Canary Labs
Store & Forward API

Contents

Overview

The purpose of the “Store & Forward” application is to send data from remote locations to a secondary machine that will then push the data into the Canary Enterprise Historian. The application consists of two web services that communicate with each other through a specified port. The names of the two services are the “Sender Service” and the “Receiver Service”.

Sender Service

The “Sender Service” is a .NET WCF application. An application will use this service to send data to the Historian to be stored. Best practice is to place the “Sender Service” on the machine with the logging application. This service will buffer data that is passed into it and forward it to a specified “Receiver Service” when the connection is available. There are fail safes in place to protect the data when connections are lost. The data will not be removed from the buffer until it receives a response from the “Receiver Service” that says it can release the data.

Any web service compatible client can communicate with the service using the http protocol. The default port that the “Sender Service” uses is port 55251. A different port can be specified in the “Sender Service” SAF_SenderService.exe.config file if necessary. In addition, a .NET WCF application running on the same machine as the “Sender Service” can communicate with the service using the net.pipe protocol.

Receiver Service

The “Receiver Service” is used by the “Sender Service” to put data into the Canary Enterprise Historian. A client application cannot use the “Receiver Service” directly.

The “Receiver Service” must be located on a machine that also has the Canary Enterprise Historian software installed on it. This is necessary because the “Receiver Service” pushes data into the Historian through a local interface. When data has been successfully stored in the Historian a response will be sent back to the “Sender Service” to clear the data from the “Sender Service” buffer.

The default port that the “Receiver Service” uses is port 55254 for http and 55256 for net.tcp. A different port can be specified in the “Receiver Service” SAF_ReceiverService.exe.config file if necessary. The “Sender Service” always uses the net.tcp to communicate with the “Receiver Service”.

Helper Class

A .NET helper class has been created for the “Store & Forward” application. The purpose of the helper class is to provide an additional interface for .NET applications that can simplify use of the “Store & Forward” technology. It can help simplify the connection process and provides a different way of interacting with the client that may be more user-friendly.

Time Extension

Time extension is used to extend the time for the last value of a tag if the tag is still logging but a new value has not been stored for an extended amount of time. If a time extension for a tag is set to 15 seconds and the last value has not changed in the last 15 seconds the value will be updated with a current timestamp. If a tag value is then stored with a timestamp that occurred before the extended timestamp but after the timestamp at the start of the last value, the extended value will be modified to reflect the corrected timestamp.

Buffer Files

Default buffer files will be placed on the “Sender Service” machine in the path “ProgramData/Canary Labs/Logger/StoreAndForward/{historian}/{client id}” where the {historian} and {client id} are the historian and client ids that get passed to the “Sender Service” when the session is initialized through

the GetSessionId method request. Buffer files get cleaned up automatically, but the paths to the files will be left intact.

Error Logs

An error log containing all “Sender Service” and “Receiver Service” errors will be created on each machine. The default error log file path will be “ProgramData/Canary Labs/Logger/StoreAndForward/{error log}” where {error log} will be the name of the error log file.

Methods

There are two ways to communicate to the “Sender Service”. Using the base functionality of the “Sender Service” without the .NET helper class, or by using the base functionality of the “Sender Service” along with the added .NET helper class methods. All calls should be made to the “Sender Service” through a client variable after a connection has been made to the “Sender Service”. Calls should never be made to the “Receiver Service” as the “Sender Service” communicates with the “Receiver Service” to transfer data to the historian.

Sender Service Interface

The following methods are available in the “Sender Service”. There are various settings that can be passed to the “Sender Service” that will be explained in detail in this document.

```
string Version()
string[] GetDataSets(out bool failed, string historian)
string GetSessionId(out bool failed, string historian, string clientId, Setting[] settings)
string[] UpdateSettings(out bool failed, string sessionId, Setting[] settings)
object[] GetTagIds(out bool failed, string sessinId, Tag[] tags)
string StoreData(out bool failed, string sessionId, TVQ[] tvqs, Property[] properties, Annotation[] annotations)
string[] NoData(out bool failed, string sessionId, int[] ids)
string CreateNewFile(out bool failed, string sessionId, HistorianFile newFile)
string FileRollOver(out bool failed, string sessionId, HistorianFile rollOverFile)
string GetErrors(out bool failed, out Errors errors, string sessionId)
bool KeepAlive(string sessionId)
string ReleaseSession(out bool failed, string sessionId)
```

Version

Summary

Used to return the current build version of the “Store & Forward” application.

Method

```
string Version()
```

Input Parameters

[No input parameters.]

Response

Current build version of the “Store & Forward” application.

GetDataSets

Summary

Used to return a string array of all data sets in the historian passed to the method.

Method

```
string[] GetDataSets(out bool failed, string historian)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string historian` [Name of a historian machine.]

Response

A string array of data sets in the historian machine passed to the method. If failed flag is set, the response will contain an error message.

GetSessionId

Summary

Used to return a session id that will be required in later method calls. Settings passed in will override default settings in the SAF_SenderService.exe.config files. These settings will be stored for the session until the session is closed and all data has been forwarded successfully to the historian.

Method

```
string GetSessionId(out bool failed, string historian, string clientId, Setting[] settings)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string historian` [Name or IP address of a historian machine where data will be forwarded. The historian machine must have the "Receiver Service" running in order to receive the data. If the "Receiver Service" is running on a port other than the default port of 55256, the port can be set by passing the historian parameter using the following format: "Historian:Port".]

`string clientId` [The clientId must be unique for all sessions connected to the "Sender Service". If the client application loses connection to the "Sender Service" and wishes to resume the existing session after reconnecting, the client must use the same clientId that was previously used. If the client application uses a new clientId, it will be treated as a new session by the "Sender Service". This means the client application will have to send new values for all tags in order for the data time extension to work correctly.]

`Setting[] settings` [Settings that will be specific to the session. Any settings not specified will have their default value. See available settings for input parameters.]

Setting

Class Structure

```
class Setting
{
    string name;
    object value;
}
```

Class Variables

`string name` [Setting name.]

`object value` [Setting value.]

Available Settings

`Pair<"AutoCreateDataSets", bool autoCreateDataSets>` [If a tag is added for a DataSet that does not exist, the DataSet will automatically be created. Default value is false.]

`Pair<"AutoWriteNoData", bool autoWriteNoData>` [Write a value with a "No Data" quality one tick after the last received value or data time extension for each tag in the session when the session is released. Default value is true.]

