



# Smart Integration Connector Guide

---

8.1.3 Release

Copyright © 2023 OneStream Software LLC. All rights reserved.

Any warranty with respect to the software or its functionality will be expressly given in the Subscription License Agreement or Software License and Services Agreement between OneStream and the warrantee. This document does not itself constitute a representation or warranty with respect to the software or any related matter.

OneStream Software, OneStream, Extensible Dimensionality and the OneStream logo are trademarks of OneStream Software LLC in the United States and other countries. Microsoft, Microsoft Azure, Microsoft Office, Windows, Windows Server, Excel, .NET Framework, Internet Information Services, Windows Communication Foundation and SQL Server are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. DevExpress is a registered trademark of Developer Express, Inc. Cisco is a registered trademark of Cisco Systems, Inc. Intel is a trademark of Intel Corporation. AMD64 is a trademark of Advanced Micro Devices, Inc. Other names may be trademarks of their respective owners.

# Table of Contents

About This Guide .....	1
Benefits .....	2
Common Understanding .....	2
OneStream Client Application Terms .....	2
OneStream Local Gateway Configuration Terms .....	4
Architecture .....	5
Additional Considerations .....	7
Requirements .....	8
OneStream Smart Integration Connector Environment Setup .....	8
Advanced Networking and Whitelisting .....	9
Upgrade Smart Integration Connector for Private Preview or Limited Availability Customers .....	10
Upgrade from 7.2, 7.3, 7.4, or 8.0 to 8.1 .....	10
Upgrade from 7.1 to 8.1 .....	12
Migration from VPN Considerations .....	14
Setup and Installation .....	15
Smart Integration Connector Setup .....	15
Gateway Terms .....	15
Local Gateway Server Installation .....	18

Create a New Gateway .....	19
Create a Database Connection .....	20
Create a Direct Connection .....	21
Export and Import the Gateway Configuration .....	23
New Gateway Key Generation .....	26
Create a Local Gateway Connection to a Data Source .....	26
Microsoft SQL Server .....	28
MySQL Data Provider .....	29
Oracle Database Examples .....	30
PostgreSQL (Npgsql Data Provider) .....	32
OleDb Data Provider .....	32
ODBC Data Provider .....	33
(Optional) Remove UserID and Passwords by Integrated Security .....	34
Test the Gateway .....	37
Restart OneStream Smart Integration Connector Gateway .....	39
Redundant and Fail-over Gateways .....	40
Define Custom Database Connections in OneStream System Configuration Setup .....	41
Smart Integration Additional Settings .....	43
Local Application Data Settings .....	43

## Table of Contents

---

Referenced Assemblies Folder .....	43
Log Settings .....	45
Advanced Networking and Whitelisting .....	48
Whitelist the Azure Relay to your Firewall .....	48
Use Smart Integration Connector .....	49
Examples .....	49
Data Adapters Example .....	49
SQL Table Editor Example .....	49
Grid View Example .....	50
Perform a Drill Back .....	51
Perform a Write Back .....	53
Support for sFTP .....	56
Transferring Files from Local FileShare .....	59
Step 1 - Setup the Remote Server / Remote Share .....	59
Step 2 - Pull file from Extender Business Rule .....	61
Step 3 - Automate from Data Management / Task Scheduler ....	64
Support for DLL Migration .....	64
Support for ERPConnect (SAP) .....	65
Business Rules .....	68
ExecRemoteGatewayRequest .....	69

ExecRemoteGatewayCachedBusinessRule .....	71
ExecRemoteGatewayJob .....	73
ExecRemoteGatewayBusinessRule .....	77
GetRemoteDataSourceConnection .....	81
GetRemoteGatewayJobStatus .....	83
GetSmartIntegrationConfigValue .....	85
GetGatewayConnectionInfo .....	87
Incompatible Business Rules .....	90
Obtain Data through a WebAPI .....	90
Troubleshooting .....	92
Error Log .....	92
Common Errors .....	92
Memory Issues .....	92
Gateway Version is empty .....	93
Custom Data Source Names .....	93
Array cannot be null Error .....	93
Opening and Saving Configuration Errors .....	94
Incorrect or Missing Library References .....	95
Script Error During Upgrade .....	95
Smart Integration Functions with a DataTable/DataSet Parameter	96

## Table of Contents

---

Data Returned as a String .....	99
Null Columns are Not Returned .....	99
Running Encrypted Smart Integration Connector Functions (Remote Business Rules) .....	99
Manual Start and Stop .....	99
Communication Error .....	100
Trusted Certificate Chain .....	101
Gateway Unable to Connect .....	101

# About This Guide

This guide is intended for OneStream administrators and IT professionals. It describes how to manage Smart Integration Connector to connect local data sources to your OneStream Cloud instance. OneStream Cloud Operations and Support can assist with the tasks needed to set up Smart Integration Connector:

- Installing or upgrading to OneStream platform version 8.1.
- Assisting with the installation of Smart Integration Connector Local Gateway Server in your environment.



# Benefits

OneStream applications are strategic components in your financial environment. Data from financial systems is imported to OneStream and contributes to financial closing and reporting processes. While performing analysis, you leverage data lineage capabilities to make contextual associations to data sources in your network.

You will need to set up and configure data sources that may be accessed by OneStream processes. Traditionally, data connectivity between a OneStream Cloud instance and local data sources is established using a Virtual Private Network (VPN) and all data source credentials and supporting files are located on OneStream application servers.

The goals for Smart Integration Connector are to establish all required data source connections without VPN and establish residency and management of data source connections solely in your network.

With Smart Integration Connector, you can:

- Securely establish connectivity between OneStream Cloud and data sources in your network without a VPN connection.
- Create and manage network data source integration using OneStream administration interfaces.
- Locally manage database credentials and ancillary files.

## Common Understanding

Use the reference charts below to understand common terms used throughout the product and this document.

## OneStream Client Application Terms

Term	Definition
OneStream Windows Application client	The Windows client facilitating user interface access for all user personas to OneStream applications.

## Benefits

---

OneStream Windows Application Server (App Server)	The application server executing all OneStream business logic and processing.
Gateway	Gateways define direct channels of integration between the OneStream Cloud and a local customer network. Gateways are represented by a unique gateway key and are configured for communication to an Azure Relay endpoint. Gateways carry a 1:1 correlation to a local gateway. The channel of communication established from the OneStream gateway and a local gateway created in Smart Integration Connector.
Gateway Server	A gateway server carries no unique technical definition or configuration address. It is a node in the tree control UI to organize gateways and typically corresponds to an installed local gateway server name.
Custom Database Connections	Custom database connections define a named data source to which OneStream may connect using Smart Integration Connector for the purpose of data import, data export, or drill through querying. The named custom database connection is referenced in OneStream business logic (data management objects or business rules) to initiate data source connectivity. Credentials and ancillary files required for a designated data source connection are configured to and reside on the corresponding local gateway server.
Direct Connection (for example, SFTP, WebAPI)	A direct connection represents a point-to-point channel to designated resources such as an sFTP server or Web API (including iPaaS services). The OneStream Local Gateway Server Configuration Utility UI facilitates configuration of mapped connections to resources where the on-premises TCP port is mapped to a server (hostname/IP).

Database Connection	A database connection represents the ultimate datasource destination for Smart Integration Connector. A local gateway connection may be a designated database. The OneStream Local Gateway Server Configuration Utility facilitates configuration of required credentials and supporting files. The identification of a local gateway connection must correspond to a custom database connection established to the OneStream Application Server.
Whitelist (Whitelisting)	Whitelisting provides the capability of providing a list of IP addresses or namespaces that data can flow through the Smart Integration Connector. Whitelisting can be applied to the Relay in the OneStream Windows Application client and also applied to your firewall through your IT Admin.

