntryIterator constructor.
     * @param collection Collection with  {@code <AddressBookEntry>} entries.
     */
    public MyAddressBookEntryIteratorImpl(Collection<AddressBookEntry>
collection) {
        super();
        mEnumeration = Collections.<AddressBookEntry>enumeration(collection);
    }


    /**
     * AddressBookEntryIterator constructor.
```

```
     * @param enumeration Enumeration with  {@code <AddressBookEntry>} entries.
     */
    public MyAddressBookEntryIteratorImpl(Enumeration<AddressBookEntry>
enumeration) {
        super();
        mEnumeration = enumeration;
    }


    /**
     * AddressBookEntryIterator constructor.
     * @param iterator Iterator with  {@code <AddressBookEntry>} entries.
     */
    public MyAddressBookEntryIteratorImpl(Iterator<AddressBookEntry> iterator)
{
        super();
        mIterator = iterator;
    }


    @Override
    public boolean hasNext() {
        if(mDefaultEntry != null){
            return true;
        }


        if (mEnumeration != null) {
            Boolean hasMore = mEnumeration.hasMoreElements();
            return hasMore;
        } else if (mIterator != null) {
            Boolean hasMore = mIterator.hasNext();
            return hasMore;
        } else {
            return false;
        }
    }


    @Override
    public AddressBookEntry next() {
        AddressBookEntry result = null;


        if(mDefaultEntry != null){
            result = mDefaultEntry;
            mDefaultEntry = null;
            return result;
        }


        if (mEnumeration==null && mIterator==null) {
```

```
            throw new NoSuchElementException();
        } else if (mIterator!=null) {
            result = mIterator.next();
        } else if (mEnumeration!=null) {
            result = mEnumeration.nextElement();
        }


        if (mSourceId != null) {
            String id = result != null && result.getEntryId() != null ?
result.getEntryId().getUniqueId() : null;
            result.setEntryId(new MyAddressBookEntryBean.Id(id, mSourceId));
        }


        if (mDefaultGroupName != null && (result.getParentGroup() == null ||
result.getParentGroup().length() == 0)) {
            ((MyAddressBookEntryBean) result).setParentGroup
(mDefaultGroupName);
        }
        return result;
    }


    /**
     * Closes the context and the tls(if any).
     * @throws LdapAddressBookEntryFinderException
     */
    @Override
    public void close() throws IOException {
    }


    @Override
    public void remove() {
        throw new UnsupportedOperationException();
    }


    /**
     * Add an address book entry to the iterator.
     * @param entry - the entry to be added
     */
    @Override
    public void addEntry(AddressBookEntry entry){
        mDefaultEntry = entry;
    }


    /**
     * Sets the Entry sourceId.
     * @param id
```

```
     */
    @Override
    public void setEntrySourceId(String id) {
        mSourceId = id;
    }


    /**
     * Gets the Entry sourceId.
     * @return SourceID Associated Entry sourceId to the enumerator
     */
    @Override
    public String getEntrySourceId() {
        return mSourceId;
    }


    @Override
    public void setDefaultGroupName(String defaultGroupName) {
        mDefaultGroupName = defaultGroupName;
    }


    @Override
    public String getDefaultGroupName() {
        return mDefaultGroupName;
    }


    @Override
    public boolean isSorted() {
        return mIsSorted;
    }


    @Override
    public boolean isPaginated() {
        return mIsPaginated;
    }


    /**
     * @param isSorted
     */
    protected void setIsSorted(boolean isSorted) {
        mIsSorted = isSorted;
    }


    /**
     * @param isPaginated
     */
```

```
    protected void setIsPaginated(boolean isPaginated) {
        mIsPaginated = isPaginated;
    }
}
```

MyAddressBookEntryBean class

```
package com.axway.st.plugins.abcustomsource.sdk.file.criterion;


import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
import com.axway.st.plugins.absource.AddressBookEntry;


/**
 *
 */
public class MyAddressBookEntryBean implements AddressBookEntry {
    /**
     *
     */
    private static final long serialVersionUID = 861135864414006671L;


    /** The entry ID. It is a composition of unique ID and address book source
ID. */
    private AddressBookEntry.EntryId mId;


    /** The entry display name. */
    private String mDisplayName;


    /** The entry email address. */
    private String mEmail;


    /** The entry parent group. */
    private String mParentGroup;


    /** The entry type. It could be a user or a group type. */
    private Type mType;


    /** The custom properties of the entry. */
    private Map<String, String> mCustomProperties;
```

```
    /**
     * Default constructor.
     */
    public MyAddressBookEntryBean() {
        mCustomProperties = new HashMap<>();
    }


/**
 * Copyright (c) Axway Software, 2016. All Rights Reserved.
 */


    /**
     * Create a new address book entry.
     *
     * @param id - the entry id
     * @param displayName - the display name
     * @param email - the email.
     * @param parentGroup - the parent group.
     * @param type - the type of entry
     */
    public MyAddressBookEntryBean(Id id, String displayName, String email,
String parentGroup, Type type) {
        this();
        mId = id;
        mDisplayName = displayName;
        mEmail = email;
        mParentGroup = parentGroup;
        mType = type;
    }


    /**
     * Create a new address book entry.
     *
     * @param id - the entry id
     * @param displayName - the display name
     * @param email - the email.
     * @param parentGroup - the parent group.
     * @param type - the type of entry
     * @param customProperties - the custom properties
     */
    public MyAddressBookEntryBean(Id id, String displayName, String email,
String parentGroup, Type type,
            Map<String, String> customProperties) {
        this(id, displayName, email, parentGroup, type);
        mCustomProperties.putAll(customProperties);
    }
```

```
    @Override
    public AddressBookEntry.EntryId getEntryId() {
        return mId;
    }


    @Override
    public String getDisplayName() {
        return mDisplayName;
    }


    @Override
    public String getEmail() {
        return mEmail;
    }


    @Override
    public String getParentGroup() {
        return mParentGroup;
    }


    @Override
    public Type getType() {
        return mType;
    }


    @Override
    public Map<String, String> getCustomProperties() {
        Map<String, String> result = new HashMap<String, String>();
        result.putAll(mCustomProperties);
        return result;
    }


    /**
     * Gets the id.
     *
     * @return the id
     */
     public AddressBookEntry.EntryId getId() {
         return mId;
    }


    /**
     * Sets the custom properties.
     *
     * @param customProperties - the custom properties map
```

```
     */
    public void setCustomProperties(Map<String, String> customProperties) {
        mCustomProperties.clear();
        mCustomProperties.putAll(customProperties);
    }


    /**
     * Sets the id.
     *
     * @param id the id to set
     */
     public void setId(AddressBookEntry.EntryId id) {
         mId = id;
     }


    /**
     * Sets the displayName.
     *
     * @param displayName the displayName to set
     */
    public void setDisplayName(String displayName) {
        mDisplayName = displayName;
    }


    /**
     * Sets the email.
     *
     * @param email the email to set
     */
    public void setEmail(String email) {
        mEmail = email;
    }


    /**
     * Sets the parentGroup.
     *
     * @param parentGroup the parentGroup to set
     */
    public void setParentGroup(String parentGroup) {
        mParentGroup = parentGroup;
    }


    /**
     * Sets the type.
     *
     * @param type the type to set
     */
```

```
    public void setType(Type type) {
        mType = type;
    }


    @Override
    public int hashCode() {
        if (mId != null) {
            return mId.hashCode();
        } else {
            int result = 1;
            int prime = 31;
            result = prime * result + (mDisplayName == null ? 0 :
mDisplayName.hashCode());
            result = prime * result + (mEmail == null ? 0 : mEmail.hashCode());
            result = prime * result + (mParentGroup == null ? 0 :
mParentGroup.hashCode());
            result = prime * result + (mType == null ? 0 : mType.hashCode());
            result = prime * result + (mCustomProperties == null ? 0 :
mCustomProperties.hashCode());
            return result;
        }
    }


    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }


        if (o == null) {
            return false;
        }


        if (!(o instanceof MyAddressBookEntryBean)) {
            return false;
        }


        MyAddressBookEntryBean entry = (MyAddressBookEntryBean) o;
        if (mId != null) {
            return mId.equals(entry.mId);
        }


        if (mDisplayName == null) {
            if (entry.mDisplayName != null) {
                return false;
            }
```

```
        } else if (!mDisplayName.equals(entry.mDisplayName)) {
            return false;
        }


        if (mEmail == null) {
            if (entry.mEmail != null) {
                return false;
            }
        } else if (!mEmail.equals(entry.mEmail)) {
            return false;
        }


        if (mParentGroup == null) {
            if (entry.mParentGroup != null) {
                return false;
            }
        } else if (!mParentGroup.equals(entry.mParentGroup)) {
            return false;
        }


        if (mType == null) {
            if (entry.mType != null) {
                return false;
            }
        } else if (!mType.equals(entry.mType)) {
            return false;
        }


        if (mCustomProperties == null) {
            if (entry.mCustomProperties != null) {
                return false;
            }
        } else if (!mCustomProperties.equals(entry.mCustomProperties)) {
            return false;
        }
        return true;
    }


    @Override
    public String toString() {
        return "{display name [" + mDisplayName + "] " + "email [" + mEmail +
"] parent group [" + mParentGroup + "] "
            + "custom properties " + mCustomProperties + "}";
    }


    @Override
```

```
    public void setEntryId(AddressBookEntry.EntryId entryId) {
        mId = entryId;
    }


    /**
     * Address book entry Id implementation.
     */
    public static class Id implements AddressBookEntry.EntryId {


        /** The serial version. */
        private static final long serialVersionUID = 6066501149301629310L;


        /** The provider specific entry id combined with the source id. */
        private String mUniqueId;


        /** The address book source id. */
        private String mAddressBookSourceId;


        /** The address book entry id. */
        private String mEntryId;