`Pair<"ClientTimeout", int clientTimeout>` [Number of milliseconds the session will wait for a call. If no calls are received within the specified amount of time, the session will automatically close.]

`Pair<"InsertReplaceData", bool insertReplaceData>` [Allow data to overwrite or insert into existing data. Default value is false.]

`Pair<"TrackErrors", bool trackErrors>` [Store errors that happen in session. Errors can be retrieved by calling the GetErrors method. Default value is false.]

`Pair<"FileSize", int fileSize>` [Buffer file roll-over size in MB. Default value is 32 MB.]

`Pair<"PacketDelay", int packetDelay>` [Millisecond delay between forwarding data. Used to throttle the communication between the "Sender Service" and the "Receiver Service". Default value is 0 milliseconds.]

`Pair<"PacketSize", int packetSize>` [Number of items to forward per request to the "Receiver Service". Used to throttle the communication between the "Sender Service" and the "Receiver Service". Default value is 32,000 items.]

`Pair<"SuppressTimestampErrors", bool suppressTimestampErrors>` [Will not report timestamp errors if they occur.]

`Pair<"ReceiverLogFile", string receiverLogFile>` [Separate log file location if wanting to log errors that occur in the "Receiver Service" on a session by session basis. Default value is null and all errors are logged to the main error log file.]

`Pair<"SenderLogFile", string senderLogFile>` [Separate log file location if wanting to log errors that occur in the "Sender Service" on a session by session basis. Default value is null and all errors are logged to the main error log file.]

Response

A session id that will be required in later method calls.

UpdateSettings

Summary

Used to update the session settings after the session has already been initialized. Any settings not specified in the array will retain their existing value.

Method

```
string[] UpdateSettings(out bool failed, string sessionId, Setting[] settings)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string sessionId` [Session id that was returned from the "GetSessionId" call.]

`Setting[] settings` [Settings that will be specific to the session. See available settings for input parameters.]

Setting

Class Structure

```
class Setting
{
    string name;
    object value;
}
```

Class Variables

`string name` [Setting name.]

`object value` [Setting value.]

Available Settings

`Pair<"AutoCreateDataSets", bool autoCreateDataSets>` [If a tag is added for a DataSet that does not exist, the DataSet will automatically be created. Default value is false.]

`Pair<"AutoWriteNoData", bool autoWriteNoData>` [Write a value with a "No Data" quality one tick after the last received value or data time extension for each tag in the session when the session is released. Default value is true.]

`Pair<"ClientTimeout", int clientTimeout>` [Number of milliseconds the session will wait for a call. If no calls are received within the specified amount of time, the session will automatically close.]

`Pair<"InsertReplaceData", bool insertReplaceData>` [Allow data to overwrite or insert into existing data. Default value is false.]

`Pair<"TrackErrors", bool trackErrors>` [Store errors that happen in session. Errors can be retrieved by calling the GetErrors method. Default value is false.]

`Pair<"FileSize", int fileSize>` [Buffer file roll-over size in MB. Default value is 32 MB.]

`Pair<"PacketDelay", int packetDelay>` [Millisecond delay between forwarding data. Used to throttle the communication between the "Sender Service" and the "Receiver Service". Default value is 0 milliseconds.]

`Pair<"PacketSize", int packetSize>` [Number of items to forward per request to the "Receiver Service". Used to throttle the communication between the "Sender Service" and the "Receiver Service". Default value is 32,000 items.]

`Pair<"SuppressTimestampErrors", bool suppressTimestampErrors>` [Will not report timestamp errors if they occur.]

`Pair<"ReceiverLogFile", string receiverLogFile>` [Separate log file location if wanting to log errors that occur in the "Receiver Service" on a session by session basis. Default value is null and all errors are logged to the main error log file.]

`Pair<"SenderLogFile", string senderLogFile>` [Separate log file location if wanting to log errors that occur in the "Sender Service" on a session by session basis. Default value is null and all errors are logged to the main error log file.]

Response
An array of strings containing error messages that map to each setting passed in. Each value will contain an error message if an error occurred and it will be null if the setting was successfully set. Response could also be an array of length 1 if a general error occurred.

GetTagIds

Summary
Used to get tag ids that will be used in the "StoreData" method when writing tvqs, properties, and annotations.

Method
```
object[] GetTagIds(out bool failed, string sessinId, string[] Tag[] tags)
```

Input Parameters
`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`Tag[] tags` [An array of tags we need to get ids for.]

Tag

Class Structure
```
class Tag
{
    string name;
    string transformEquation;
    bool timeExtention;
}
```
Class Variables

`string name` [An array of tags that the session will be writing data to. These are the full tag names in the format "{DataSet}.{TagName}". Each tag must contain a DataSet. The first period separates the DataSet from the rest of the Tagname.]

`string transformEquation` [An equation used to transform values for a tag as they are being logged. The keyword "Value" will be replaced with each sent tvq value and the equation will be evaluated. As an example, use the equation "Value + 100" to add 100 to each value that is logged for a tag. If no transform is desired, this parameter can be left null or set to an empty string.] `string[] tagNames` [An array of tags that the session will be writing data to. These are the full tag names in the format "{DataSet}.{TagName}". Each tag must contain a DataSet. The first period separates the DataSet from the rest of the Tagname.]

`bool timeExtension` [Time extension is used to extend the time for the last value of a tag if the tag is still logging but a new value has not been stored for an extended amount of time. For more information on how Time Extension works, see the Time Extension section earlier in this document. If set to true, Time Extension will be activated for tag.]

Response
An object array parallel to the tag names array passed to the method containing the logging ids of the tags. The logging id of the tag will be an integer if the tag was successfully mapped. If there was an error with the tag format, a string will be returned for that tag with an error message. Response could also be an array of length 1 if a general error occurred.

StoreData

Summary

Used to buffer data in the "Sender Service" that will be sent to the "Receiver Service" so that it can be stored in the Canary Enterprise Historian. You should not send more than 32,000 data values in one call as this could cause the call to fail due to binding configuration limits.

Method

```
string StoreData(out bool failed, string sessionId, TVQ[] tvqs, Property[]
properties, Annotation[] annotations)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`TVQ[] tvqs` [An array of TVQ data points to store in the historian. See below for more information on the TVQ class structure.]

TVQ

Class Structure

```
class TVQ
{
    int id;
    DateTime timestamp;
    object value;
    ushort quality;
}
```

Class Variables

`int id` [Tag id obtained through the GetTagIds method call.]

`DateTime timestamp` [Data point timestamp.]