## OneStream Local Gateway Configuration Terms

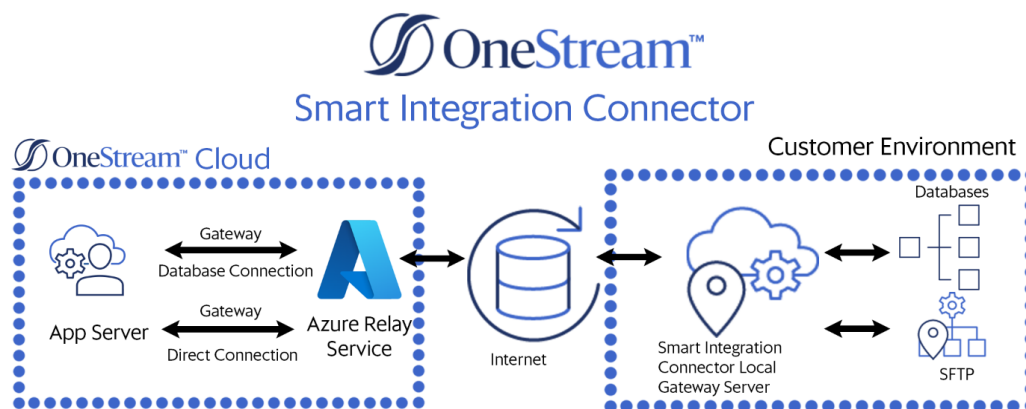
Term	Definition
Local Gateway Server	Smart Integration Connector requires a client installation on Windows servers to establish a local gateway server. The local gateway server houses one or more local gateways which are configured through the OneStream Local Gateway Configuration. A local gateway server corresponds 1:1 with a defined gateway server.
Local Gateway	Local gateways define the local customer endpoint for distinct channels of communication used by Smart Integration Connector. A local gateway facilitates connections to local databases, Web API connections, iPaaS servers, or sFTP servers and corresponds 1:1 with a gateway definition on the OneStream Application Server. To ensure a valid connection, a local

## Benefits

	gateway must be configured by importing the corresponding gateway definition exported from the OneStream Windows Application client.
Local Gateway Connections	Local gateway connections are the database connections defined in the utility and confirm the connection between the local gateway and the local data sources.
OneStream Local Gateway Configuration	This utility is where you configure the Local Gateway Server, Local Gateways and Local Gateway Connections to data sources.

## Architecture

In contrast to a direct data source connection established using a VPN, Smart Integration Connector makes an indirect connection to data sources. Smart Integration local gateways integrate with on-premises customer environments through a cloud hosted service called Azure Relay. The locally installed and configured local gateway server makes the direct connection to data sources and responds to the OneStream application.



**NOTE:** In OneStream, Custom Database Server Connections define the connection through the gateway to the data source.

The two primary services of Smart Integration Connector are:

- **OneStream Application Server:** The application server brokers communication between the OneStream Cloud instance application and the Azure Relay service.
- **Local Gateway Server:** Instances of the Smart Integration Connector Local Gateway Server are installed inside your network and configured to make direct connections to designated data sources. The Smart Integration Connector Local Gateway Server runs as a Windows service and brokers communication between local data sources and Azure Relay using an outbound connection over port 443.

The components of the Smart Integration Connector are:

- OneStream Windows Application client

Direct and Database connections (Gateways) configured through **System > Administration > Smart Integration Connector**.

**NOTE:** The SmartIntegrationConnectorAdminPage role must be assigned to a user for this to be visible.

- A Custom Database Connection to the local gateway data source. Custom Database Connections are configured in **System > System Configuration > Application Server Configuration > Database Server Connections**.

**NOTE:** The ManageSystemConfiguration role must be assigned to a user for this to be visible.

- OneStream Smart Integration Connector Local Gateway Server
  - Local Gateway Settings provide the connection information to establish the gateway connection to the OneStream Windows Application. Gateway settings are exported from the gateway settings in the OneStream Windows Application and imported to the Local Gateway section of the **OneStream Local Gateway Configuration**.
  - Local Gateway Connections provide the setup information necessary for the Smart Integration Connector Local Gateway to connect to local data sources. Local Gateway Connections are set up through the **OneStream Local Gateway Configuration** in the Gateway Connections Settings section.

## Additional Considerations

- To provide high availability, there can be multiple instances of a designated local gateway server, each running on a separate server bound to the same gateway where the services run in an active / passive (fail over) manner.
- Multiple local gateways can be installed to establish global connectivity to data sources in different subnetworks.
- Local gateway configuration must align to the corresponding gateway as defined in the OneStream Windows application. An export process from the OneStream Windows application gateway user interface can assist with the alignment to ensure corresponding names and keys are identical.

# Requirements

## OneStream Smart Integration Connector Environment Setup

Smart Integration Connector is Generally Available to all SaaS customers starting with OneStream version 8.0.

- Install or upgrade OneStream to the latest version. See [Setup and Installation](#).
- Work with your IT team to install the latest version of the Smart Integration Connector Local Gateway Server in your environment.
  - Windows Server 2019+
  - .NET Framework 4.8
  - Minimum of 8 GB of RAM

**NOTE:** Memory and processor requirements are driven by the frequency and volume of remote data accessed through the gateway service or if remote business rules / long running jobs are leveraged. Typical data-access patterns with 1 million or less records being queried at a time can be accomplished with 8-16 GB of RAM and two newer generation processors. For queries returning over 1 million records, 32 GB or more RAM is recommended.

- The installer requires administrative permission on the server to perform the installation.
- See [Smart Integration Connector Local Gateway Server Installation](#).
- Create a valid database connection string and internally test the connection from the Windows server to the database. See [Create a Database Connection](#) for more information.
- Be a OneStream administrator to configure corresponding data sources in the OneStream environment.

# Advanced Networking and Whitelisting

If your organization needs to filter and/or whitelist network traffic for the Smart Integration Connector, you will need to work with your IT team to restrict this traffic. See [Advanced Networking / Whitelisting](#) for more information. For any additional questions, please reach out to Customer Support.



# Upgrade Smart Integration Connector for Private Preview or Limited Availability Customers

The following section describes how to upgrade Smart Integration Connector.

## Upgrade from 7.2, 7.3, 7.4, or 8.0 to 8.1

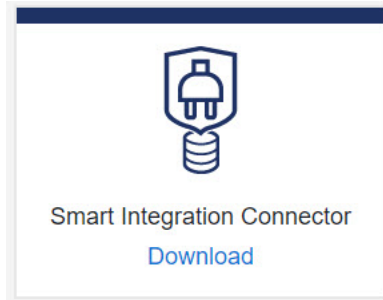
This is only required if you are upgrading from 7.2, 7.3, 7.4, or 8.0 to version 8.1. As part of the upgrade, you can expect the following:

- A copy of the original configuration file from the prior version will be saved.
- Existing gateways should continue to function as they did prior to the install.
- If the Smart Integration Connector Windows Service is running, then the service will automatically be started after install.
- Previous versions of Smart Integration Connector are compatible with newer versions of OneStream. For example, the OneStream Local Gateway Configuration is compatible with OneStream Platform version 8.1.0. However, it is always recommended to upgrade to the latest version.

If you perform an upgrade and have issues or do not achieve these results, contact OneStream Support.

1. Install the latest version of OneStream. The latest version can be requested and scheduled through the [OneStream Software Cloud Customer Service Catalog](#). Make a note in the details section of the ticket that you want to install and configure the Smart Integration Connector.

2. Download the Smart Integration Connector install (OneStream\_Connector\_#.#.#.zip) file from the Platform section of the [Solution Exchange](#).