        /**
         * Create a new entry id instance.
         *
         * @param id - the entry id.
         */
        public Id(String id) {
            mEntryId = id;
            mUniqueId = id;
            mAddressBookSourceId = null;
        }


        /**
         * Create a new entry id instance.
         *
         * @param id - the entry id.
         * @param sourceId - the source id.
         */
        public Id(String id, String sourceId) {
            mEntryId = id;
            mAddressBookSourceId = sourceId;
            if (sourceId != null) {
                mUniqueId = id + ":" + sourceId;
            } else {
                mUniqueId = id;
```

```
            }
        }


        @Override
        public String getUniqueId() {
            return mUniqueId;
        }


        /**
         * Returns entry ID.
         *
         * @return Entry id
         */
        public String getEntryId() {
            return mEntryId;
        }


        /**
         * Get the address book source ID.
         *
         * @return the address book source id or <code>null</code> if not
specified.
         */
        public String getAddressBookSourceId() {
            return mAddressBookSourceId;
        }


        @Override
        public String toString() {
            return mUniqueId;
        }


        @Override
        public int hashCode() {
            return Objects.hash(mUniqueId, mAddressBookSourceId);
        }


        @Override
        public boolean equals(Object id) {
            if (id == null) {
                return false;
            }


            if (id == this) {
                return true;
```

```
            }


            if (id instanceof Id) {
                Id entryId = (Id) id;
                if (entryId.getUniqueId().equals(mUniqueId)) {
                    return true;
                }
            }
            return false;
        }
    }
}
```

MyAddressBookCriteriaBean class

```
package com.axway.st.plugins.abcustomsource.sdk.file.criterion;


import com.axway.st.plugins.absource.AddressBookCriteria;
import com.axway.st.plugins.absource.AddressBookEntry.EntryId;
import com.axway.st.plugins.absource.AddressBookEntry.Type;
import com.axway.st.plugins.absource.AddressBookProviderOrder;


/**
 * Local (BU) address book criteria class.
 */
public class MyAddressBookCriteriaBean implements AddressBookCriteria{


    /** Display Name. */
    private String mDisplayName;


    /** ID. */
    private EntryId mId;


    /** The email. */
    private String mEmail;


    /** The parent group. */
    private String mParentGroup;


    /** The search for.*/
    private String mSearchFor;
```

```
    /** The type.*/
    private Type mType;


    /** The order. */
    private AddressBookProviderOrder mOrder = AddressBookProviderOrder.DISPLAY_
NAME_ASC;


    /**
     * Constructor.
     */
    public MyAddressBookCriteriaBean() {
    }


    /**
     * Constructor.
     * @param beanClass Bean class
     * @param alias Cache Alias
     */
    public MyAddressBookCriteriaBean(Class<?> beanClass, String alias) {
    }


    @Override
    public AddressBookCriteria entryId(EntryId entryId) {
        mId = entryId;
        return this;
    }


    @Override
    public AddressBookCriteria displayName(String displayName) {
        mDisplayName = displayName;
        return this;
    }


    @Override
    public AddressBookCriteria email(String email) {
        mEmail = email;
        return this;
    }


    @Override
    public AddressBookCriteria parentGroup(String parentGroup) {
        mParentGroup = parentGroup;
        return this;
    }
```

```
    @Override
    public AddressBookCriteria type(Type type) {
        mType = type;
        return this;
    }


    @Override
    public AddressBookCriteria searchFor(String searchFor) {
        mSearchFor = searchFor;
        return this;
    }


    @Override
    public AddressBookCriteria orderByDisplayName(boolean ascending) {
        mOrder = ascending?AddressBookProviderOrder.DISPLAY_NAME_
ASC:AddressBookProviderOrder.DISPLAY_NAME_DESC;
        return this;
    }


    @Override
    public AddressBookCriteria orderByEmail(boolean ascending) {
        mOrder = ascending?AddressBookProviderOrder.EMAIL_
ASC:AddressBookProviderOrder.EMAIL_DESC;
        return this;
    }


    @Override
    public String getDisplayName() {
        return mDisplayName;
    }

    @Override
    public void setDisplayName(String displayName) {
        mDisplayName = displayName;
    }


    @Override
    public EntryId getId() {
        return mId;
    }


    @Override
    public void setId(EntryId id) {
        mId = id;
    }
```

```
@Override
public String getEmail() {
    return mEmail;
}


@Override
public void setEmail(String email) {
    mEmail = email;
}


@Override
public String getParentGroup() {
    return mParentGroup;
}


@Override
public void setParentGroup(String parentGroup) {
    mParentGroup = parentGroup;
}


@Override
public String getSearchFor() {
    return mSearchFor;
}


@Override
public void setSearchFor(String searchFor) {
    mSearchFor = searchFor;
}


@Override
public Type getType() {
    return mType;
}


@Override
public void setType(Type type) {
    mType = type;
}


@Override
public AddressBookProviderOrder getOrder() {
    return mOrder;
```

```
    }


    @Override
    public void setOrder(AddressBookProviderOrder order) {
        mOrder = order;
    }
}
```

MyAddressBookContactComparator class

```
package com.axway.st.plugins.abcustomsource.sdk.file.criterion;


import java.util.Comparator;


import com.axway.st.plugins.absource.AddressBookEntry;
import com.axway.st.plugins.absource.AddressBookProviderOrder;


/**
* LDAP AddressBookContactComparator.
*
*/
public class MyAddressBookContactComparator implements
Comparator<AddressBookEntry> {


    /**
     * Order.
     */
    private AddressBookProviderOrder mOrder;


    /**
     * Constructor.
     * @param order Order
     */
    public MyAddressBookContactComparator(AddressBookProviderOrder order) {
        mOrder = order;
    }


    @Override
    public int compare(AddressBookEntry contact1, AddressBookEntry contact2) {
        int res = compareContactField(mOrder, contact1, contact2, 0);
        return res;
    }
```

```
    /**
     * Compares fields.
     * @param order Order.
     * @param contact1 Item 1.
     * @param contact2 Item 2.
     * @param defValue Default value
     * @return compare index
     */
    private int compareContactField(AddressBookProviderOrder order,
AddressBookEntry contact1, AddressBookEntry contact2, int defValue) {
        String field1 = null;
        String field2 = null;
        if (order == null) {
            return defValue;
        }
        switch (order) {
        case EMAIL_ASC:
        case EMAIL_DESC:
            field1 = contact1!=null?contact1.getEmail():null;
            field2 = contact2!=null?contact2.getEmail():null;
            break;
        default:
            field1 = contact1!=null?contact1.getDisplayName():null;
            field2 = contact2!=null?contact2.getDisplayName():null;
        }
        int res;
        if (field1 == null && field2 == null) {
            res = 0;
        } else if (field2 == null) {
            res = 1;
        } else if (field1 == null) {
            res = -1;
        } else {
            res = field1.compareToIgnoreCase(field2);
        }


        return order.isAscending() ? res : -res;
    }
}
```

Pluggable authentication 7

This section describes the capability of SecureTransport Server to use custom authentication plug-ins for executing custom authentication logic and provides instructions on how to implement and configure a Custom Authenticator.

During the installation or upgrade of SecureTransport 5.3.7 or later, a dedicated folder for custom authentication plug-ins is created:

```
${FILEDRIVE_HOME}/plugins/authentication
```

You need to upload a jar file in this folder for every authentication plug-in that is implemented and configured in SecureTransport. The jar file contains all the information required for the custom authenticator including the specific configuration metadata.

In general, an authentication plug-in is a custom plug-in that can be plugged into SecureTransport Server and used as a built-in SecureTransport authenticator.

In SecureTransport 5.4, the Pluggable Authentication Service Provider Interface (SPI) has been exposed to allow developers to implement custom authentication logic for authenticating end-users and administrators.

The SecureTransport Pluggable Authentication SPI is a set of Java interfaces and services used to create, configure, and register custom authenticators to a SecureTransport Server.

Note To integrate a custom authentication plug-in for administrators and end-users refer to the Administrator Guide.
Only one authentication plug-in can be enabled at a time. Enabling more than one authentication plug-in will result in errors.

Authenticator

SecureTransport currently supports the following authentication methods:

- Basic authentication – username and password
- Certificate authentication
- Dual-authentication – certificate + basic authentication

The authenticator is a tool which adopts one of the following authentication methods. Authentication plug-in can have one or more authenticators which can be plugged in SecureTransport to execute a custom authentication. It authenticates users based on input data – username, password, and certificate. Upon successful authentication, the authenticator must provide user parameters that will be used by SecureTransport – username, email, home folder, UID, GID and additional parameters.

Note Email, home folder, UID, GID are not mandatory for virtual accounts and accounts through templates but are all mandatory for external accounts The username is mandatory for all above-listed types of accounts .

If the authentication is not successful, the user is not granted access to SecureTransport. If the authenticator does not support or recognize the input data, it must forward the authentication process to another authenticator that may successfully consume it.

Authentication Plug-in Implementation

To support any of the above authentication methods, a plug-in must implement at least one of the following interfaces. The SecureTransport authentication SPI provides two interfaces for implementation – `BasicAuthenticator` and `CertificateAuthenticator`. To support the Basic authentication method, you must provide class, implementing `BasicAuthenticator` interface. To support the certificate authentication method, a class implementation of the `CertificateAuthenticator` interface is required. To support dual-authentication, you must provide implementation of both interfaces.

Implement a Basic Authentication method

Create a class that implements the *BasicAuthenticator* interface. It must have a public default constructor and implement the authenticate method, declared in the interface.