`object value` [Data point value.]

`ushort quality` [Data point quality. Quality value of 0xC0 or 192 will be a "Good" quality. Quality value of 0x8000 or 32768 will be a "No Data" value.]

`Property[] properties` [An array of properties to store in the historian. See below for more information on the Property class structure.]

Property

Class Structure

```
class Property
{
    int id;
    string description;
    DateTime timestamp;
    string name;
    object value;
    ushort quality;
}
```

Class Variables

`int id` [Tag id obtained through the GetTagIds method call.]

`string description` [Description of the property. Can be null or an empty string if description is unnecessary.]

`DateTime timestamp` [Property value timestamp.]

`string name` [Name of property that will be set. Some default property names are “Description”, “Eng Units”, “Default High Scale”, “Default Low Scale”, “High Limit”, “Low Limit”, “Sample Interval”. Custom property names can also be used.]

`object value` [Property value.]

`ushort quality` [Property value quality. Quality value of 0xC0 or 192 will be a “Good” quality. Quality value of 0x8000 or 32768 will be a “No Data” value.]

`Annotation[] annotations` [An array of annotations to store in the historian. See below for more information on the Annotation class structure. Annotations can only be written within the bounds of data that has been stored for a tag.]

Annotation

Class Structure

```
class Annotation
{
    int id;
    DateTime timestamp;
    DateTime createdAt;
    string user;
    object value;
}
```

Class Variables

`int id` [Tag id obtained through the GetTagIds method call.]

`DateTime timestamp` [Annotation timestamp.]

`DateTime createdAt` [Annotation timestamp of annotation creation time if different from the timestamp of the annotation.]

`string user` [Name of the user writing the annotation.]

`object value` [Annotation value.]

Response

Response will be null if no errors happened. If an error did occur the response will be a string containing the error.

NoData

Summary

Used to write a “No Data” quality for each tag id sent to method. The “Receiver Service” keeps track of the last tvq that was written for each tag including data time extension and will write a tvq with a “No Data” quality 1 tick after the last timestamp it stored in the Historian. This method is necessary to store a “No Data” value tags after their last time extension because the client application does not know the time of the the last time extension for the tags.

Method

```
string[] NoData(out bool failed, string sessionId, int[] ids)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`string[] ids` [Tag ids of the tags that "No Data" qualities should be written for. If ids is passed to the method as null, "No Data" qualities will be written for all tags in the session.]

Response

An array of strings containing error messages that map to each id passed in. Each value will contain an error message if an error occurred and it will be null if no errors occurred for the id. Response could also be an array of length 1 if a general error occurred.

CreateNewFile

Summary

Used to force the historian to create a new file for the specified DataSet. The new file will be created after all current data in the buffer has been forwarded to the "Receiver Service". If more data is stored after the NewFile method has been called, the data will be written into the new DataSet file. This method is useful for removing old tags from the browse tree of the Historian. The browse tree for a DataSet is created from the tags present in the most recent Historian data file (HDB2). Normally, files are rolled-over at some set interval such as daily and the tags from the previous file are copied to the new file. In order to keep old tags from going into the current file, this method is used to create a new file before calling GetTagIds for the tags that still exist in the system.

Method

```
string CreateNewFile(out bool failed, string sessionId, string dataSet)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`string dataSet` [Name of DataSet that the new file will be created in.]

Response

Response will be null if no errors happened. If an error did occur the response will be a string containing the error.

GetErrors

Summary

Used to retrieve errors that happened for a session in the "Receiver Service" write process and some general status information. TrackErrors setting must be set to true to receive any errors. All errors are logged to an error log file regardless of whether or not the TrackErrors setting has been set.

Method

```
string GetErrors(out bool failed, out Errors errors, string sessionId)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`Errors errors` [A class that contains the errors that occurred in a specific session. If the TrackErrors setting has not been set to true for the session, this parameter will be returned as null. See below for more information on the Error class structure.]

Errors

Class Structure

```
public class Errors
{
    public int storeDataCalls;
    public int forwardDataCalls;
    public List<KeyValuePair<int, string>> errors;
}
```

Class Variables

`public int storeDataCalls` [Number of times the StoreData method has been called. This count will start tracking when the TrackErrors setting has been set to true. Each time GetErrors is called this count will be reset.]

`public int forwardDataCalls` [Number of times the buffered data has been forwarded to the "Receiver Service" method. Since a separate thread is used to forward data to the "Receiver Service", the forward process calls are not parallel to the storage process calls and these counts may not coincide. This count will start tracking when the TrackErrors setting has been set to true. Each time GetErrors is called this count will be reset.]

`List<KeyValuePair<int, string>> errors` [List of KeyValuePairs where the integer is the error id and the string is the error message. An error id of 0 is a general error. If the error id is greater than 0, the error happened in a specific tag where the error id corresponds to the tag id obtained through the GetTagIds method. Each time GetErrors is called this list will be cleared.]

`string sessionId` [Session id that was returned from the GetSessionId call.]

Response

Response will be null if the method succeeded. If the method did not succeed the response will be a string containing the error.

KeepAlive

Summary

Sessions will automatically close if they do not receive any calls within the specified client timeout setting. This method could be called to reset the session timeout to prevent it from closing. Any other call the session receives will also reset the client timeout, but this is an alternate method that can be used when no data presently needs sent to be stored.

Method

```
bool KeepAlive(string sessionId)
```

Input Parameters

`string sessionId` [Session id that was returned from the GetSessionId call.]

Response

Response will be true if session exists and the keep alive method was successful.

ReleaseSession

Summary

Used to close a session. When the session is closed, it will no longer be available for logging. The thread used to forward data to the “Receiver Service” will remain open until all data has been successfully passed to the “Receiver Service”. If the “WriteNoData” setting is true, a “No Data” value will be written one tick after the last recorded timestamp of each tag. The session will continue using the settings it was passed until it fully closes.

Method

```
string ReleaseSession(out bool failed, string sessionId)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`string sessionId` [Session id that was returned from the GetSessionId call.]

Response

Response will be null if no errors happened. If an error did occur the response will be a string containing the error.