3. Extract the OneStreamSmartIntegrationConnectorGateway.msi from the downloaded zip file.
4. Back up a copy of your configuration folder C:\Program Files\OneStream Software\OneStream Gateway\App\_Data before upgrading.
5. (Optional) If you are using any Custom DLLs in the Gateway Installation folder C:\Program Files\OneStream Software\OneStream Gateway, create a backup copy.
6. Follow the steps in [Setup and Installation](#) to complete your upgrade.

**NOTE:** For OneStream Platform version 8.1 and above, the default location for Reference Assembly Folder is C:\Program Files\OneStream Software\OneStream Gateway\Referenced Assemblies.

Previously, it was required that a OneStream Business Rule developer invoking a remote Smart Integration Function be aware of the data type returned and convert accordingly after the result is returned.

**Example:** An example where the returned result was a byte array involved code that appeared as follows:

```
bytesFromFile = CompressionHelper.InflateJsonObject(Of System.Byte())  
(si,objRemoteRequestResultDto.resultDataCompressed)
```

The Smart Integration Connector Gateway now provides this type of information back to OneStream and streamlines this conversion process using a newly added property called `ObjectResultValue`, which is populated.

When invoking the same operation shown above that previously required the type to be converted, a BR developer can do the following:

```
bytesFromFile = objRemoteRequestResultDto.ObjectResultValue
```

## Upgrade from 7.1 to 8.1

This is only required if you are upgrading from the 7.1 version to 8.1. If you are upgrading from 7.1 to another version, you must redo any previous setup configurations in the OneStream Smart Integration Connector Gateway Configuration Utility. Any previously configured data source name must be reconfigured in the utility. You must keep **Data Source Name**, **Connection String**, and **Database Provider** the same as the previously configured connections. You can copy this information from the System Configuration page.

Before upgrading to OneStream Platform version 8.0, follow these steps:

1. Remove all gateways under **System > Administration > Smart Integration Connector**.

**TIP:** Make a copy of the gateway name. You will use this name when creating the gateway in a version 8.1.

2. After all the gateways are removed, remove all Custom Gateway Connections under **System > System Configuration > Application Server Configuration > Database Server Connections > Custom**.

**CAUTION:** This is imperative. If you do not remove all Custom Gateway connections, you will need to contact OneStream support to resolve this issue.

- a. Make a copy of the **Database Server Name**, **Gateway**, and **Gateway Connection**. You will use these to recreate your Custom Gateway Connections in 8.1.
3. Uninstall the previous version of Smart Integration Connector Gateway from your OneStream Windows Application Server completely.
  4. After gateways and custom connections are removed, you can upgrade to 8.1.
  5. After 8.1 is installed, you can follow the rest of this guide. Make sure you use the same values you copied before removing the gateway and custom database connection.

**NOTE:** If you used any BR APIs, you will need to update them with Gateway in the name. For example, if you used `BRApi.Utilities.ExecRemoteRelayBusinessRule`, the new business rule name is `BRApi.Utilities.ExecRemoteGatewayBusinessRule`. Likewise, any existing business rules specifying a `DbProviderType` as `DbProviderType.Relay` needs to be migrated to `DbProviderType.Gateway`:

### Example:

```
drillBackInfo.DataTable = api.Parser.GetXFDataTableForSQLQuery(si, DbProviderType.Relay, "RevMgmt"..  
'needs to be migrated to:  
drillBackInfo.DataTable = api.Parser.GetXFDataTableForSQLQuery(si, DbProviderType.Gateway, "RevMgmt"
```

# Migration from VPN Considerations

If you are migrating from a VPN solution to Smart Integration Connector, there are items to take into consideration. Use the checklist below to prepare yourself for migrating from VPN to Smart Integration Connector.

**NOTE:** While migrating, a VPN and Smart Integration Connector can be used in tandem. This allows for A/B testing and validation prior to disconnecting the VPN tunnel.

Checklist Item	Complete
Establish and confirm connectivity for all your data sources including any supporting DLL files that need to be migrated.	<input type="checkbox"/>
Plan the setup and configuration through Smart Integration Connector for inventoried Data Sources. Determine connection type (Database or Direct) for each Data Source.	<input type="checkbox"/>
Establish and confirm connectivity for all your data sources.	<input type="checkbox"/>
Take inventory of what you currently use for example, business rules, dashboards, queries, grid views, drill-backs, and whitelisted endpoints for each plan for any updates needed when using Smart Integration Connector.	<input type="checkbox"/>
Set up a time with your OneStream Cloud Support Representative to plan when the VPN can be disconnected.	<input type="checkbox"/>

# Setup and Installation

## Smart Integration Connector Setup

You must set up Smart Integration Connector in this order:

1. Install the **OneStream Smart Integration Connector Local Gateway Server** (OneStreamSmartIntegrationConnectorGateway.msi) on a Windows Server 2019+ in your environment.
2. Create a Gateway in the OneStream Windows application to connect OneStream Cloud instance to a **Local Gateway**.
3. Export the gateway configuration and import this configuration to the **Gateway Settings** in the **OneStream Local Gateway Configuration**.
4. For gateways of type Database Connection, to allow connections to local databases,
  - a. Define a Local Gateway connection including **Data Sources** through the **OneStream Local Gateway Configuration**.
  - b. Test any configured **Data Sources** to confirm they are communicating properly. Note that testing direct connections may involve building test business rules to perform proper validation.
  - c. Define a custom database connection in the OneStream System Configuration Setup.

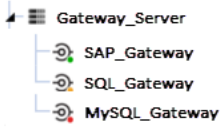
When installation is complete, you can access remote data sources using business rules, member formulas, or dashboard data adapters in OneStream through the Smart Integration Connector.

## Gateway Terms

The Smart Integration fields define the gateway. You can find more information about this below.

Relay Name	Refers to the internal namespace of the relay service that is responsible for managing the data flow for all defined Gateways. For example, arn-
------------	--

	mysite.servicebus.windows.net.
IPv4 Whitelist	Contains the list of IPs or CIDR addresses that are allowed to transfer data via SIC.
Name	<p>The name of the gateway. Gateway names are completely arbitrary and typically refer to the region (North East) or data source such as (SAP).</p> <p><b>NOTE:</b> The gateway name cannot be changed once created and they must be unique across all environments--both development and production. You can delete an existing gateway and recreate it with a new name.</p>
Description	Text describing the role and purpose for the gateway and the data sources to which it is connecting.
Gateway Server Name	This is the name of the gateway Server that the gateway is associated with. Select an existing gateway server or enter a new one.
Web API Key (Database Connections only)	This is an editable field. You can change your key as needed. If changed, it must also be changed in the Smart Integration Connector Local Gateway Server. It is designed to offer an additional layer of protection within your network when invoking APIs embedded in the Smart Integration Connector Local Gateway Server. The purpose of the Web API Key is to give you full control on who can access the data sources in your network.
Gateway Key	This is the cloud-key used to authenticate the Smart Integration Connector gateway to the customer OneStream environment. This key can be rotated in the OneStream Application by Smart

	Integration Connector Gateway administrators and must be the same in both the remote Gateway service and in OneStream.
Status	Value will be Online if the local gateway is running and returning heartbeat messages back to OneStream. If the Smart Integration Connector Local Gateway Server is unavailable, stopped, or network connectivity is interrupted, it will display Offline.
Status Indicators 	<p>Status indicators give a visual notification based on the <b>Gateway</b> status. An indicator turns green on the side menu if the <b>Gateway</b> is <b>Online</b>, red if the Gateway is <b>Offline</b>, and yellow if the <b>Gateway</b> is <b>Offline</b> but there is a newer version of the Local Gateway Server available.</p> <p>For <b>Direct Connections</b>, the yellow status should never display as these connections cannot technically report a version number back to OneStream and would either show as online or offline.</p>
Instance Count	Displays the count of active gateways. Grayed out by default. While this value is typically 1 when the gateway is online, you may have a listener count of two or more if there are redundant active gateways for high availability. By default, OneStream allows a total of five active gateways per environment. This can be increased by contacting support.
Version (Database Connections Only)	Displays the Smart Integration Connector Local Gateway Server version. This version may be different from the deployed version of OneStream and allows administrators to observe and monitor versions of Smart Integration Connector Gateway software deployed.