```
public class BasicAuthenticatorImpl implements BasicAuthenticator {

public BasicAuthenticatorImpl() {
        }

@Override
public AuthenticationResult authenticate(
       BasicAuthenticationRequest authRequest) {

        }
}
```

An object of this class will be instantiated and its authenticate method will be called by SecureTransport, every time a user tries to log in with a username and a password. The *authenticate* method's body is the authentication logic.

BasicAuthenticationRequest is an interface of a bean that provides the following properties:

- Username and password – the log-in credentials
- Protocol – the SecureTransport daemon in which the user tries to log in – HTTP, FTP, SSH or ADMIN
- Remote host address and host name – the IP and the host name of user's machine
- Edge host address – the IP of the daemon's machine
- The ID of the edge where the daemon is located in case of Streaming configuration
- Additional parameters – additional attributes provided by SecureTransport (e.g. HTTP headers if the protocol is HTTP)

AuthenticationResult is an interface of a bean declaring the properties your return object must provide:

- Exit code
  - **SUCCESS** - in case of successful authentication
  - **FAILURE** - in case of unsuccessful authentication (denying access to the user)
  - **CONTINUE** - the implementation doesn't support the provided data in the request or doesn't recognize the user; the authenticated process will be delegated to other authenticators
- Message: Human-readable information about the authentication process.
- Result: An object of a *UserData* type in case of successful authentication

UserData is an interface of a bean declaring the properties you must provide – *username, UID, GID, email, homeDir* and additional attributes. When authenticating administrator users (protocol: ADMIN) only the username is required, but when authenticating end users, who are not mapped to a virtual accounts or account templates in SecureTransport (real users) *homeDir, UID and GID* are also required.

Implement a Certificate Authentication method

Create a class that implements the *CertificateAuthenticator* interface. It must have a public default constructor and implement the authenticate method, declared in the interface.

```
public class CertificateAuthenticatorImpl implements CertificateAuthenticator {

public CertificateAuthenticatorImpl() {
        }

@Override
public AuthenticationResult authenticate(
       CertificateAuthenticationRequest authRequest) {
        }
}
```

An object of this class will be instantiated and its authenticate method will be called by SecureTransport every time a user tries to log in using a certificate. The authenticate method executes the custom authentication logic.

CertificationAuthenticationRequest is an interface of a bean that declares the following properties:

- Certificate – string representation of the certificate used by the user to log in; it is Base64 encoded with attached "---BEGIN CERTIFICATE---" and "---END CERTIFICATE---" tags.

Note If the `protocol` is SSH, the provided `certificate` is actually a Base64 encoded SSH authentication key. SSH authentication also provides a log-in name, that could be found in the additional attributes of the request with key LOGIN_NAME.

- Protocol, Remote host address and host name, Edge host address and ID, Additional parameters are described in the previous section

AuthenticationResult interface is described in the previous section.

Support for a Dual-authentication method

In order to support dual-authentication, the plug-in has to provide implementations of both interfaces. In case of dual-authentication, the user will be first prompted for certificate. The plug-in certificate authenticator will be called first. If the authentication process is successful, the user will be prompted for a username and a password and the plug-in basic authenticator will be called.

If the custom Certificate authenticator returns code CONTINUE, the authentication process will be delegated to internal SecureTransport authenticators and the Basic authenticator will not be called.

BasicAuthenticationRequest contains one additional field – *certificateAuthenticatedUserData*. This field will be populated while calling the plug-in Basic Authenticator in case of dual-authentication, and will contain the *UserData* with the same field values returned by the plug-in Certificate Authenticator. If the authentication is successful, the *UserData* returned by the Basic Authenticator will be used by SecureTransport

Note Same authenticator classes will be used for Basic Authentication, Certificate authentication and Dual-authentication.

Implement a UserRelease method

Optionally you can provide a class that will be notified by SecureTransport that an object of a *UserData* type will not be used anymore – usually after user logout or user session expiration.

Create a class that implements the *UserReleaseListener* interface which must contain a public default constructor and a release method.

```
public class UserReleaseListenerImpl implements UserRealeaseListener {

public UserReleaseListenerImpl() {
        }

@Override
public void release(UserData userData) {

        }
}
```

An object of this class will be instantiated an its release method will be called each time a user, that has been authenticated by the plug-in

- has logged out
- their session has expired or killed
- failed second step of dual-authentication – the certificate user data will be released

The *userData*'s fields have the same values retrieved by one of the plug-ins' authenticators.

Implement a custom plug-in configuration

To implement a custom plug-in configuration, you must create a class that implements the interface *PluginConfiguration* and conforms to the *JavaBean convention*. The configuration itself is the fields of the class. Additional *initDefault* method must be implemented for the explicit initialization of the property values. An object of this class is instantiated on:

- Plug-in's first discovery and registration – initDefault method is being called, fields are being discovered and added to SecureTransport Server configuration with the default values obtained by the class getters
- Each Authentication call – the values are read from the Server configuration and the objects fields are set using the class setters. The object is then injected in the authenticator class
- User release call – identical to the authentication call

Note Currently the only supported field types are: String, Integer, Boolean.

An additional @*Description* annotation can be provided to every field's getter or setter, describing what the configuration is used for. The description will appear on the SecureTransport Server configuration admin page.

Example:

```
public class PluginConfigurationBean implements PluginConfiguration {

    private String successMessage;

    private String failureMessage;

    public PluginConfigurationBean() {
    }

    @Override
    public void initDefault() {
        successMessage = "HOORAY";
        failureMessage = "ACCESS DENIED";
    }

    @Description(value = "Message, that will be logged in Server Log in case of
successful authentication")
    public String getSuccessMessage() {
        return successMessage;
    }

    public void setSuccessMessage(String successMessage) {
        this.successMessage = successMessage;
    }

    @Description(value = "Message, that will be logged in Server Log in case of
unsuccessful authentication")
```

```
    public String getFailureMessage() {
        return failureMessage;
    }

    public void setFailureMessage(String failureMessage) {
        this.failureMessage = failureMessage;
    }
}
```

An object of this class could be injected in each Authenticator class and User Listener class using the following statement:

```
@Inject
private PluginConfigurationBean mConfiguration;
```

The plugin-authentication-services library also provides an annotation @*Encrypted*. It indicates that a configuration value must be stored encrypted in SecureTransport Server configuration. Put the annotation to one of the field's accessors.

Using the SecureTransport Logging service

To use the logging service, provided by SecureTransport Plug-in Authentication Services library, inject it in your Authenticator or `UserListener` class using the following statement:

```
@Inject
private LoggingService mLogger;
```

The *LoggingService* interface declares functionality for logging messages and exceptions with a different log level – ERROR, WARN, INFO, DEBUG, etc.

Examples:

```
mLogger.info(String.format("User '%s' authenticated successfully.", username));
mLogger.warn(String.format("Authentication of user '%s' failed - wrong
password.", username));
mLogger.warn("Problem while parsing client certificate!", e);
mLogger.debug("Start custom authentication process...");
```

The logging level can be controlled by logger in `tm-log4j.xml` and `admin-log4j.xml`.

Using the SecureTransport Certificate service

SecureTransport provides a service for certificate parsing and validation against its keystore. The service is called `CertificateService` and can be injected in the plug-in's Authenticators and `UserReleaseListener` using the following code:

```
@Inject
private CertificateService mCertificateService;
```

It declares the following methods:

- `CertificateStatus getCertificateStatus(X509Certificate certificate)` – validates the certificate against the SecureTransport keystore; returns one of these enumeration values: `VALID, EXPIRED, NOT_YET_VALID, UNTRUSTED`.
- `KeyPair getKeyPair(String certificateAlias)` – Retrieves public/private key pair from the SecureTransport keystore using certificate alias.
- `X509Certificate parseCertificate(String certificateString)` – converts a certificate string to a `X509Certificate` object; could be used directly with the certificate, obtained from a `CertificateAuthenticationRequest`, if it is a X509 certificate.
- `X509Certificate getCertificateBySshKey(String sshKey)` – retrieves a certificate from the SecureTransport keystore, that the provided `sshKey` corresponds to; could be used directly with the SSH key string, obtained from a `CertificateAuthenticationRequest` in case of a SSH protocol.
- `X509Certificate getCertificateByAlias(String certificateAlias)` – retrieves certificate from the SecureTransport keystore using certificate alias.
- `X509Certificate getCertificateById(String certificateId)` – retrieves certificate from the SecureTransport keystore using certificate's unique identifier.
- `X509Certificate getIssuer(X509Certificate certificate)` – retrieves certificate issuer from the SecureTransport keystore for given certificate.
- `List<X509Certificate> getCertificateChain(X509Certificate certificate)` – retrieves certificate chain list from the SecureTransport keystore for given certificate. Can be empty, if certificate does not have a chain. The first element is the certificate itself, next element is its issuer and so on to the root certificate.
- `Collection<X509Certificate> getCertificates(String subjectDN)` – retrieves certificate list from the SecureTransport keystore for given subject DN.
- `Subject getCertificateSubject(X509Certificate certificate)` – extracts the domain name of the `certificate`'s subject into a bean of type `Subject`, that contains the following properties: `Common name, Country, Organization, Organization Unit, Locality, State`
- `String getFingerprint(X509Certificate certificate, FingerprintAlgorithm algorithm, boolean base64Encoded) throws NoSuchAlgorithmException` - calculates the fingerprint of the supplied X509 certificate using one of the following fingerprint algorithms: `MD5, SHA1,SHA256`.

This service is available in the `plugins-authentication-services` library.

Using the SecureTransport SSLContext service

SecureTransport provides a service for creating `javax.net.ssl.SSLContext` and `javax.net.ssl.SSLSocketFactory` objects using its internal keystore. These objects can be used for setting up secure SSL connections to other applications. This service is identical to the *Certificate service* exposed for *Custom connectors*, see *SecureTransport exposed services* on page 27.

The service interface is called `com.axway.st.plugins.authentication.services.SSLContextService` and can be injected in the plug-in's Authenticators and `UserReleaseListener` class, using the following code:

```
@Inject
private SSLContextService sslContextService;
```

The service methods are:

- `SSLContext createSSLContext(String certId, boolean verifyPeer) throws SecurityException;`
- `SSLContext createSSLContext(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm) throws SecurityException;`
- `SSLSocketFactory configSSLContextFactory(SSLContext sslContext, String[] protocols, String[] cipherSuites) throws SecurityException;`
- `SSLSocketFactory configSSLContextFactory(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm, String[] protocols, String[] cipherSuites) throws SecurityException;`

These methods have the same behavior as those from the Custom connectors' `CertificateService`.

Using the SecureTransport Expression Evaluator service

You can use expression evaluator service to evaluate and validate the expressions used in custom authenticator implementation.

To use the expression evaluator service, declare a variable with `@Inject` annotation to the interface of the expression evaluator service provided in the API:

```
@Inject
```

```
private ExpressionEvaluatorService mExpressionEvaluatorService;
```

This service is identical to the Expression Evaluator service exposed for Custom connectors, see *SecureTransport exposed services* on page 27.

Example BasicAutnenticator implementation

```
public class BasicAuthenticatorImpl implements BasicAuthenticator {

    @Inject
    private LoggingService mLogger;

    @Inject
    private PluginConfigurationBean mConfiguration;

    public BasicAuthenticatorImpl() {
    }

    @Override
    public AuthenticationResult authenticate(BasicAuthenticationRequest
authRequest) {

        mLogger.debug("Start custom authentication process...");

        String username = authRequest.getUsername();

        mLogger.debug("Authenticating...");

        if (!username.startsWith("s")) {

            mLogger.info(String.format("Unknown user '%s'.", username));
            return new AuthenticationResultBean("Unknown user",
               AuthenticationResult.ExitCode.CONTINUE);
        }

        char[] userPass = authRequest.getPassword();
        boolean areNameAndPassEqual = Arrays.equals(username.toCharArray(),
userPass);
        Arrays.fill(userPass, (char) 0);

        if (!areNameAndPassEqual) {
            mLogger.warn(String.format("Authentication of user '%s' failed -
wrong password.", username));
            return new AuthenticationResultBean
(mConfiguration.getFailureMessage(),
                    AuthenticationResult.ExitCode.FAILURE);
        }
```

```
        UserData user;
        UserManager userManager = new UserManager();
        switch (authRequest.getProtocol().toLowerCase()) {
        case "admin":
            user = userManager.getAdmin(username);
            break;
        case "ssh":
        case "ftp":
        case "http":
        default:
            user = userManager.getUser(username);
            break;
        }

        mLogger.info(String.format("User '%s' authenticated successfully.",
username));
        return new AuthenticationResultBean(mConfiguration.getSuccessMessage(),
user,
                AuthenticationResult.ExitCode.SUCCESS);

    }
}
```

This authenticator recognizes only users whose names start with an 's'. If the password matches the username, the user is authenticated successfully. Based on the protocol different user data is returned.

Example CertificateAuthenticator implementation

```
public class CertificateAuthenticatorImpl implements CertificateAuthenticator {

    @Inject
    private LoggingService mLogger;

    @Inject
    private CertificateService mCertificateService;

    public CertificateAuthenticatorImpl() {
    }

    @Override
    public AuthenticationResult authenticate(CertificateAuthenticationRequest
authRequest) {
```

```
        mLogger.debug("Start custom authentication process...");

        String username;
        String protocol = authRequest.getProtocol().toLowerCase();
        try {
            String clientCert = authRequest.getCertificate();
            username = getUserByCertificate(clientCert, protocol);
        } catch (Exception e) {
            mLogger.warn("Problem while parsing client certificate!", e);
            return new AuthenticationResultBean("Unknown certificate",
                    AuthenticationResult.ExitCode.CONTINUE);
        }

        mLogger.debug("Authenticating...");

        if (username.startsWith("s")) {

            UserData user;
            UserManager userManager = new UserManager();
            switch (protocol) {
            case "admin":
                user = userManager.getAdmin(username);
                break;
            case "ssh":
            case "ftp":
            case "http":
            default:
                user = userManager.getUser(username);
                break;
            }

            mLogger.info(String.format("User '%s' authenticated successfully.",
username));
            return new AuthenticationResultBean("Authentication successful",
user,
                    AuthenticationResult.ExitCode.SUCCESS);

        }

        if (username.startsWith("f")) {
            mLogger.warn(String.format("Authentication of user '%s' failed.",
username));
            return new AuthenticationResultBean("Authentication failed",
AuthenticationResult.ExitCode.FAILURE);
        }

        mLogger.info(String.format("Unknown user '%s'.", username));
        return new AuthenticationResultBean("Unknown user certificate",
AuthenticationResult.ExitCode.CONTINUE);
    }
```

```
cateString);

        Subject subject = mCertificateService.getCertificateSubject
(certificate);
        return subject.getCommonName();
    }
```

This authenticator extracts the username from a X509 certificate's subject using the provided by SecureTransport`CertificateService`. It recognizes usernames starting with an 's' or an 'f'. It denies access to the 'f' users and returns different `UserData` objects based on the protocol.

Example UserRelease implementation

```
public class UserReleaseListenerImpl implements UserRealeaseListener {

    @Inject
    private LoggingService mLogger;

    public UserReleaseListenerImpl() {
    }

    @Override
    public void release(UserData userData) {
        String username = userData.getUsername();
        UserManager userManager = new UserManager();
        if (!userManager.isOurUser(userData)) {
            mLogger.error(String.format("Unknown user '%s' send for release!",
               username));
            return;
        }

        //some release logic

        mLogger.info(String.format("User '%s' successfully released.",
username));
    }
}
```

Objects of this class execute some release logic and log a success message in the SecureTransport Server log. They also verify that the *userData* has been created by the plug-in.

The above sample implementations along with the classes they use, like (UserManager), will be delivered as source files along with the `plugin-authentication-spi.jar` and `securetransport-plugins-authentication-services.jar` in the SecureTransport Software Development Kit.

Note The delivered plug-in is only given as an example and it should not be used in real systems.

Create a plug-in metadata file

A `META-INF/MANIFEST.MF` file must be created in the plug-in jar file. This metadata file is used by SecureTransport to register and use the Custom Authentication implementation as a regular SecureTransport internal Authenticator.

Note The `MANIFEST.MF` file is used with SPI version 1.0 only. For later versions of SPI, you will need to use the `Plugin-Info.yaml` file, also located in the `META-INF` directory.

It must contain the following properties:

Property	Value	
`Basic-Authenticator-Class`	The full name of the class that implements the `BasicAuthenticator` interface.	One of these is required.
`Certificate-Authenticator-Class`	The full name of the class that implements the `CertificateAuthenticator` interface.	
`User-Release-Listener-Class`	The full name of the class that implements the `UserReleaseListener` interface.	Optional
`Plugin-Configuration-Class`	The full name of the class that implements the `PluginConfiguration` interface.	Optional
`Plugin-Label`	The label (name) of the authentication plug-in.	Required
`Class-Path`	The `Class-Path` property is useful for specifying the list of the external libraries your custom connector needs to successfully run.	Optional

If your Custom Authentication implementation depends on third party libraries, you have two implementation approaches:

- You can use a fat jar file that stores your classes and the classes from all dependent jars into a one single jar file.
- Add a Class-Path attribute to the `MANIFEST.MF` file, in which you must enumerate your dependent jars separated by spaces or tabs. SecureTransport will add the enumerated libraries to the Custom Authentication implementation classpath. See https://docs.oracle.com/javase/tutorial/deployment/jar/downman.html

Note When you are deploying your Plug-in Authentication implementation, you should only include third party dependencies, used by your implementation. You must not add APIs provided run-time by SecureTransport. This includes the SecureTransport Pluggable Authentication SPI, Plug-in Services API and JSR-330 (javax.inject) API.

Authentication SPI versioning

Each Authentication SPI is versioned. The initial SPI release is version 1.0. Every version extends the previous one and adds additional content. The SPI methods and classes added in a specific version have the **@since** annotation in their Javadoc documentation.

To declare what SPI version is used in the Authentication implementation, place the `Plugin-Info.yaml` file inside the `Meta-Inf` directory. This file contains the plug-in information, similar to the `MANIFEST.MF` file in Authentication SPI version 1.0. The description schema used in the yaml file is:

```
'<SPI version>':
  <property>: <value>
  <property>: <value>
  ..................
'<SPI version>':
  <property>: <value>
  <property>: <value>
  ..................
```

The convention for the indentation used is two spaces for each block.

The `Plugin-Info.yaml` file is introduced in Authentication SPI version 1.1. The old `MANIFEST.MF` file (if any) is associated with version 1.0 and is with lower priority when there is a `Plugin-Info.yaml` file.

If the version is not specified, the default one (1.0) will be used.

Only the methods and classes available in the specified version can be used.

The current SPI version, introduced along with SecureTransport 5.4 Patch 18, is **1.2**.

All previous Authentication SPI versions must be compatible with the latest SecureTransport version, unless stated otherwise.

Plug-in deployment

Read the *Authentication->Pluggable authentication* section in the SecureTransport Administrator Guide to:

- Deploy plug-ins
- Configure plug-ins
- Configure pluggable authentication

Enable Logger service with authentication

In order to enable logging when using the Logger service, edit the `com.axway.st.plugins` loggers in the `tm-log4j.xml` or `admin-log4j.xml` file. For example, with pluggable authentication:

```
<logger name="com.axway.st.plugins.authentication" additivity="false">
        <level value="info" />
        <appender-ref ref="ServerLog" />
</logger>
```

For fine-tuning debug logging, add the plug-in name in conjunction with the `com.axway.st.plugins` logger.

In this case, use `com.axway.st.plugins.authentication.<plugin_name>` as shown on the example:

```
<logger name="com.axway.st.plugins.authentication.authentication-plugin"
additivity="false">
        <level value="debug" />
        <appender-ref ref="ServerLog" />
</logger>
```

Pluggable authorization 8

This section describes the capability of SecureTransport Server to use custom authorization plug-ins for executing custom authorization logic and provides instructions on how to implement and configure such a plug-in.

When installing or upgrading SecureTransport 5.4 and later, an additional folder is created for the deployment of the Custom Authorization plug-ins – '`${FILEDRIVEHOME}/plugins/authorization`'.

The customer must upload a JAR file in the above directory for every Authorization plug-in that is implemented and configured in SecureTransport. The JAR file contains all the information required for the custom authorization, including the specific configuration metadata.

The SecureTransport Pluggable Authorization SPI is a set of Java interfaces and services used to create, configure, and register custom *authorizers* in a SecureTransport Server. It is exposed from SecureTransport 5.4 onwards.

The SPI allows customer-specific custom plug-ins to be implemented, plugged into SecureTransport Server and used as a built-in SecureTransport authorizer. The customer authorizers will be applied for end users.

Note Only one authorization plug-in can be enabled at a time with SecureTransport version 5.4 or later. Enabling more than one authorization plug-in will result in errors.

SecureTransport currently supports the following types of custom authorization:

- Upload a file
- Download a file
- List content of a directory
- Change permissions (file or directory)
- Rename a file
- Delete a file
- Create a directory
- Delete a directory
- Change the working directory

All of the above operations are defined by the `CustomAuthorizer` interface from the Authorization SPI.

Additionally, one more operation is supported - file filtering operation defined by `CustomFileFIlter` interface from the Authorization SPI.

Create a plug-in metadata file

`A META-INF/MANIFEST.MF` file must be created in the plug-in's JAR file. This metadata file is used by SecureTransport to register and use the Custom Authorization implementation as a regular SecureTransport internal Authorizer.

It can contain the following properties:

Property	Value	
Authorizer-Class	The full name of the class that implements the `CustomAuthorizer` interface.	Required
Custom-File-Filter-Class	The full name of the class that implements the `CustomFileFilter` interface. Requires `CustomAuthorizer` interface implementation in order to be used.	Optional
Plugin-Configuration-Class	The full name of the class that implements the `PluginConfiguration` interface.	Optional
Plugin-Label	Name of the custom plug-in.	Required

If your Custom Authorization implementation depends on third party libraries, you have two implementation approaches:

1. You can use a fat JAR file that stores your classes and the classes from all dependent JARs into one single JAR file.
2. Add a Class-Path attribute to the `MANIFEST.MF` file, in which you must enumerate your dependent JARs separated by spaces or tabs. SecureTransport will add the enumerated libraries to the Custom Authorization implementation class path. See https://docs.oracle.com/javase/tutorial/deployment/jar/downman.html

Note When you are deploying your Plug-in Authorization implementation, you must only include third party dependencies, used by your implementation. You must not add APIs provided run-time by SecureTransport. This includes the SecureTransport Pluggable Authorization SPI, Plug-in Services API and JSR-330 (`javax.inject`) API.

Authorization SPI versioning

Each Authorization SPI is versioned. The initial SPI release is version 1.0. Every version extends the previous one and adds additional content. The SPI methods and classes added in a specific version have the **@since** annotation in their Javadoc documentation.

To declare what SPI version is used in the Authorization implementation, place the `Plugin-Info.yaml` file inside the `Meta-Inf` directory. This file contains the plug-in information, similar to the `MANIFEST.MF` file in Authorization SPI version 1.0. The description schema used in the yaml file is:

```
'<SPI version>':
  <property>: <value>
  <property>: <value>
  ..................
'<SPI version>':
  <property>: <value>
  <property>: <value>
  ..................
```

The convention for the indentation used is two spaces for each block.

The `Plugin-Info.yaml` file is introduced in Authorization SPI version 1.1. The old `MANIFEST.MF` file (if any) is associated with version 1.0 and is with lower priority when there is a `Plugin-Info.yaml` file.

If the version is not specified, the default one (1.0) will be used.

Only the methods and classes available in the specified version can be used.

The current SPI version is **1.3**.

All previous Authorization SPI versions must be compatible with the latest SecureTransport version, unless stated otherwise.

Implement a CustomAuthorizer

To support any authorization operation, a plug-in must implement `CustomAuthorizer` interface. Create a class that implements the `CustomAuthorizer` interface. It must have a public default constructor and implement the declared methods which are the authorization operations.

All methods accept a `DataEnvironment` as a parameter and must return `AuthorizationResult`.

`DataEnvironment` is an interface that defines the current SecureTransport data for a single authorization operation. It consists of:

- AccountDataEnv
  - account.attributes
  - account.deliveryMethod
  - account.disabled
  - account.email
  - account.enrollment
  - account.implicitEnrollment

- account.htmlTemplate
- account.id
- account.name
- account.notes
- account.phone
- account.type
- account.homeDir
- account.buzinessUnitName
- account.userData
  - user.loginName
  - user.type
  - user.uid
  - user.gid
  - user.className
  - user.nativeUserName

- SessionDataEnv
  - session.protocol
  - session.workDir
  - session.workDirFull
  - session.remoteAddress
  - session.remoteHost
  - session.streamingClient
  - session.isSSL
  - session.isPull
  - session.routingInitiatedBy
  - session.userLoginType
  - session.edgeId
  - session.edgeHostAddress
- FlowDataEnv
  - flow.attributes
- HttpDataEnv
  - http.headers

- TransferDataEnv
  - transfer.targetDir
  - transfer.targetFull
  - transfer.target
  - transfer.targetDirFull
  - transfer.transferredBytes
  - transfer.startTime
  - transfer.xferType
  - transfer.umask
  - transfer.parentCycleId
  - transfer.coreId
  - transfer.targetRenameTo
  - transfer.targetRenameFrom
- AuthenticationDataEnv
  - authn.ldapData
    - ldap.domainId
    - ldap.domainName
    - ldap.dn
    - ldap.authByEmail
    - ldap.uid
    - ldap.gid
    - ldap.homeDir
    - ldap.shell
    - ldap.userType
    - ldap.sysUser
    - ldap.attributes
  - authn.pluginData
    - plugin.uid
    - plugin.gid
    - plugin.homeDir
    - plugin.pluginName
    - plugin.userName
    - plugin.email
    - plugin.attributes

- authn.ssoData
  - sso.uid
  - sso.gid
  - sso.email
  - sso.idpld
  - sso.tenant
  - sso.userName
  - sso.homeDir
  - sso.attributes

`AuthorizationResult` is an interface that defines the result an authorization operation must return. It provides the following properties:

- Exit code
  - `AUTHORIZATION_CONTINUE` – in case of successful authorization or in case of default behavior
  - `AUTHORIZATION_DENY` – in case of unsuccessful authorization
- Message: Human - readable information about the executed authorization operation

Note Make sure that you will not return null as an `AuthorizationResult`. In that case the authorization action will be denied.

Put your authorization logic inside and return appropriate result for the methods you want to implement.

If you don't need authorization of particular operations, you can simply implement them to return a value of `AuthorizationResult.ExitCode.AUTHORIZATION_CONTINUE`.

Your custom authorizer implementation will be able to use two SecureTransport services – `LoggingService` and `StfsService`. Both of them can be used by injecting them as class members using the standard java injection annotation (see the example below). The logging service provides logging capabilities specified by your administrator. The STFS service provides a way to read the basic filesystem attributes that are stored for a files transferred by SecureTransport.

For more information about the possible SecureTransport-specific file attributes, please refer to `StSpecificFileAttributes` interface in `securetransport-program-api`.

Example of a CustomAuthorizer implementation

Below is the example of how you can implement the `CustomAuthorizer` interface.

```
import javax.inject.Inject;
public class SimpleCustomAuthorizer implements CustomAuthorizer {
@Inject
private LoggingService LOG;
```

```
private boolean isAccountEligible(AccountType type) {
return type == AccountType.VIRTUAL || type == AccountType.TEMPLATE;
}
private AuthorizationResult getResultInstance(final boolean accountEligible) {
return new AuthorizationResult() {
@Override
public ExitCode getExitCode() {
return accountEligible==true ? ExitCode.AUTHORIZATION_
CONTINUE:ExitCode.AUTHORIZATION_DENY;
}
@Override
public String getMessage() {
return null;
}
};
}
@Override
public AuthorizationResult authorizeUpload(DataEnvironment dataEnvironment) {
LOG.info("Start authorization process for UPLOAD operation.");
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
@Override
public AuthorizationResult authorizeDownload(DataEnvironment dataEnvironment)
{
LOG.info("Start authorization process for DOWNLOAD operation.");
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
@Override
public AuthorizationResult authorizeList(DataEnvironment dataEnvironment) {
LOG.info("Start authorization process for LIST operation.");
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
@Override
public AuthorizationResult authorizeChmod(DataEnvironment dataEnvironment) {
LOG.info("Start authorization process for CHMOD operation.");
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
@Override
public AuthorizationResult authorizeDeleteFile(DataEnvironment
dataEnvironment) {
LOG.info("Start authorization process for DELETE_FILE operation.");
```

```
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
@Override
public AuthorizationResult authorizeRenameFile(DataEnvironment
dataEnvironment) {
LOG.info("Start authorization process for RENAME_FILE operation.");
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
@Override
public AuthorizationResult authorizeCreateDirectory(DataEnvironment
dataEnvironment) {
LOG.info("Start authorization process for CREATE_DIRECTORY operation.");
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
@Override
public AuthorizationResult authorizeDeleteDirectory(DataEnvironment
dataEnvironment) {
LOG.info("Start authorization process for DELETE_DIRECTORY operation.");
final boolean accountEligible = isAccountEligible
(dataEnvironment.getAccountDataEnv().getType());
return getResultInstance(accountEligible);
}
}
```

Implement a CustomFileFilter

In order a plug-in to support file filter operation it must implement `CustomFileFilter` interface. Create a class that implements the `CustomFileFilter` interface. It must have a public default constructor and implement the declared method.

The method `doFilter` that must be implemented accepts 2 parameters – sorted set of Path objects and a `DataEnvironment` object. Its return type is set of Path objects which will be the filtered result.

Your file filter implementation will be able to use two SecureTransport services – `LoggingService` and `StfsService`. Both of them can be used by injecting them as class members using the standard java injection annotation (see the example below). The logging service provides logging capabilities specified by your administrator. The STFS service provides a way to read the basic filesystem metadata attributes that are stored for a files transferred by SecureTransport.

All custom file filter implementations are restricted to return no more than the target files (or part of them) which are going to be filtered. Therefore, only file deletions from the set are supported. Any experiment in adding any files to the returned set will result in a non - filtered output (like there's no file filter implementation) in SecureTransport and an error.

Note If, for some reasons, the `doFilter` return null, no filter will be applied, and the original files will be listed.

Note Do not try to add files in the `targetFiles`. This will cause exception and the File Filter process will fail and no custom file filtering will be applied.

Example of a CustomFileFilter implementation