Helper Class

The following methods are available in the "Sender Service" using the added functionality of the helper class. All methods are static.

```
void SenderConnect(ConnectionType connectionType, string host, string
usernameCredentials, string passwordCredentials, out SAFSenderServiceContractClient
senderClient)

void SenderConnect(ConnectionType connectionType, string host, int port, string
usernameCredentials, string passwordCredentials, out SAFSenderServiceContractClient
senderClient)

object[] GetTagIds(out bool failed, SAFSenderServiceContractClient senderClient, string
sessionId, string[] tagList)

object[] GetTagIds(out bool failed, SAFSenderServiceContractClient senderClient, string
sessionId, string[] tagList, TimeSpan timeExtension)

object[] GetTagIds(out bool failed, SAFSenderServiceContractClient senderClient, string
sessionId, string[] tagList, TimeSpan[] timeExtensions)

string StoreData(SAFSenderServiceContractClient senderClient, string sessionId, TVQ[]
tvqList, out int tvqsStored)

string StoreData(SAFSenderServiceContractClient senderClient, string sessionId,
Property[] propertyList, out int propertiesStored)

string StoreData(SAFSenderServiceContractClient senderClient, string sessionId,
Annotation[] annotationList, out int annotationsStored)

string StoreData(SAFSenderServiceContractClient senderClient, string sessionId, TVQ[]
tvqList, Property[] propertyList, Annotation[] annotationList, out int tvqsStored, out
int propertiesStored, out int annotationsStored)

bool IsLocalhost(string hostNameOrAddress)

bool TryParseEndpoint(string historianEndpoint, out ConnectionType connectionType, out
string host, out int port)
```

SenderConnect

Summary

Used to create a client connection that can be used to communicate with the "Sender Service". There are two overloads for this method.

Method

```
void SenderConnect(ConnectionType connectionType, string host, string
usernameCredentials, string passwordCredentials, out
SAFSenderServiceContractClient senderClient)

void SenderConnect(ConnectionType connectionType, string host, int port, string
usernameCredentials, string passwordCredentials, out
SAFSenderServiceContractClient senderClient)
```

Input Parameters

`ConnectionType connectionType` [Enumeration parameter specifying the connection type to the "Sender Service". Many of the available connections are not yet available or work in a limited way. There is a plan to add these connection types in the near future.]

ConnectionType

Enumeration Structure

```
public enum ConnectionType
{
    //NetTcp_Windows,
    //NetTcp_Username,
    //NetTcp_Anonymous,
    Https_Username,
    Http_Anonymous,
    NetPipe_WindowsOrUsername,
    NetPipe_Anonymous
}
```

Enumeration Variables

`NetTcp_Windows` [Net.TCP binding connection using windows credentials. Not currently supported.]

`NetTcp_Username` [Net.TCP binding connection using username/password. Not currently supported.]

`NetTcp_Anonymous` [Net.TCP binding connection no credentials. Not currently supported.]

`Https_Username` [Http binding connection using no credentials. Not currently supported.]

`Http_Anonymous` [Http binding connection using no credentials. Limited support.]

`Http_Anonymous` [Http binding connection using no credentials. Limited support.]

`NetPipe_WindowsOrUsername` [Net.Pipe binding using windows credentials or username/password. Limited support. Can only be used to connect on local machine.]

`NetPipe_Anonymous` [Net.Pipe binding using no credentials. Currently recommended connection type. Can only be used to connect on local machine.]

`string host` [Name or IP address of the machine running the "Sender Service".]

`int port` [Port number that the "Sender Service" is running on if other than default.]

`string usernameCredentials` [User name used to verify credentials.]

`string passwordCredentials` [Password used to verify credentials.]

`out SAFSenderServiceContractClient senderClient` [Client used to make calls into the "Sender Service".]

Response

An Exception will be thrown if an error occurs when connecting to the "Sender Service".

GetTagIds

Summary

Used to get tag ids that will be used in the "StoreData" method when writing tvqs, properties, and annotations. There are three overloads for this method.

Method

```
object[] GetTagIds(out bool failed, SAFSenderServiceContractClient senderClient,
string sessionId, string[] tagList)
```

```
object[] GetTagIds(out bool failed, SAFSenderServiceContractClient senderClient,
string sessionId, string[] tagList, TimeSpan timeExtension)

object[] GetTagIds(out bool failed, SAFSenderServiceContractClient senderClient,
string sessionId, string[] tagList, TimeSpan[] timeExtensions)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`SAFSenderServiceContractClient senderClient` [Client used to make calls into the "Sender Service".]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`string[] tagNames` [An array of tags that the session will be writing data to. These are the full tag names in the format "{DataSet}.{TagName}". Each tag must contain a DataSet. The first period separates the DataSet from the rest of the Tagname.]

`TimeSpan timeExtension` [Time extension is used to extend the time for the last value of a tag if the tag is still logging but a new value has not been stored for an extended amount of time. For more information on how Time Extension works, see the Time Extension section earlier in this document. The timeExtension will be applied to all tags passed to the GetTagIds method.]

`TimeSpan[] timeExtensions` [Time extension is used to extend the time for the last value of a tag if the tag is still logging but a new value has not been stored for an extended amount of time. For more information on how Time Extension works, see the Time Extension section earlier in this document. The timeExtensions input parameter should be an array parallel to the tagNames input parameter that specifies a timeExtension value for each tag.]

Response

An object array parallel to the tag names array passed to the method containing the logging ids of the tags. The logging id of the tag will be an integer if the tag was successfully mapped. If there was an error with the tag format, a string will be returned for that tag with an error message. Response could also be an array of length 1 if a general error occurred.

StoreData

Summary

Used to buffer data in the "Sender Service" that will be sent to the "Receiver Service" " so that it can be stored in the Canary Enterprise Historian. There are four overloads for this method. The method will loop through all items sending only 32,000 items to the "Sender Service" until all items have been sent. This will ensure that the call made to the "Sender Service" is not so large that the call fails due to binding configuration limits.

Method

```
string StoreData(SAFSenderServiceContractClient senderClient, string sessionId,
TVQ[] tvqList, out int tvqsStored)

string StoreData(SAFSenderServiceContractClient senderClient, string sessionId,
Property[] propertyList, out int propertiesStored)

string StoreData(SAFSenderServiceContractClient senderClient, string sessionId,
Annotation[] annotationList, out int annotationsStored)
```

```
string StoreData(SAFSenderServiceContractClient senderClient, string sessionId,
TVQ[] tvqList, Property[] propertyList, Annotation[] annotationList, out int
tvqsStored, out int propertiesStored, out int annotationsStored)
```

Input Parameters

`SAFSenderServiceContractClient senderClient` [Client used to make calls into the "Sender Service".]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`TVQ[] tvqList` [An array of TVQ data points to store in the historian. See below for more information on the TVQ class structure.]

TVQ

Class Structure