Bound Port at Gateway	Remote port bound to Gateway endpoint. Relational Database Gateways should default to 20433 while direct connections allow any port running on a remote host to be used.
Remote Gateway Host (Direct Connections Only)	Remote port host to Gateway Server. Used if surfacing an endpoint such as an SFTP Server. This could be the hostname or IP address on the network that the Gateway Server resides in. example: 172.168.4.7 or sftp.mycompany.com
Bound Port in OneStream (Direct Connections Only)	This is an optional customer defined port that can be referenced in data management or business rules to directly access services such as sFTP and WebAPI. This must be a globally unique port in a OneStream deployment per direct connection and should be a TCP port number > 1024 and <65535. When creating the gateway, use the default of -1 and OneStream will automatically assign an open port.
Gateway failures reporting interval (min)	Minutes to wait between reporting gateway failures into the OneStream Error Log. The default is five minutes and the max is 1440 minutes. If a gateway is unreachable, an item is put in the error log using this interval value in minutes and can be adjusted.

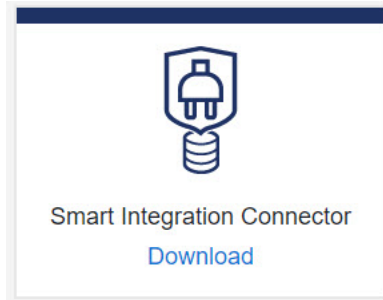
## Local Gateway Server Installation

Smart Integration Connector is available in OneStream from the **System > Administration** tab.

## Setup and Installation

---

1. Download the Smart Integration Connector install (OneStream\_Connector\_#.#.#.zip) file from the Platform section of the [Solution Exchange](#).



2. Copy the **Smart Integration Connector Local Gateway Server** installer to a Windows Server within your environment.
3. Run the installer as an administrator. Accept all the default prompts. When completed, the Local Gateway Server will be installed on your Windows Server.

**IMPORTANT:** If you are upgrading, you must follow steps 4-7.

4. Run the **OneStream Local Gateway Configuration Utility**.
5. The **XFGatewayConfiguration.xml** file will open by default.

**IMPORTANT:** Do not change the name of the XFGatewayConfiguration.xml file. The OneStream Smart Integration Connector Gateway Service only references this XFGatewayConfiguration.xml file upon start-up. The **Save As** functionality is used to create a backup of the file. Do not rename, move, or change the location of the XFGatewayConfiguration.xml file.


6. Save the configuration file.
7. Follow the dialog box and restart the service.

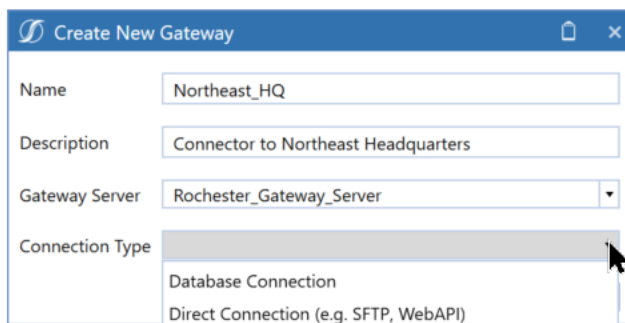
## Create a New Gateway

Gateways are used to connect OneStream to the Smart Integration Connector Local Gateway Server over the Azure Relay. You will establish whether the gateway is a direct or database connection. After the gateway is created, you will need to copy the configuration to the Smart Integration Connector Local Gateway Server using the OneStream Local Gateway Configuration.

## Setup and Installation

---

1. Go to **System > Administration > Smart Integration Connector**.
2. Click  **Create New Gateway**.
3. Enter the **Name** and **Description**. For descriptions of the fields in steps 3-6, refer to the [Gateway Information section](#).
4. Select the **Gateway Server** from the drop-down, or enter a new Gateway Server name.
5. From **Connection Type**, select a Database Connection or Direct Connection. You will have to enter different information depending on the connection type you select.



**NOTE:** The Gateway name cannot be changed once created and must be deleted and re-created.

## Create a Database Connection

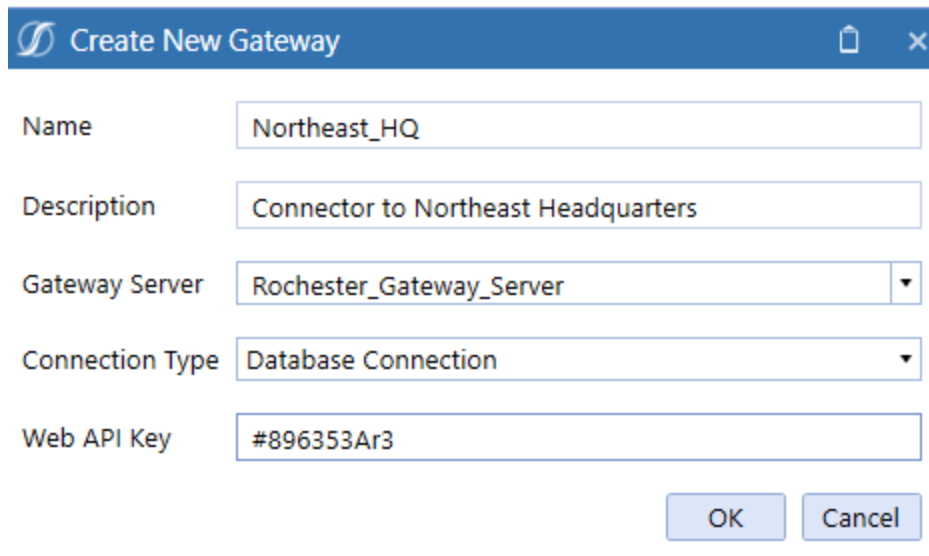
A database connection is used to connect to relational databases such as SQL Server, MySQL, etc. using ODBC, OleDB or .NET drivers and is also necessary for remote Smart Integration Functions to execute. It is recommended to have at least one Database connection endpoint per Gateway Server even if relational databases will not be accessed by OneStream. The Local Gateway Configuration Utility facilitates configuration of required credentials for associated local gateway. The identification of a local gateway connection must correspond to a custom database connection established to the OneStream Application Server.

After you create a new gateway, you can complete the database connection by following these steps:

## Setup and Installation

---

1. From **Connection Type**, select **Database Connection**. For descriptions of the fields in steps 1 and 2, refer to the [Gateway Information section](#).
2. Enter a Web API Key.



**Create New Gateway**

Name:

Description:

Gateway Server:

Connection Type:

Web API Key:

OK Cancel

**NOTE:** The Web API Key is used as an additional layer of security when communicating with the Smart Integration Connector Local Gateway Server internal APIs. WebAPI keys are not required, but recommended to enhance security and can be modified or added at anytime. The Local Gateway Service introduces a WebAPI exposed only to OneStream and bound only to localhost on the server it is deployed to. This WebAPI is inaccessible on the remote network. If the Local Gateway Service is bound to other network interfaces, it's suggested to use the WebAPI as a mechanism to enhance security on the remote network preventing unauthorized use of OneStream WebAPIs.

## Create a Direct Connection

A gateway direct connection represents a point-to-point channel to specific remote network resources such as an sFTP server or Web API (including iPaaS services).

**NOTE:** It is required to have at least one Database connection to use a Direct Connection as the database connection is used to monitor the availability of the remote Smart Integration Connector Gateway server.

The existence of a database connection does not necessarily mean it must be used or configured if only Direct Connections are desired.