```
import javax.inject.Inject;
public class CustomFileFilterImpl implements CustomFileFilter {
/**
* Provides logging service.
*/
@Inject
private LoggingService LOG;
/**
* Provides filesystem attributes retrieval service.
*/
@Inject
private StfsService stfsService;
@Override
public SortedSet<Path> doFilter(SortedSet<Path> targetFiles, DataEnvironment
dataEnvironment) {
mLogger.debug("Applying custom filter...");
AccountType accountType = dataEnvironment.getAccountDataEnv().getType();
SortedSet<Path> filteredFiles = new TreeSet<>();
if(accountType != AccountType.VIRTUAL) {
LOG.warn("Custom filter is applied only for virtual users.");
return targetFiles;
}

for (Path path : targetFiles) {
// Some custom filter logic.
Map<String, Object> stfsAttributes = stfsService.getAttributes(path);
if (stfsAttributes.isEmpty()) {
return;
}
StringBuilder sb = new StringBuilder("The STFS attributes for ")
.append(path.toString())
.append(" are: ");
for (Map.Entry<String, Object> stfsAttribute : stfsAttributes.entrySet()) {
sb.append("[")
.append(stfsAttribute.getKey())
.append(" : ")
.append(stfsAttribute.getValue())
.append("]");
}
```

```
logger.info(sb.toString());
}
mLogger.info("Custom filter applied successfully.");
return filteredFiles;
}
}
```

Performance considerations

Below you can find information on how often the authorization plug-in is called depending on some of the supported protocols:

Over FTPS:

- For an upload operation the plug-in is called once to authorize the upload event.
- For a download operation the plug-in is called once to authorize the download event.

Over SFTP:

- For an upload operation the plug-in is called four times - once to authorize the upload operation and three times for the list operation.
- For a download operation the plug-in is called once to authorize the download event.

Over HTTPS:

- For an upload operation the plug-in is called once to authorize the upload event.
- For a download operation the plug-in is called once to authorize the download event.

Note These results may vary depending on the client being used.

Using the SecureTransport Certificate service

SecureTransport provides a service for certificate parsing and validation against its keystore. The service is called `CertificateService` and can be injected in the plug-in's `CustomAuthorizer` and `CustomFileFilter` using the following code:

```
@Inject
private CertificateService mCertificateService;
```

It declares the following methods:

- `CertificateStatus getCertificateStatus(X509Certificate certificate)` – validates the certificate against the SecureTransport keystore; returns one of these enumeration values: `VALID, EXPIRED, NOT_YET_VALID, UNTRUSTED`.
- `KeyPair getKeyPair(String certificateAlias)` – Retrieves public/private key pair from the SecureTransport keystore using certificate alias.

- `X509Certificate getCertificateByAlias(String certificateAlias)` – retrieves certificate from the SecureTransport keystore using certificate alias.
- `X509Certificate getCertificateById(String certificateId)` – retrieves certificate from the SecureTransport keystore using certificate's unique identifier.
- `X509Certificate getIssuer(X509Certificate certificate)` – retrieves certificate issuer from the SecureTransport keystore for given certificate.
- `List<X509Certificate> getCertificateChain(X509Certificate certificate)` – retrieves certificate chain list from the SecureTransport keystore for given certificate. Can be empty, if certificate does not have a chain. The first element is the certificate itself, next element is its issuer and so on to the root certificate.
- `Collection<X509Certificate> getCertificates(String subjectDN)` – retrieves certificate list from the SecureTransport keystore for given subject DN.
- `Subject getCertificateSubject(X509Certificate certificate)` – extracts the domain name of the `certificate`'s subject into a bean of type `Subject`, that contains the following properties: `Common name, Country, Organization, Organization Unit, Locality, State`

Using the SecureTransport SSLContext service

SecureTransport provides a service for creating `javax.net.ssl.SSLContext` and `javax.net.ssl.SSLSocketFactory` objects using its internal keystore. These objects can be used for setting up secure SSL connections to other applications. This service is identical to the *Certificate service* exposed for *Custom connectors*, see *SecureTransport exposed services* on page 27.

The service interface is called `com.axway.st.plugins.authorization.services.SslContextService` and can be injected in the plug-in's `CustomAuthorizer` and `CustomFileFilter` class, using the following code:

```
@Inject
private SsLContextService sslContextService;
```

The service methods are:

- `SSLContext createSSLContext(String certId, boolean verifyPeer) throws SecurityException;`
- `SSLContext createSSLContext(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm) throws SecurityException;`
- `SSLSocketFactory configSSLContextFactory(SSLContext sslContext, String[] protocols, String[] cipherSuites) throws SecurityException;`

- `SSLSocketFactory configSSLContextFactory(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm, String[] protocols, String[] cipherSuites)` **`throws`** `SecurityException;`

These methods have the same behavior as those from the Custom connectors' `CertificateService`.

Using the SecureTransport Expression Evaluator service

You can use expression evaluator service to evaluate and validate the expressions used in custom authenticator implementation.

To use the expression evaluator service, declare a variable with `@Inject` annotation to the interface of the expression evaluator service provided in the API:

```
@Inject
private ExpressionEvaluatorService mExpressionEvaluatorService;
```

This service is identical to the *Expression Evaluator* service exposed for Custom connectors, see *SecureTransport exposed services* on page 27.

Enable Logger service with authorization

In order to enable logging when using the Logger service, edit the `com.axway.st.plugins` loggers in the `tm-log4j.xml` file. For example, with pluggable authorization:

```
<logger name="com.axway.st.plugins.authorization" additivity="false">
        <level value="info" />
        <appender-ref ref="ServerLog" />
</logger>
```

For fine-tuning debug logging, add the plug-in name in conjunction with the `com.axway.st.plugins` logger.

In this case, use `com.axway.st.plugins.authorization.<plugin_name>` as shown on the example:

```
<logger name="com.axway.st.plugins.authorization.authorization-plugin"
additivity="false">
        <level value="debug" />
        <appender-ref ref="ServerLog" />
```

```
</logger>
```

Custom Advanced Routing step 9

This document explains how to develop a custom Advanced Routing step and plug it into SecureTransport.

When installing or upgrading SecureTransport 5.4 or above, a new directory is created for the deployment of the Custom Advanced Routing step plug-ins – `${FILEDRIVEHOME}/plugins/routingSteps`.

The customer may upload a JAR file in this folder for every Advanced Routing step plug-in that is implemented. The JAR file must contain all the information required for the custom step including its metadata.

A SecureTransport Pluggable Advanced Routing SPI – a set of Java interfaces and services used to create, configure, and register a routing step - is exposed to allow developers to implement custom steps.

Note: The SecureTransport Plug-in Advanced Routing step SPI described in this Developer's Guide is version 1.0.0-33.

Java artifacts required for compiling the project

The artifacts required for compilation are:

- `org.hibernate:hibernate-validator: 4.3.2.Final`
- `javax.validation:validation-api:1.0.0.GA`
- `javax:juel: 2.1.0`

Structure of the JAR file containing the Advanced Routing step

```
JAR ROOT
       com/axway/st/server/route/core/component/customstep/
               CustomStepArchiveBean.class
               CustomArchiveStepProducer.class

       messages/AdvancedRoutingMessages.xml
       html/customArchiveStep.html

       META-INF/
               MANIFEST.MF
```

```
                services/
                        com/axway/st/server/route/step/metadata/CustomArchiveStep.xml
```

Configuration resources

Main configuration resource file

The file `/META-INF/services/com/axway/st/server/route/step/metadata/CustomArchiveStep.xml` has the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
       <routeStepMetadata>
               <stepType>CustomArchiveStep</stepType>
               <stepCategory>Transformation</stepCategory>
               <stepDisplayName>Custom Archive Step</stepDisplayName>
               <endpointSchema>st.customarchivestep</endpointSchema>
               <uiPagePath>customStep.html</uiPagePath>

<stepPropertyBean>com.axway.st.server.route.core.component.customstep.CustomArc
hiveStepBean</stepPropertyBean>

<stepProducer>com.axway.st.server.route.core.component.customstep.CustomArchive
StepProducer</stepProducer>
       </routeStepMetadata>
```

Where:

- `stepType` must be unique for the different steps. It is used for distinguishing the different steps in the SecureTransport Rest API.
- `stepCategory` is the category of the step (`Transformation` or `Routing`).
- `stepDisplayName` is the way the step is displayed in the SecureTransport UI.
- `endpointSchema` must be unique for the different steps. It is used as internal ID of the custom steps.
- `stepPropertyBean` is the bean representation of the step custom properties. It is used for validation of the properties when creating the step.
- `stepProducer` implements the actual logic of the step for the transformations/routing of the files.

The name of the step configuration XML file must be unique across the different steps.

Custom logs resource file

The file `/messages/AdvancedRoutingMessages.xml` contains custom messages which are used by the step.