```
class TVQ
{
    int id;
    DateTime timestamp;
    object value;
    ushort quality;
}
```

Class Variables

`int id` [Tag id obtained through the GetTagIds method call.]

`DateTime timestamp` [Data point timestamp.]

`object value` [Data point value.]

`ushort quality` [Data point quality. Quality value of 0xC0 or 192 will be a "Good" quality. Quality value of 0x8000 or 32768 will be a "No Data" value.]

`Property[] propertiyList` [An array of properties to store in the historian. See below for more information on the Property class structure.]

Property

Class Structure

```
class Property
{
    int id;
    string description;
    DateTime timestamp;
    string name;
    object value;
    ushort quality;
}
```

Class Variables

`int id` [Tag id obtained through the GetTagIds method call.]

`string description` [Description of the property. Can be null or an empty string if description is unnecessary.]

`DateTime timestamp` [Property value timestamp.]

`string name` [Name of property that will be set. Some default property names are "Description", "Eng Units", "Default High Scale", "Default Low Scale", "High Limit", "Low Limit", "Sample Interval". Custom property names can also be used.]

`object value` [Property value.]

`ushort quality` [Property value quality. Quality value of 0xC0 or 192 will be a "Good" quality. Quality value of 0x8000 or 32768 will be a "No Data" value.]

`Annotation[] annotationList` [An array of annotations to store in the historian. See below for more information on the Annotation class structure. Annotations can only be written within the bounds of data that has been stored for a tag.]

Annotation

Class Structure

```
class Annotation
{
    int id;
    DateTime timestamp;
    DateTime createdAt;
    string user;
    object value;
}
```

Class Variables

`int id` [Tag id obtained through the GetTagIds method call.]

`DateTime timestamp` [Annotation timestamp.]

`DateTime createdAt` [Annotation timestamp of annotation creation time if different from the timestamp of the annotation.]

`string user` [Name of the user writing the annotation.]

`object value` [Annotation value.]

`out int tvqsStored` [Number of tvqs that have been successfully stored.]

`out int propertiesStored` [Number of properties that have been successfully stored.]

`out int annotationsStored` [Number of annotations that have been successfully stored.]

Response

Response will be null if no errors happened. If an error did occur the response will be a string containing the error.

NoData

Summary

Used to write a "No Data" quality for each tag id sent to method. There are two overloads for this method. The "Receiver Service" keeps track of the last tvq that was written for each tag including data time extension and will write a tvq with a "No Data" quality 1 tick after the last timestamp it stored in the Historian. This method is necessary to store a "No Data" value tags after their last time extension because the client application does not know the time of the the last time extension for the tags.

Method

```
string NoData(out bool failed, SAFSenderServiceContractClient senderClient, string sessionId)

string[] NoData(out bool failed, SAFSenderServiceContractClient senderClient, string sessionId, int[] tagIds)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`SAFSenderServiceContractClient senderClient` [Client used to make calls into the "Sender Service".]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`string[] tagIds` [Tag ids of the tags that "No Data" qualities should be written for. For the overload without the tagIds parameter or If tagIds is passed to the method as null, "No Data" qualities will be written for all tags in the session.]

Response

For the first overload a string which is null if no error occurred or containing an error message. For the second overload, an array of strings containing error messages that map to each id passed in. Each value will contain an error message if an error occurred and it will be null if no errors occurred for the id. Response could also be an array of length 1 if a general error occurred.

CreateNewFile

Summary

Used to force the historian to create a new file for the specified DataSet. The new file will be created after all current data in the buffer has been forwarded to the "Receiver Service". If more data is stored after the CreateNewFile method has been called, the data will be written into the new DataSet file. This method is useful for removing old tags from the browse tree of the Historian. The browse tree for a DataSet is created from the tags present in the most recent Historian data file (HDB2). Normally, files are rolled-over at some set interval such as daily and the tags from the previous file are copied to the new file. In order to keep old tags from going into the current file, this method is used to create a new file before calling GetTagIds for the tags that still exist in the system.

Method

```
string CreateNewFile(out bool failed, SAFSenderServiceContractClient senderClient,
string sessionId, HistorianFile newFile)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`SAFSenderServiceContractClient senderClient` [Client used to make calls into the "Sender Service".]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`HistorianFile newFile` [A class that contains file information.]

HistorianFile

Class Structure

```
public class HistorianFile
{
    public string dataSet;
    public DateTime fileTime;
}
```

Class Variables

`public string dataSet` [Name of the DataSet that new file will be created in.]

`DateTime fileTime` [File time of the new file that will be created. If the fileTime is assigned a value of DateTime.MinValue, the new file will be created at the timestamp of the last value stored in the current file.]

Response

Response will be null if there were no errors. If an error occurred the response will be a string containing the error.

FileRollOver

Summary

Used to force the historian to roll-over the current DataSet file. A roll-over will write the last value of each tag in the current file into the new file when the roll-over happens.

Method

```
string FileRollOver(out bool failed, SAFSenderServiceContractClient senderClient,
string sessionId, HistorianFile newFile)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`SAFSenderServiceContractClient senderClient` [Client used to make calls into the "Sender Service".]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`HistorianFile newFile` [A class that contains file information.]

HistorianFile

Class Structure

```
public class HistorianFile
{
    public string dataSet;
    public DateTime fileTime;
}
```

Class Variables

`public string dataSet` [Name of the DataSet that new file will be created in.]

`DateTime fileTime` [File time of the new file that will be created. If the fileTime is assigned a value of DateTime.MinValue, the new file will be created at the timestamp of the last value stored in the current file.]

Response

Response will be null if there were no errors. If an error occurred the response will be a string containing the error.

GetErrors

Summary

Used to retrieve errors that happened for a session in the "Receiver Service" write process and some general status information. TrackErrors setting must be set to true to receive any errors. All errors are logged to an error log file regardless of whether or not the TrackErrors setting has been set.

Method

```
string GetErrors(out bool failed, SAFSenderServiceContractClient senderClient,
string sessionId, out Errors errors)
```

Input Parameters

`out bool failed` [Flag that will be set to true if any errors occurred within the method.]

`SAFSenderServiceContractClient senderClient` [Client used to make calls into the "Sender Service".]

`string sessionId` [Session id that was returned from the GetSessionId call.]

`Errors errors` [A class that contains the errors that occurred in a specific session. If the TrackErrors setting has not been set to true for the session, this parameter will be returned as null. See below for more information on the Error class structure.]