After you create a new gateway, you can configure the direct connection by following these steps:

1. From **Connection Type**, select **Direct Connection (e.g, SFTP, WebAPI)**. For descriptions of the fields in steps 1-4, refer to the [Gateway Information section](#).
2. Enter the Bound Port at Gateway. This port represents the well-known TCP service to expose from an on-premises host such as SFTP which would equate to port 22.

**NOTE:** Note that the remote service port is required to configure the connection and may require consultation with network or IT resources to obtain. It is also required that any firewalls between the Local Gateway Server and the remote host allows traffic to the destination port specified.

3. Enter the Remote Gateway Host (for example, localhost). This represents the remote host name or IP address accessible by the OneStream Smart Integration Connector Local Gateway Server. If the host or IP address is accessible or resolvable from the OneStream Smart Integration Connector Gateway service, or using remote resources accessible through on-premises WAN, it can be exposed for use.
4. Enter a Bound Port in OneStream. It is recommended to use -1 for this value as the OneStream application servers will locate an unused and available port to map to this connection. This port number must be globally unique across all application servers in a OneStream deployment and care should be taken if a specific port is specified. This is the port that is then used to access the remote host via business rules, data management jobs, and so on. from OneStream application servers to allow network traffic to traverse to the remote host and port.
5. Using this direct connection in OneStream is done by accessing localhost: [Bound Port In OneStream] which will tunnel traffic back to the configured remote Gateway Host to the configured bound port at gateway.

## Setup and Installation

---

- a. Example: Remote sFTP server at 172.168.3.4 listening on port 22.
- b. Bound Port in OneStream is configured as port 45000. Note that when -1 is used, the selected port number is available/displayed after saving and also surfaced in the OneStream Error Log.
- c. In OneStream Business Rules, you can access the remote host by connecting to localhost:45000.
- d. In a OneStream Business Rule, this port can also be obtained in code allowing this port number to be changed without updating Business Rules:

```
Dim gatewayDetails As GatewayDetails = BRapi.Utilities.GetGatewayConnectionInfo(si,
"northamerica_sftp")
Dim remotePort = gatewayDetails.OneStreamPortNumber
```

**Create New Gateway**

Name: northamerica\_sftp

Description: North America sFTP server

Gateway Server: North\_America

Connection Type: Direct Connection (e.g. SFTP, WebAPI)

Bound Port at Gateway: 22

Remote Gateway Host: 172.168.3.4

Bound Port in OneStream: -1


OK Cancel

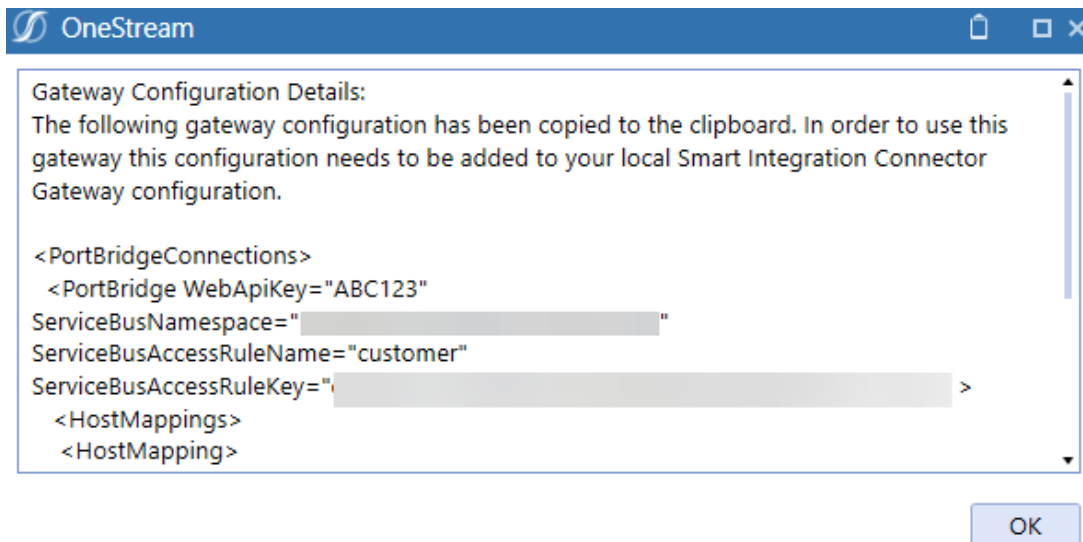
## Export and Import the Gateway Configuration


You must copy the gateway configuration settings and paste them into your Smart Integration Connector Gateway to establish the connection.

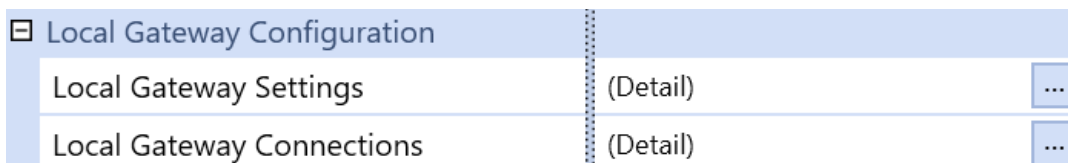
## Setup and Installation


---

1. Go to **System > Administration > Smart Integration Connector**.
2. Select the Gateway to export.
3. Click  **Export Gateway Configuration**. The Gateway Configuration Details are copied to the clipboard.




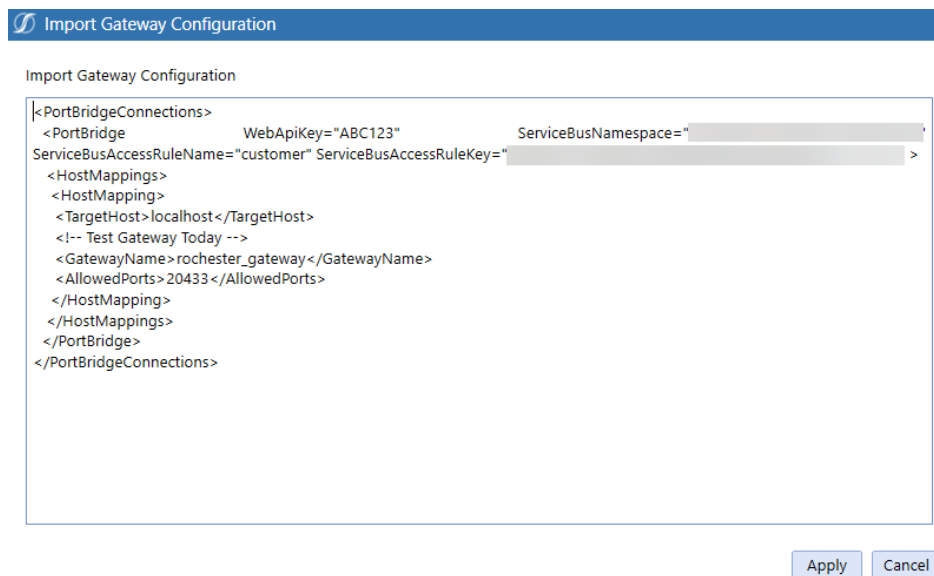
4. On your Windows Server, open the **OneStream Local Gateway Configuration**. This runs as administrator by default.
5. Click  next to **Local Gateway Settings**.



6. Click  next to **Local Gateways**.



7. Import  the previously copied **Gateway Configuration**.



8. Click **Apply**.
9. Click **Test Connection** to test the connection.





10. Click **OK** twice.
11. Save the configuration.
12. Click **Yes** to apply the changes and restart the Local Gateway Server.

## New Gateway Key Generation

Smart Integration Connector administrators can rotate the Gateway Key maintained by the underlying cloud service, however it must be the same for both the Smart Integration Connector local gateway and the gateway configuration to function properly.