The format is:

```
<?xml version="1.0" encoding="UTF-8"?>
    <messages>
        ...
        <message>
            <id>SCCS0000</id>
            <code>ARCS0000</code>
            <text>General error while executing CustomStep step.</text>
            <description>General error while executing CustomStep
step.</description>
        </message>
        <message>
            <id>SCCS0001</id>
            <code>ARCS0001</code>
            <text>Custom message with %s placeholders.</text>
            <description>Custom message for Custom Step.</description>
        </message>
        ...
    </messages>
```

The `ID` must be unique. The format used for the new `ID`s is as follows:

```
SCCS0000
```

Where:

- first two characters `SC` are an identifier
- followed by two characters for the step abbreviation (examples are `LE` for Line Ending, `CO` for Compress, `CS` for Custom Step, etc.)
- followed by four decimal digits for numbering in ascending order, starting from `0000` for each step.

Custom logging used by the Advanced Routing

The custom steps can use the defined messages in the `/messages/AdvancedRoutingMessages.xml` file of the custom step JAR.

Example:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.axway.st.plugins.improvedrouting.logger.ImmutableMsg;
import com.axway.st.plugins.improvedrouting.logger.MessagesHolder;
import com.axway.st.plugins.improvedrouting.logger.MessagesFactory;

// ...

Logger sLogger = LoggerFactory.getLogger(SomeClass.class);
MessagesHolder messagesFactory = MessagesFactory.getInstance();

// Status code as defined in AdvancedRoutingMessages.xml
sLogger.error(messagesFactory.getMessage("SCCS0001")
        .getFormatedMsg("parameter to replace placeholder"));

sLogger.error(messagesFactory.getMessage("SCCS0000").getText());
```

The log messages will be displayed in the SecureTransport server log with the following format.

```
<code> [user performing the action] [route being executed]: log message
```

Implementation of the step

Bean representation of the Advanced routing step properties

The following bean is used for validation of the step properties values when the step is created or modified.

```
package com.axway.st.server.route.core.component.customstep;

import javax.validation.constraints.NotNull;

/**
 * Bean representation of the custom properties of the step. Each custom step
property
 * has a corresponding property in this class. Uses Bean Validation Framework
in
 * order to validate properties.
 */
public class CustomArchiveStepBean {
```

```
    /** The file filter expression. */
    private String mFileFilterExpression;

    /** The file filter expression type. */
    private String mFileFilterExpressionType;

    /**
     * Default constructor.
     */
    public CustomArchiveStepBean() {
    }

    /**
     * Gets the file filter expression.
     *
     * @return the file filter expression
     */
    @NotNull(message = "File filter cannot be empty.")
    public String getFileFilterExpression() {
        return mFileFilterExpression;
    }

    /**
     * Sets the file filter expression.
     *
     * @param fileFilter
     *            the new file filter expression
     */
    public void setFileFilterExpression(String fileFilter) {
        mFileFilterExpression = fileFilter;
    }

    /**
     * Gets the file filter expression type.
     *
     * @return the file filter expression type
     */
    @NotNull(message = "File filter type cannot be empty.")
    public String getFileFilterExpressionType() {
        return mFileFilterExpressionType;
    }

    /**
     * Sets the file filter expression type.
     *
     * @param filterType
     *            the new file filter expression type
     */
    public void setFileFilterExpressionType(String filterType) {
        mFileFilterExpressionType = filterType;
    }
}
```

File processing

Handles the actual execution of the step.

```
package com.axway.st.server.route.core.component.customstep;

import java.io.BufferedWriter;
import java.nio.charset.Charset;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import
com.axway.st.plugins.improvedrouting.expression.function.RoutingExpressionEvalu
atorWrapper;
import com.axway.st.plugins.improvedrouting.spi.ProducerSettings;
import com.axway.st.plugins.improvedrouting.spi.StProducer;

/**
 * The CustomStepProducer.
 */
public class CustomArchiveStepProducer implements StProducer {

    /**
     * {@inheritDoc}
     * @return
     */
    @Override
    public List<Path> process(DirectoryStream<Path> directoryStream,
ProducerSettings producerSettings) throws Exception {

        List<Path> producedFiles = new ArrayList<Path>();
        // the actual processing of the files
        for (final Path filePath : directoryStream) {
            producedFiles.add(filePath);
            try (BufferedWriter newBufferedWriter = Files.newBufferedWriter
(filePath, Charset.forName("UTF-8"), StandardOpenOption.APPEND)) {
                newBufferedWriter.write("File redirected to archive by sample
step.");
            }
        }
        return producedFiles;
    }

}
```

Configuration option registration

Plug-ins can auto-register global configuration options. This can be achieved by editing the `MANIFEST.MF` file and providing the path to a specific bean, which describes the desired configuration options.

Example MANIFEST.MF file content:

```
Plugin-Configuration-Class:
com.axway.st.server.route.core.component.customstep.CustomStepC
onfigurationBean

Plugin-Label: custom-step
```

Note The `MANIFEST.MF` file must end with a new line. The last line will not be parsed properly if it does not end with a new line or carriage return.

Example CustomStepConfigurationBean:

```
package com.axway.st.server.route.core.component.customstep;

import com.axway.st.plugins.improvedrouting.spi.Description;
import com.axway.st.plugins.improvedrouting.spi.PluginConfigurationBase;

/**
Sample configuration.
*/
public class CustomArchiveStepConfigurationBean implements
PluginConfigurationBase

{

        private String successMessage;

        private String failureMessage;

        public CustomArchiveStepConfigurationBean() {
}

        @Override

        public void initDefault() {

            successMessage = "STEP SUCCEEDED.";

            failureMessage = "STEP FAILED.";

        }

       @Description(value = "Message, that will be logged in Server Log in case of
successful step execution.")

        public String getSuccessMessage() {
```

```
            return successMessage;
        }
       public void setSuccessMessage(String successMessage) {
            this.successMessage = successMessage;
        }
        @Description(value = "Message, that will be logged in Server Log in case of
unsuccessful step execution.")
        public String getFailureMessage() {
            return failureMessage;
        }
        public void setFailureMessage(String failureMessage) {
            this.failureMessage = failureMessage;
        }
}
```

The code and `MANIFEST.MF` file described above will create two configuration options - `successMessage` and `failureMessage`. Both will have default values but will be editable from the **Server Configuration** menu. The configuration option will be named following the model: "`Plugins.AdvancedRouting`" + JAR name + option name.

CustomStepExitStatusException and previous step exit status

All advanced routing steps are capable of evaluating the exit status of the previous step. It can be either "success" or "failure" for regular steps, plus any custom string for the custom steps. Setting a custom status to your custom step is achieved by throwing `CustomStepExitStatusException` with arguments the error message to be logged and the custom status. The exit status of the previous step can be evaluated using the following expression: `${routing.precedingStepExitStatus}`.

Example:

```
if (condition) {
    throw new CustomStepExitStatusException("Custom step failure message",
"CustomFailureStatusExample");
 }
```

Using the SecureTransport SSLContext service

SecureTransport provides a service for creating `javax.net.ssl.SSLContext` and `javax.net.ssl.SSLSocketFactory` objects using its internal keystore. These objects can be used for setting up secure SSL connections to other applications. This service is identical to the *SSL Context service*

exposed for *Custom connectors*, see *SecureTransport exposed services* on page 27.

The service interface is called `com.axway.st.plugins.improvedrouting.environment.SSLContextService` and can be used in the process method of the custom advanced routing step using the following code:

```
SSLContextService sslContextService = producerSettings.getSSLContextService();
```

The service methods are:

- `SSLContext createSSLContext(String certId, boolean verifyPeer) throws SecurityException;`
- `SSLContext createSSLContext(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm) throws SecurityException;`
- `SSLSocketFactory configSSLContextFactory(SSLContext sslContext, String[] protocols, String[] cipherSuites) throws SecurityException;`
- `SSLSocketFactory configSSLContextFactory(String certId, boolean verifyPeer, String sslProtocol, String keyAlgorithm, String trustAlgorithm, String[] protocols, String[] cipherSuites) throws SecurityException;`

These methods have the same behavior as those from the Custom connectors' SSLContextService.

Using the SecureTransport Certificate service

SecureTransport provides a service for certificate parsing and validation against its keystore. The service is called `com.axway.st.plugins.improvedrouting.environment.CertificateService` and can be used in the process method of the custom advanced routing step using the following code:

```
CertificateService certificateService =
producerSettings.getCertificateService();
```

This service is identical to the Certificate service exposed for Custom connectors, see *SecureTransport exposed services* on page 27.

It declares the following methods:

- `CertificateStatus getCertificateStatus(X509Certificate certificate)` – validates the certificate against the SecureTransport keystore; returns one of these enumeration values: `VALID, EXPIRED, NOT_YET_VALID, UNTRUSTED`.
- `KeyPair getKeyPair(String certificateAlias)` – Retrieves public/private key pair from the SecureTransport keystore using certificate alias.
- `X509Certificate getCertificateByAlias(String certificateAlias)` – retrieves certificate from the SecureTransport keystore using certificate alias.

- `X509Certificate getCertificateById(String certificateId)` – retrieves certificate from the SecureTransport keystore using certificate's unique identifier.
- `X509Certificate getIssuer(X509Certificate certificate)` – retrieves certificate issuer from the SecureTransport keystore for given certificate.
- `List<X509Certificate> getCertificateChain(X509Certificate certificate)` – retrieves certificate chain list from the SecureTransport keystore for given certificate. Can be empty, if certificate does not have a chain. The first element is the certificate itself, next element is its issuer and so on to the root certificate.
- `Collection<X509Certificate> getCertificates(String subjectDN)` – retrieves certificate list from the SecureTransport keystore for given subject DN.
- `Subject getCertificateSubject(X509Certificate certificate)` – extracts the domain name of the `certificate`'s subject into a bean of type `Subject`, that contains the following properties: `Common name, Country, Organization, Organization Unit, Locality, State`

Using the SecureTransport Logging service

SecureTransport provides a service for logging messages and exceptions with a different log level. The service interface is called `com.axway.st.plugins.improvedrouting.services.LoggingService` and can be used in the process method of the custom advanced routing step using the following code:

```
LoggingService loggingService = producerSettings.getLoggingService();
```

The *LoggingService* interface declares functionality for logging messages and exceptions with a different log level – ERROR, WARN, INFO, DEBUG, etc.

Examples:

```
mLogger.info(String.format("User '%s' authenticated successfully.", username));
mLogger.warn(String.format("Authentication of user '%s' failed - wrong
password.", username));
mLogger.warn("Problem while parsing client certificate!", e);
mLogger.debug("Start custom authentication process...");
```

The logging level can be controlled by logger in `tm-log4j.xml` and `admin-log4j.xml`.

Using the SecureTransport Expression Evaluator service

SecureTransport provides a service for evaluating and validating expressions used in the custom step implementation.

The service interface is called `com.axway.st.plugins.improvedrouting.services.ExpressionEvaluatorService` and can be used in the process method of the custom advanced routing step using the following code:

```
ExpressionEvaluatorService loggingService =
producerSettings.getExpressionEvaluatorService();
```

This service is identical to the Expression Evaluator service exposed for Custom connectors, see *SecureTransport exposed services* on page 27.

Examples:

```
Map<String, Object> expressionContext = new HashMap<>();

expressionContext.put(AccountEnvService.ENVIRONMENT_KEY,
producerSettings.getAccountEnvService());
expressionContext.put(LdapEnvService.ENVIRONMENT_KEY,
producerSettings.getLdapEnvService());
expressionContext.put(FlowEnvService.ENVIRONMENT_KEY,
producerSettings.getFlowEnvService());
expressionContext.put(HttpEnvService.ENVIRONMENT_KEY,
producerSettings.getHttpEnvService()); expressionContext.put
(SessionEnvService.ENVIRONMENT_KEY, producerSettings.getSessionEnvService());
expressionContext.put(TransferEnvService.ENVIRONMENT_KEY,
producerSettings.getTransferEnvService());
expressionContext.put(StfsEnvService.ENVIRONMENT_KEY,
producerSettings.getStfsEnvService());
expressionContext.put(SsoEnvService.ENVIRONMENT_KEY,
producerSettings.getSsoEnvService());
expressionContext.put(PluginEnvService.ENVIRONMENT_KEY,
producerSettings.getPluginEnvService());
expressionContext.put(RoutingEnvService.ENVIRONMENT_KEY,
producerSettings.getRoutingEnvService());

mExpressionEvaluatorService.loadExpressionService(expressionContext);
```

Note The internal StProducer object is not instantiated for every execution. It reuses the already existing instances which are shared between executions, leading to shared instance members as well. Therefore, to have correct data in producerSettings, the services should not be assigned to variables once and then reused multiple times. Instead, any service within the producerSetting should be evaluated using the getter method each time before using it.

Step deployment

The step should be placed in the `SecureTransport/plugins/routingSteps` directory. In cluster environment the step should be present on all SecureTransport cluster nodes. Restart of the SecureTransport Admin UI and the SecureTransport TransactionManager is required.

Step undeployment

Before removing the step make sure it is not used in any existing SecureTransport routes. After that the step JAR can be deleted. Restart of the SecureTransport Admin UI and the SecureTransport TransactionManager is required.

Accessing third party libraries

The classes of the custom steps are loaded with custom classloader which inherits the TransactionManager classloader. If you want to use third-party libraries, then you must create a JAR together with its dependency JARs into a single executable JAR file.

REST API

Creating the step using the SecureTransport Routes and Steps REST API:

```
curl -i -X POST -A "SecureTransport\HttpClient" -H "Content-Type:
application/json" -H "Accept: application/json" -H "Referer:
https://<SecureTransport Host>:<SecureTransport AdminUi Port>" -u<admin
name>:<admin password>; -k -d "{
    {   "type" : "CustomStep",
        "status" : "ENABLED",
        "fileFilterExpressionType" : "TEXT_FILES",
        "fileFilterExpression": "*",
        "firstCustomProperty" : "value1",
        "secondCustomProperty" : "value2"
    }
}" https://<SecureTransport Host>:<SecureTransport AdminUi
Port>/api/v1.2/routes/<simple route id>/steps

    HTTP/1.1 201 Created
    {
      "id" : "8a6883dc41ff98b60142041b4467006e",
      "link" :
"https://10.232.3.92:444/api/v1.2/routes/8a6883dc41ff98b601420413b61f0068/steps
/8a6883dc41ff98b60142041b4467006e",
      "status" : "success",
      "message" : "A new route step with id = 8a6883dc41ff98b60142041b4467006e
is created."
    }
```

UI page for custom route step

This section of the document explains how to plug the custom developed AdvancedRouting step page into SecureTransportAdmin UI.

After the deployment of the JAR, you can verify the step is plugged in SecureTransport using the REST API:

```
https://<host>:<port>/api/v1.4/routeStepsMetadata
```

Result:

```
[ ...
        {
          "stepType" : "CustomArchiveStep",
          "stepCategory" : "Transformation",
          "stepDisplayName" : "Custom Archive Step",
          "endpointSchema" : "st.customarchivestep",
          "uiPagePath" : "customStep.html",
          "stepPropertyBean" :
"com.axway.st.server.route.core.component.customstep.CustomArchiveStepBean",
          "stepProducer" :
"com.axway.st.server.route.core.component.customstep.CustomArchiveStepProducer"
        }
    ... ]
```

The `uiPagePath` is the page you have developed for the custom Advanced Routing step.

UI component

The user interface logic of the custom Advanced Routing step is added as JavaScript methods to a HTML file in the `/html` folder from the step JAR.

The step is displayed in the SecureTransport Administration Tool and you can verify it by navigating to the **Routes** page.

Example HTML excerpt

```
<script type="text/javascript">
    components.register(new CustomComponent());

        function CustomComponent() {
            components.register(new FileFilterActionsTemplate
("#fileFilterActionsContainer"));
            components.register(new StepFailureActionsTemplate
("#stepFailureActionsContainer"));
        }
```

```
        function fill(currentRouteStep) {
            $('#attr-name').val(currentRouteStep.attrKey);
            $('#attr-value').val(currentRouteStep.attrValue);
        }

        function collect(collectionBean) {
            var attrKey = $.trim($('#attr-name').val());
            collectionBean.attrKey = attrKey;
            var attrValue = $.trim($('#attr-value').val());
            collectionBean.attrValue = attrValue;
        }

        var instance = new ComponentInterface();
        instance.fill = fill;
        instance.collect = collect;

        return Object.seal(instance);
    }
</script>

<div id="step-errors" class="errors"></div>

<form style="width: 800px;" method="post">
    <div class="wrapper-section wide-heading">
        <div id="fileFilterActionsContainer"></div>
        <div style="height: 20px;"></div>

        <div id="stepFailureActionsContainer"></div>
        <div class="clear" style="height: 20px;"></div>

        <fieldset>
            <legend>Add new flow attribute</legend>
            <table>
                <tr>
                    <td class="checkColumn"></td>
                    <td class="labelColumn">
                        <label>Flow attribute Name:</label>
                    </td>
                    <td class="inputColumn">
                        <input type="text" id="attr-name"
value="userVars.zipFileName" style="width: 285px" />
                    </td>
                   <td class="inputColumn">
                        <div class="contextual-help">
                             <a data-content-id="help-box-custom-step-key"
title="Open contextual help" class="modern-iconhelp" tabIndex="-1"
href="#">?</a>
                             <div class="help-dialog hidden">
                             <div class="dialog-header">
                                   <span class="hidden"></span><a class="btn-
close" href="#">x</a>
                             </div>
```

```
                            <div class="dialog-content"></div>
                            </div>
                            <div class="hidden" id="help-box-custom-step-key">
                                 <p>The name of the flow attribute to be
created. Once the step has set its value, this name can later be used as a key
elsewhere in the Advanced Routing steps. </p>
                                 <p>If the value remains and the key is used
as an archive name in the Compress step, the name will be the first 10
characters of the transferred file's content. </p>
                            </div>
                        </div>
                   </td>
               </tr>
               <tr>
                    <td class="checkColumn"></td>
                    <td class="labelColumn">
                        <label>Flow attribute Value:</label>
                    </td>
                    <td class="inputColumn">
                        <input type="text" id="attr-value" value="&#36;
{getFileContent(routing.routeSourceFile, '0', '10', 'UTF-8')}" class="template-
input" style="width: 285px" />
                    </td>
                    <td class="inputColumn">
                        <div class="contextual-help">
                            <a data-content-id="help-box-custom-step-value"
title="Open contextual help" class="modern-iconhelp" tabIndex="-1"
href="#">?</a>
                            <div class="help-dialog hidden">
                            <div class="dialog-header">
                                 <span class="hidden"></span><a class="btn-
close" href="#">x</a>
                            </div>
                            <div class="dialog-content"></div>
                            </div>
                            <div class="hidden" id="help-box-custom-step-
value">
                                 <p>The value of the specified key. Currently
it is the first 10 characters of the content of the file being transferred.</p>
                                 <p>The value can be any hardcoded string or
an expression.</p>
                            </div>
                        </div>
                   </td>
               </tr>
           </table>
       </fieldset>

<div class="clear" style="height: 20px;"></div>

       <!-- Route legend begin. -->
```

```
        <div class="page-legend message clear">
            <p>
                <em class="asterisk">*</em><span>Indicates required
field</span>
            </p>
            <p class="template-input">Enter value or expression</p>
        </div>
        <!-- Route legend end. -->
    </div>
</form>
```

To use a Component in the UI, you must only call `components.register(new CustomComponent())`. This adds the component to a registry that is used to control the flow of operations for all components and the step itself.

For every registered component the `fill()` function is called to populate previous values (in case of Edit step operation).

Note that flow attributes can be used for expression evaluation in Advanced Routing only when the application operates with files.

When the **Save** button is pressed, the following events occur:

- Each component `validate()` function is called, if such is implemented, to ensure that all entered values are valid;
- If validation succeeds, the `collect()` function is called to fill values in `stepBean` used to make the REST call to save new data. The dialog is closed.