Errors

Class Structure

```
public class Errors
{
    public int storeDataCalls;
    public int forwardDataCalls;
    public List<KeyValuePair<int, string>> errors;
}
```

Class Variables

`public int storeDataCalls` [Number of times the StoreData method has been called. This count will start tracking when the TrackErrors setting has been set to true. Each time GetErrors is called this count will be reset.]

`public int forwardDataCalls` [Number of times the buffered data has been forwarded to the "Receiver Service" method. Since a separate thread is used to forward data to the "Receiver Service", the forward process calls are not parallel to the storage process calls and these counts may not coincide. This count will start tracking when the TrackErrors setting has been set to true. Each time GetErrors is called this count will be reset.]

`List<KeyValuePair<int, string>> errors` [List of KeyValuePairs where the integer is the error id and the string is the error message. An error id of 0 is a general error. If the error id is greater than 0, the error happened in a specific tag where the error id corresponds to the tag id obtained through the GetTagIds method. Each time GetErrors is called this list will be cleared.]

Response

Response will be null if the method succeeded. If the method did not succeed the response will be a string containing the error.

IsLocalhost

Summary

Used to check if a hostname or IP address is the local machine.

Method

```
bool IsLocalhost(string hostNameOrAddress)
```

Input Parameters

`string hostNameOrAddress` [Hostname or IP address of a machine to test if it is the local machine.]

Response

Result will be true if the hostname or IP address is the local machine and false if it is not.

TryParseEndpoint

Summary

Used to parse an endpoint for individual pieces.

Method

```
bool TryParseEndpoint(string historianEndpoint, out ConnectionType connectionType,
out string host, out int port)
```

Input Parameters

`string historianEndpoint` [Endpoint used to connect to the "Sender Service".]

`ConnectionType connectionType` [Enumeration parameter specifying the connection type to the "Sender Service". Many of the available connections are not yet available or work in a limited way. There is a plan to add these connection types in the near future. Will return as Http_Anonymous as default value if method was unable to parse the endpoint.]

ConnectionType

Enumeration Structure

```
public enum ConnectionType
{
    //NetTcp_Windows,
    //NetTcp_Username,
    //NetTcp_Anonymous,
    Https_Username,
    Http_Anonymous,
    NetPipe_WindowsOrUsername,
    NetPipe_Anonymous
}
```

Enumeration Variables

`NetTcp_Windows` [Net.TCP binding connection using windows credentials. Not currently supported.]

`NetTcp_Username` [Net.TCP binding connection using username/password. Not currently supported.]

`NetTcp_Anonymous` [Net.TCP binding connection no credentials. Not currently supported.]

`Https_Username` [Http binding connection using no credentials. Not currently supported.]

`Http_Anonymous` [Http binding connection using no credentials. Limited support.]

`Http_Anonymous` [Http binding connection using no credentials. Limited support.]

`NetPipe_WindowsOrUsername` [Net.Pipe binding using windows credentials or username/password. Limited support. Can only be used to connect on local machine.]

`NetPipe_Anonymous` [Net.Pipe binding using no credentials. Currently recommended connection type. Can only be used to connect on local machine.]

`string host` [Name or IP address of the machine specified in the endpoint. Will be returned as an empty string if the method was unable to parse the endpoint.]

`int port` [Port number specified in the endpoint. Will be returned as 0 if the method was unable to parse the endpoint.]

Response

Result will be true if the endpoint was successfully parsed and false if it was not.

Example Code

The following code is written in C#. Each example is dependent on the container code which contains local variables and client connection methods that hook up to the “Sender Service”. Example code has not been optimized and is given merely as a demonstration of how the methods described in this documentation can used.

Sender Service Interface

The sender service interface example code contains example code for all methods present in the “Sender Service” excluding the helper class methods. Example methods for the helper class methods can be found under the “Helper Class” section of the example code documentation.

Service Reference

A service reference to the “Sender Service” will be necessary for communication with the “Sender Service”. The default service reference address is http://localhost:55251/saf/sender/mex.

Container

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using SAF_Helper.SAF_SenderService;

namespace SAF_ExampleCode
{
    class ExampleCode
    {
        private string host = "localhost"; // necessary when using named pipe binding
(other bindings could be configured if necessary)
        private SAFSenderServiceContractClient client = null;
        private string sessionId = null;
        private Dictionary<string, int> tagIds = new Dictionary<string, int>();

        static private NetNamedPipeBinding CreateNamedPipeBindingBase()
        {
            NetNamedPipeBinding pipeBinding = new NetNamedPipeBinding();
            pipeBinding.MaxBufferPoolSize = 2147483647;
            pipeBinding.MaxReceivedMessageSize = 2147483647;
            pipeBinding.MaxConnections = 50;
            pipeBinding.ReaderQuotas.MaxStringContentLength = 2147483647;
            pipeBinding.ReaderQuotas.MaxArrayLength = 2147483647;
            pipeBinding.ReaderQuotas.MaxBytesPerRead = 2147483647;
            pipeBinding.ReceiveTimeout = TimeSpan.FromHours(1);

            return pipeBinding;
        }

        private void ConnectClient()
        {
            if (client == null)
            {
                NetNamedPipeBinding pipeBinding = CreateNamedPipeBindingBase();
                pipeBinding.Security.Mode = NetNamedPipeSecurityMode.None;
                System.ServiceModel.Channels.Binding binding = pipeBinding;
```

```
                // host and port are ignored because this is always local
                string address = "net.pipe://localhost/saf/sender/anonymous";
                EndpointAddress endpoint = new EndpointAddress(new Uri(address));
                client = new SAFSenderServiceContractClient(binding, endpoint);
            }
        }

        public Setting PopulateSetting(string name, object value)
        {
            Setting setting = new Setting();
            setting.name = name;
            setting.value = value;
            return setting;
        }

        // methods...
    }
}
```

InterfaceVersion

```
public string InterfaceVersion()
{
     ConnectClient();
     return client.InterfaceVersion();
}
```

GetDataSets

```
public string[] GetDataSets()
{
    ConnectClient();

    bool failed;
    string historian = host;

    string[] results = client.GetDataSets(out failed, historian);
    if (failed)
    {
        // handle error
        string error = results[0];
    }

    return results;
}
```

GetSessionId

```
public string GetSessionId()
{
    ConnectClient();

    if (sessionId == null)
    {
        bool failed;
        string historian = host;
        string clientId = "ExampleCode";
```