1. Select an existing gateway.
2. Click **Regenerate Gateway Key for Selected Gateway**.
3. You must re-export your Gateway Configuration and apply the new settings through the OneStream Local Gateway Configuration.
4. Click **OK**.

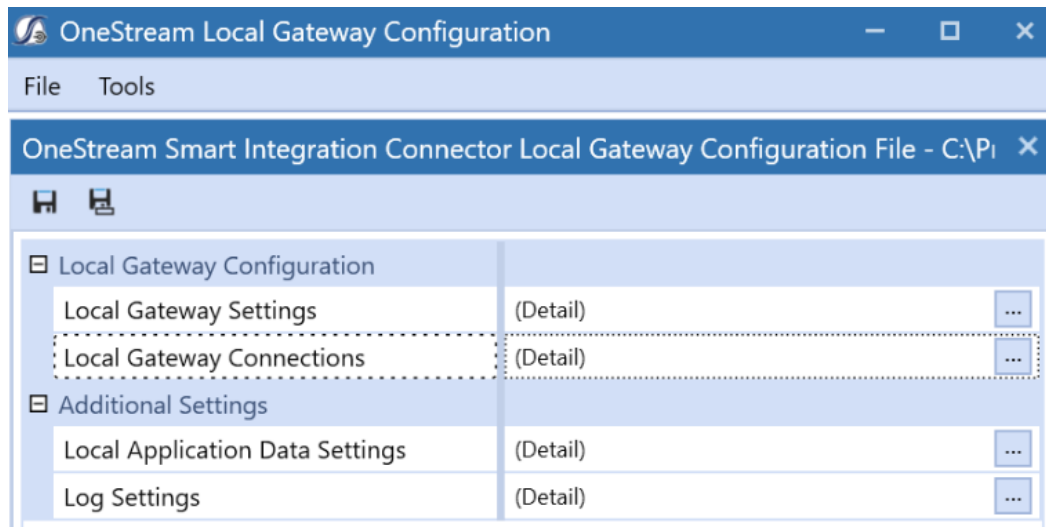
## Create a Local Gateway Connection to a Data Source


A data source contains the name, connection string, and database provider for the database of your choice. You can set up a PostgreSQL, SQL, Oracle, OleDb, MySQL, ODP.net, or Microsoft ODBC connection here. The data source is configured using the Local Gateway Configuration Utility. The utility was installed as part of the Smart Integration Connector Local Gateway install.

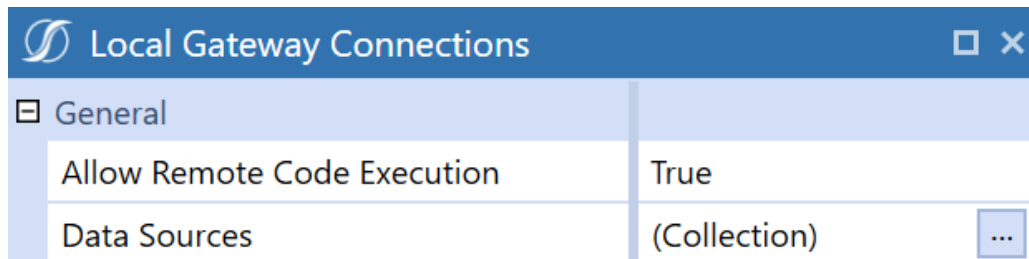
## Setup and Installation

---

1. Start the **OneStream Local Gateway Configuration**.
2. Click  to configure **Local Gateway Connections** details to set up the **Data Sources** to local databases, APIs, or other on-premises resources.



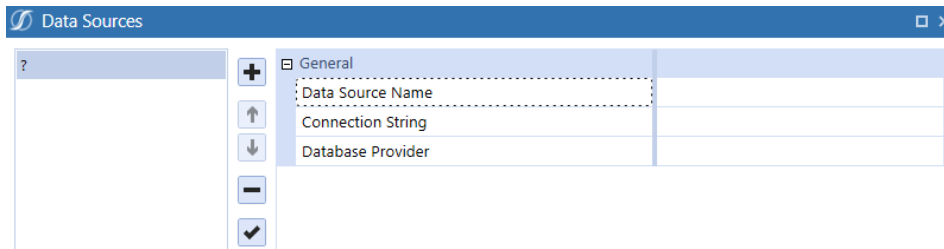
3. Click  next to **Data Sources**.



4. Click  **Add Item** to add a new data source.

## Setup and Installation

---



5. Enter the **Data Source Name**, **Connection String**, and select a **Database Provider**. You can add as many data sources as necessary. The **Data Source Name** must be unique for each connection defined within a specific OneStream Smart Integration Connector Local Gateway Server. Names can be re-used across deployed instances of the Windows Service across your network. See the examples below for connection string examples to a variety of relational data sources such as PostgreSQL, SQL, and ODBC, and Oracle. **Connection Strings** are encrypted automatically. You can edit the plain text string by clicking the ellipsis.



**NOTE:** Oracle databases require drivers and specific configuration provided by Oracle.

6. Click **OK** to save your configuration.

**NOTE:** The connection strings below include user IDs and passwords. If Integrated security is desired, you can configure the OneStream Smart Integration Connector Gateway Service to run under a specific service account versus saving usernames and passwords in connection strings.

## Microsoft SQL Server

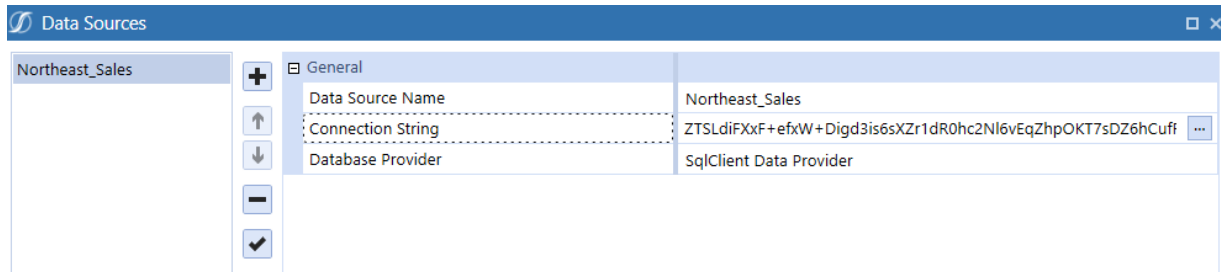
Below is an example for setting up a SQL database using the SqlConnection provider.

1. Click  next to **Data Sources**.
2. Click  **Add Item** to add the data source.
3. **Data Source Name:** Northeast\_Sales

## Setup and Installation



---


4. **Connection String:**  
with UserID / Password: Data Source=localhost;Initial Catalog=Sales\_DB;Persist Security Info=True;User ID=sa;Password=\*\*\*\*\*;Max Pool Size=1000;Connect Timeout=60;  
  
using Integrated Security: Data Source=localhost;Initial Catalog=Sales\_DB;Trusted\_Connection=True;
5. From Database Provider, select **SqlClient Data Provider**.
6. Click **Test Connection** to test the data source.
7. Click **OK** to save.



## MySQL Data Provider

Below is an example for setting up a MySQL Data Provider.