```
        // arbitrary settings used for example code
        List<Setting> settings = new List<Setting>();
        settings.Add(new Setting("AutoCreateDataSets", true));
        settings.Add(new Setting("PacketDelay", 500));
        settings.Add(new Setting("TrackErrors", true));

        string result = client.GetSessionId(out failed, historian, clientId,
settings.ToArray());
        if (failed)
        {
            // handle error
            string error = result;
        }
        else
            sessionId = result;
    }

    return sessionId;
}
```

UpdateSettings

```
public string[] UpdateSettings()
{
    ConnectClient();

    bool failed;
    string sessionId = GetSessionId();

    // arbitrary settings used for example code
    List<Setting> settings = new List<Setting>();
    settings.Add(new Setting("PacketDelay", 0));
    settings.Add(new Setting("TrackErrors", false));


    string[] results = client.UpdateSettings(out failed, sessionId,
settings.ToArray());
    if (failed)
    {
        foreach (string result in results)
        {
            if (result != null)
            {
                // handle error
                string error = result;
            }
        }
    }

    return results;
}
```

GetTagIds

```
public Dictionary<string, int> GetTagIds()
{
    ConnectClient();

    if (tagIds.Count != 4)
```

```
    {
        bool failed;
        string sessionId = GetSessionId();

        // arbitrary tag names used for example code
        string[] tagNames = new string[] {
            "ExampleCode.Tag 0001",
            "ExampleCode.Tag 0002",
            "ExampleCode.Tag 0003",
            "ExampleCode.Tag 0004"
        };

        // arbitrary time extension settings used for example code
        // to apply same time extension to each tag use an array length of 1
        // to prevent time extension use an array length of 0
        TimeSpan[] timeExtensions = new TimeSpan[] {
            TimeSpan.FromSeconds(30),
            TimeSpan.FromSeconds(30),
            TimeSpan.FromSeconds(15),
            TimeSpan.FromSeconds(15)
        };

        object[] results = client.GetTagIds(out failed, sessionId, tagNames,
timeExtensions);
        if (failed)
        {
            for (int i = 0; i < tagNames.Length; i++)
            {
                object result = results[i];
                if (!(result is int))
                {
                    // handle error
                    string error = (string)result;
                }
            }
            return tagIds;
        }
        else
        {
            // create tag mapping to reference id
            for (int i = 0; i < tagNames.Length; i++)
            {
                string tagName = tagNames[i];
                int id = (int)results[i];
                tagIds.Add(tagName, id);
            }
        }
    }

    return tagIds;
}
```

StoreData

```
public string StoreData()
{
    ConnectClient();

    List<TVQ> tvqsList = new List<TVQ>();
```

```
    List<Property> propertiesList = new List<Property>();
    List<Annotation> annotationsList = new List<Annotation>();

    // create data to store
    DateTime now = DateTime.Now;
    Dictionary<string, int> tagIds = GetTagIds();
    foreach (KeyValuePair<string, int> pair in tagIds)
    {
        string tagName = pair.Key;
        int id = pair.Value;

        // add tvq data
        for (int i = 0; i < 500; i++)
        {
            TVQ tvq = new TVQ();
            tvq.id = id;
            tvq.timestamp = now.AddTicks(i);
            tvq.value = i % 100;
            tvq.quality = 0xC0;
            tvqsList.Add(tvq);
        }

        // add property data
        Property highScale = new Property();
        highScale.id = id;
        highScale.description = null;
        highScale.timestamp = now;
        highScale.name = "Default High Scale";
        highScale.value = 100;
        highScale.quality = 0xC0;
        propertiesList.Add(highScale);

        // add property data
        Property lowScale = new Property();
        lowScale.id = id;
        lowScale.description = null;
        lowScale.timestamp = now;
        lowScale.name = "Default Low Scale";
        lowScale.value = 0;
        lowScale.quality = 0xC0;
        propertiesList.Add(lowScale);

        // add annotation
        Annotation annotation = new Annotation();
        annotation.id = id;
        annotation.timestamp = now;
        //annotation.createdAt = now; // only necessary if creation time differs from
annotation timestamp
        annotation.user = "Example User";
        annotation.value = "Example Annotation";
        annotationsList.Add(annotation);
    }

    bool failed;
    string sessionId = GetSessionId();
    TVQ[] tvqs = tvqsList.ToArray();
    Property[] properties = propertiesList.ToArray();
    Annotation[] annotations = annotationsList.ToArray();
```

```
    string result = client.StoreData(out failed, sessionId, tvqs, properties,
annotations);
    if (failed)
    {
        // handle error
        string error = result;
    }

    return result;
}
```

NoData

```
public string[] NoData()
{
    ConnectClient();

    bool failed;
    string sessionId = GetSessionId();
    Dictionary<string, int> tagIds = GetTagIds();

    return client.NoData(out failed, sessionId, tagIds.Values.ToArray());
}
```

CreateNewFile

```
public string CreateNewFile()
{
    ConnectClient();

    bool failed;
    string sessionId = GetSessionId();
    HistorianFile newFile = new HistorianFile();
    newFile.dataSet = "ExampleCode";
    newFile.fileTime = DateTime.MinValue;

    string result = client.CreateNewFile(out failed, sessionId, newFile);
    if (failed)
    {
        // handle error
        string error = result;
    }

    return result;
}
```

FileRollOver

```
public string FileRollOver()
{
    ConnectClient();

    bool failed;
    string sessionId = GetSessionId();
    HistorianFile rollOverFile = new HistorianFile();
    rollOverFile.dataSet = "ExampleCode";
    rollOverFile.fileTime = DateTime.MinValue;

    string result = client.FileRollOver(out failed, sessionId, rollOverFile);
```

```
    if (failed)
    {
        // handle error
        string error = result;
    }

    return result;
}
```

GetErrors

```
public string GetErrors()
{
    ConnectClient();

    bool failed;
    Errors errors;
    string sessionId = GetSessionId();

    string result = client.GetErrors(out failed, out errors, sessionId);
    if(failed)
    {
        // handle error
        string error = result;
    }

    return result;
}
```

ReleaseSession

```
public string ReleaseSession()
{
    ConnectClient();

    bool failed;
    string sessionId = GetSessionId();

    string result = client.ReleaseSession(out failed, sessionId);
    if (failed)
    {
        // handle error
        string error = result;
    }

    // reset stored variables
    this.sessionId = null;
    this.tagIds.Clear();

    return result;
}
```

Helper Class

The helper class example code contains example code for all methods present in the helper class. Some of the methods are dependent on methods that have been previously described in the sender service interface. The helper class offers an alternate way of making calls to the “Sender Service” along with a few additional methods that may be useful.

Container

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using SAF_Helper;
using SAF_Helper.SAF_SenderService;

namespace SAF_DataGeneration
{
    class HelperCode
    {
        private string host = "localhost"; // necessary when using named pipe binding
(other bindings could be configured if necessary)
        private SAFSenderServiceContractClient client = null;
        private string sessionId = null;
        private Dictionary<string, int> tagIds = new Dictionary<string, int>();

        public string GetSessionId()
        {
            SenderConnect();

            if (sessionId == null)
            {
                bool failed;
                string historian = host;
                string clientId = "ExampleCode";

                // arbitrary settings used for example code
                List<Setting> settings = new List<Setting>();
                settings.Add(new Setting("AutoCreateDataSets", true));
                settings.Add(new Setting("PacketDelay", 500));
                settings.Add(new Setting("TrackErrors", true));

                string result = client.GetSessionId(out failed, historian, clientId,
settings.ToArray());
                if (failed)
                {
                    // handle error
                    string error = result;
                }
                else
                    sessionId = result;
            }

            return sessionId;
        }

        // methods...
    }
```

```
}
```

SenderConnect

```
public void SenderConnect()
{
    if (client == null)
    {
        ConnectionType connectionType = ConnectionType.NetPipe_Anonymous;
        string host = null;
        string usernameCredentials = null;
        string passwordCredentials = null;
        SAF_HelperClass.SenderConnect(connectionType, host, usernameCredentials,
passwordCredentials, out client);
    }
}
```

GetTagIds

```
public Dictionary<string, int> GetTagIds()
{
    SenderConnect();

    if (tagIds.Count != 4)
    {
        bool failed;
        string sessionId = GetSessionId();

        // arbitrary tag names used for example code
        string[] tagNames = new string[] {
            "ExampleCode.Tag 0001",
            "ExampleCode.Tag 0002",
            "ExampleCode.Tag 0003",
            "ExampleCode.Tag 0004"
        };

        // arbitrary time extension value
        TimeSpan timeExtension = TimeSpan.FromSeconds(30);

        // arbitrary time extension settings used for example code
        // to apply same time extension to each tag use an array length of 1
        // to prevent time extension use an array length of 0
        TimeSpan[] timeExtensions = new TimeSpan[] {
            TimeSpan.FromSeconds(30),
            TimeSpan.FromSeconds(30),
            TimeSpan.FromSeconds(15),
            TimeSpan.FromSeconds(15)
        };

        // no time extension
        object[] results = SAF_HelperClass.GetTagIds(out failed, client, sessionId,
tagNames);

        // 30 second time extension
        //object[] results = SAF_HelperClass.GetTagIds(out failed, client, sessionId,
tagNames, timeExtension);

        // mapped individually
```

```
        //object[] results = SAF_HelperClass.GetTagIds(out failed, client, sessionId,
tagNames, timeExtensions);

        if (failed)
        {
            for (int i = 0; i < tagNames.Length; i++)
            {
                object result = results[i];
                if (!(result is int))
                {
                    // handle error
                    string error = (string)result;
                }
            }
            return tagIds;
        }
        else
        {
            // create tag mapping to reference id
            for (int i = 0; i < tagNames.Length; i++)
            {
                string tagName = tagNames[i];
                int id = (int)results[i];
                tagIds.Add(tagName, id);
            }
        }
    }

    return tagIds;
}
```

StoreData

```
public string StoreData()
{
    SenderConnect();

    List<TVQ> tvqsList = new List<TVQ>();
    List<Property> propertiesList = new List<Property>();
    List<Annotation> annotationsList = new List<Annotation>();

    // create data to store
    DateTime now = DateTime.Now;
    Dictionary<string, int> tagIds = GetTagIds();
    foreach (KeyValuePair<string, int> pair in tagIds)
    {
        string tagName = pair.Key;
        int id = pair.Value;

        // add tvq data
        for (int i = 0; i < 500; i++)
        {
            TVQ tvq = new TVQ();
            tvq.id = id;
            tvq.timestamp = now.AddTicks(i);
            tvq.value = i % 100;
            tvq.quality = 0xC0;
            tvqsList.Add(tvq);
        }
```

```
        // add property data
        Property highScale = new Property();
        highScale.id = id;
        highScale.description = null;
        highScale.timestamp = now;
        highScale.name = "Default High Scale";
        highScale.value = 100;
        highScale.quality = 0xC0;
        propertiesList.Add(highScale);

        // add property data
        Property lowScale = new Property();
        lowScale.id = id;
        lowScale.description = null;
        lowScale.timestamp = now;
        lowScale.name = "Default Low Scale";
        lowScale.value = 0;
        lowScale.quality = 0xC0;
        propertiesList.Add(lowScale);

        // add annotation
        Annotation annotation = new Annotation();
        annotation.id = id;
        annotation.timestamp = now;
        //annotation.createdAt = now; // only necessary if creation time differs from
annotation timestamp
        annotation.user = "Example User";
        annotation.value = "Example Annotation";
        annotationsList.Add(annotation);
    }

    int tvqsStored;
    int propertiesStored;
    int annotationsStored;
    string sessionId = GetSessionId();
    TVQ[] tvqs = tvqsList.ToArray();
    Property[] properties = propertiesList.ToArray();
    Annotation[] annotations = annotationsList.ToArray();

    // send only tvqs in this call
    //string result = SAF_HelperClass.StoreData(client, sessionId, tvqs, out
tvqsStored);

    // send only properties in this call
    //string result = SAF_HelperClass.StoreData(client, sessionId, properties, out
propertiesStored);

    // send only annotations in this call
    //string result = SAF_HelperClass.StoreData(client, sessionId, annotations, out
annotationsStored);

    // send tvqs, properties, and annotations in this call
    string result = SAF_HelperClass.StoreData(client, sessionId, tvqs, properties,
annotations, out tvqsStored, out propertiesStored, out annotationsStored);

    return result;
}
```

IsLocalhost

```
public bool IsLocalhost()
{
    bool result = SAF_HelperClass.IsLocalhost(host);
    return result;
}
```

TryParseEndpoint

```
public bool TryParseEndpoint()
{
    ConnectionType connectionType;
    string host;
    int port;

    string endpoint = "net.pipe://localhost/saf/sender/anonymous";
    bool result = SAF_HelperClass.TryParseEndpoint(endpoint, out connectionType, out
host, out port);

    return result;
}
```