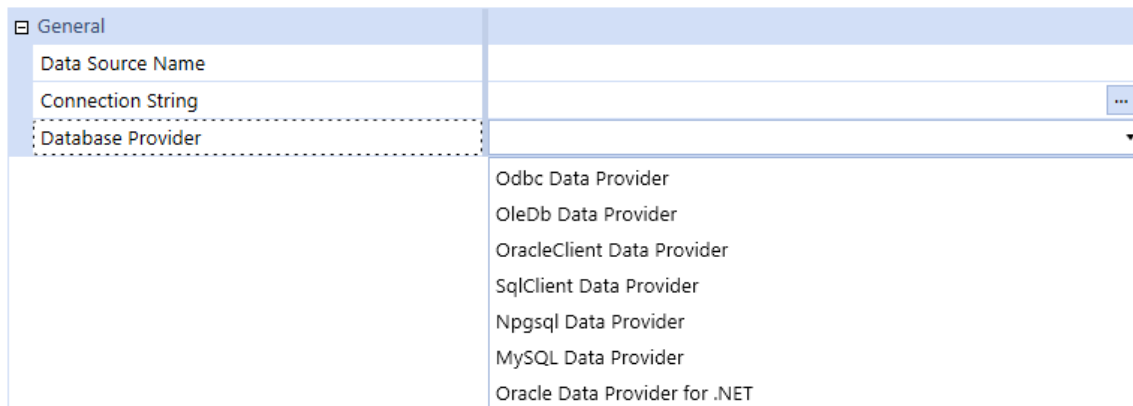
1. Click  next to **Data Sources**.
2. Click  **Add Item** to add a new data source.
3. **Data Source Name:** Sales\_UK
4. **Connection String:** Server = localhost;Port=3306;uid=root;pwd=Onestream123!;database=gatewaymysql;
5. From **Database Provider**, select **MySQL Data Provider**.

6. Click  **Test Connection** to test the data source.
7. Click **OK** to save.

## Oracle Database Examples

Connecting to Oracle requires the download and configuration of the Oracle Data Access Components (ODAC) obtained directly from Oracle's website. Follow the steps below to get access to these drivers and files.

1. Go to the latest web page for [Oracle .NET and Visual Studio ODBC Downloads for Oracle Database](#).
2. After installation, the ODP.NET Provider will display as an available Database Provider in the utility when adding a new data source.
3. The connection string for Oracle databases can be set up to either reference or require a locally defined tnsnames.ora file for the requested data sources.






Example Connection Strings:

- **Oracle Data Provider for .NET:** Data Source=oracletest;User Id=OneStream1;Password=\*\*\*\*\*;
- **Oracle Data Provider without TNSNames.ora:** Data Source=(DESCRIPTION=(ADDRESS\_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=MyHost)(PORT=MyPort))) (CONNECT\_DATA=(SERVER=DEDICATED)(SERVICE\_NAME=MyOracleSID))); User Id=myUsername;Password=myPassword;



### OracleClient Database Provider

Below is an example for setting up a OracleClient database provider.

1. Click  next to **Data Sources**.
2. Click  **Add Item** to add the data source.
3. **Data Source Name:** Sales\_EMEA
4. **Connection String:** Data Source=oracletest;User Id=OneStream1;Password=\*\*\*\*\*
5. From **Database Provider**, select **OracleClient Data Provider**.
6. Click  **Test Connection** to test the data source.
7. Click **OK** to save.


### Oracle Data Provider for .NET

Below is an example for setting up a Oracle Data Provider for .NET.

1. Click  next to **Data Sources**.
2. **Data Source Name:** Sales\_SouthAmerica
3. **Connection String:** Data Source=oracletest;User Id=OneStream1;Password=\*\*\*\*\*
4. From **Database Provider**, select **Oracle Data Provider for .NET**.
5. Click  **Add Item** to add a new data source.




## Setup and Installation

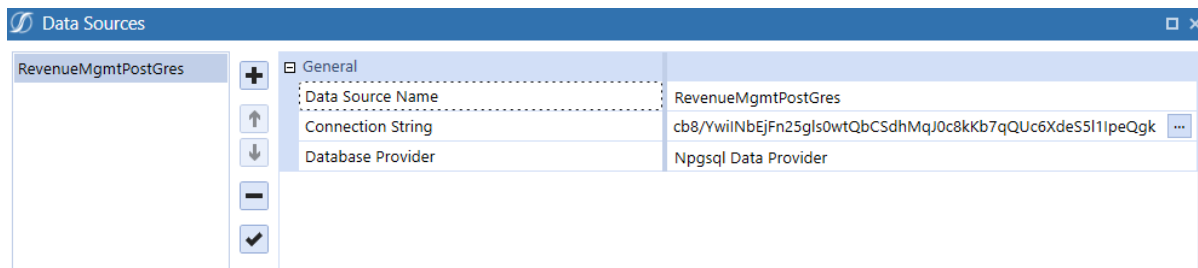
---

6. Click  **Test Connection** to test the data source.
7. Click **OK** to save.

## PostgreSQL (Npgsql Data Provider)

Below is an example for setting up a PostGres database.

1. Click  next to **Data Sources**.
2. Click  **Add Item** to add the data source.
3. **Data Source Name:** RevenueMgmtPostGres
4. **Connection String:** Server=localhost;Port=5432;Database=revmgt;User Id=onestream;Password=\*\*\*\*\*;
5. From **Database Provider**, select **Npgsql Data Provider**.
6. Click  **Test Connection** to test the data source.
7. Click **OK** to save.






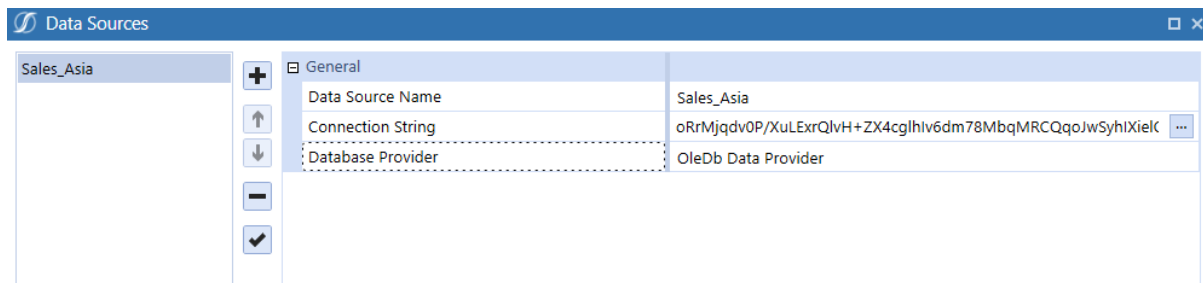
## OleDb Data Provider

Below is an example for setting up an Oracle database. This does not require additional download and configurations.

## Setup and Installation



---

1. Click  next to **Data Sources**.
2. Click  **Add Item** to add the data source.
3. **Data Source Name:** Sales\_Asia
4. **Connection String:** Provider=OraOLEDB.Oracle;Data Source=localhost:1521/XE;Initial Catalog=myDataBase;User Id=myUsername;Password=\*\*\*\*\*;
5. From **Database Provider**, select **OleDb Data Provider**.
6. Click  **Test Connection** to test the data source.
7. Click **OK** to save.



## ODBC Data Provider


Below is an example for setting up a ODBC data source for Oracle.

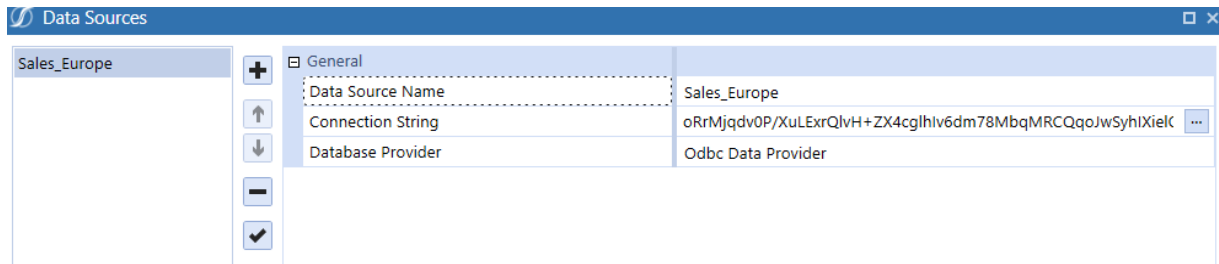
1. Click  next to **Data Sources**.
2. Click  **Add Item** to add the data source.
3. **Data Source Name:** Sales\_Europe



## Setup and Installation

---

4. **Connection String:** Driver={Microsoft ODBC for Oracle};Server=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=199.199.199.199)(PORT=1523))(CONNECT\_DATA=(SID=dbName)));Uid=myUsername;Pwd=myPassword;
5. From **Database Provider** , select **Odbc Data Provider**.
6. Click  **Test Connection** to test the data source.
7. Click **OK** to create the new source.
8. Click **Save**.



## (Optional) Remove UserID and Passwords by Integrated Security

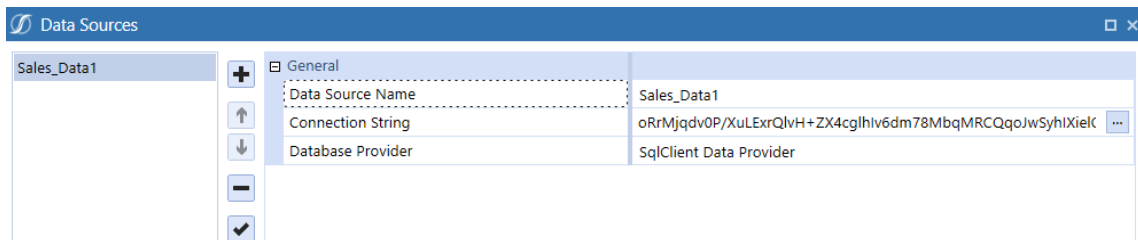
You can remove plain text UserID and Passwords from connection strings in Smart Integration Connector if your organization has concerns over credential storage in the Smart Integration Connector Gateway configuration file. This requires running the Windows Service under a **Service Account** identity and using integrated security to connect to remote data sources which eliminates local storage of any plain-text credentials. Additionally, ODBC data sources can be defined (using a system DNS) to remove credentials from the configuration file.

## Update the Local Gateway Connection String

1. Open your **OneStream Local Gateway Configuration**.
2. Open a **Local Gateway Connection**.

## Setup and Installation

3. Navigate to the Connection String and use an Integrated or Trusted Security string. For example: Data Source=localhost,Initial Catalog=OneStream\_GolfStreamDemo\_2022;Trusted\_Connection=True;



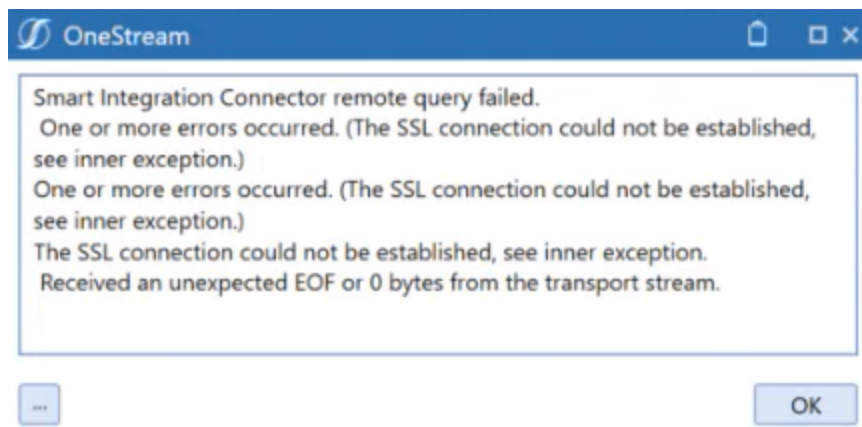
**NOTE:** Trusted Connections use the UserID and password you use to log into the Windows Server.

**NOTE:** The example above is for SQL server. Trusted connections vary by Data Provider type.

4. Click **OK**.
5. Save your **Data Source**.

## Update Permissions on the Service

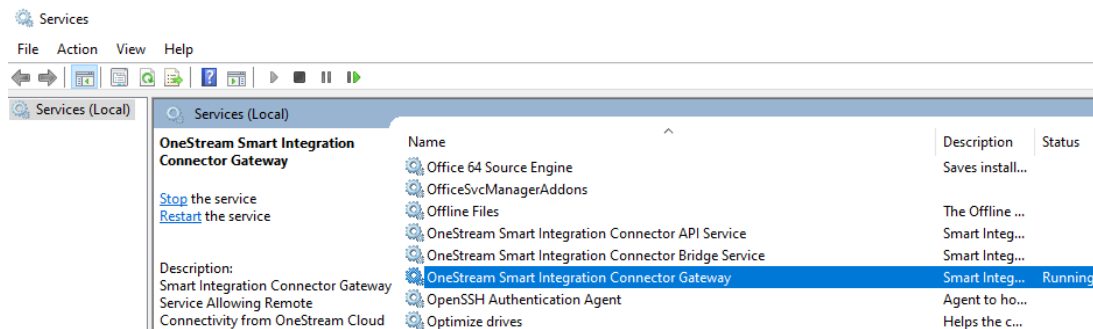
Next, you need to update the service to run as the user. If the service is not updated, the connection does not update and errors will occur.



## Setup and Installation

---

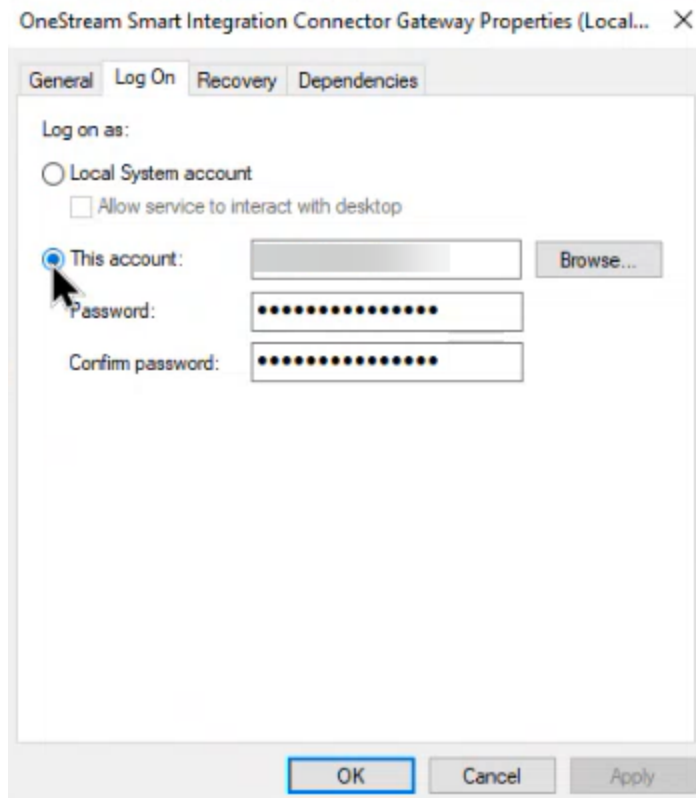
1. Open Windows Services.
2. Navigate to **OneStream Smart Integration Connector Gateway**. The service should be running.



3. Right-click and open **Properties**.
4. Click the **Log On** tab. Typically, this will default to the **Local System account**.

**IMPORTANT:** Before moving on to the next step, ensure that you have the appropriate permissions and approvals from your IT Administrators to complete the Log On change. You may need to access Microsoft SQL Server Management Studio to verify permissions.

5. Change this from **Local System account** to **This account** and enter your domain and/or login that has access to the datasource. Depending on how your SSO is configured, your account could require your domain name, UserID, and password. Contact your IT Administrator if you have questions on your account domain.



6. Click **Apply**.
7. Click **OK**.
8. Right-click and select **Restart** to restart and update the service.

## Test the Updated Integrated Connection String

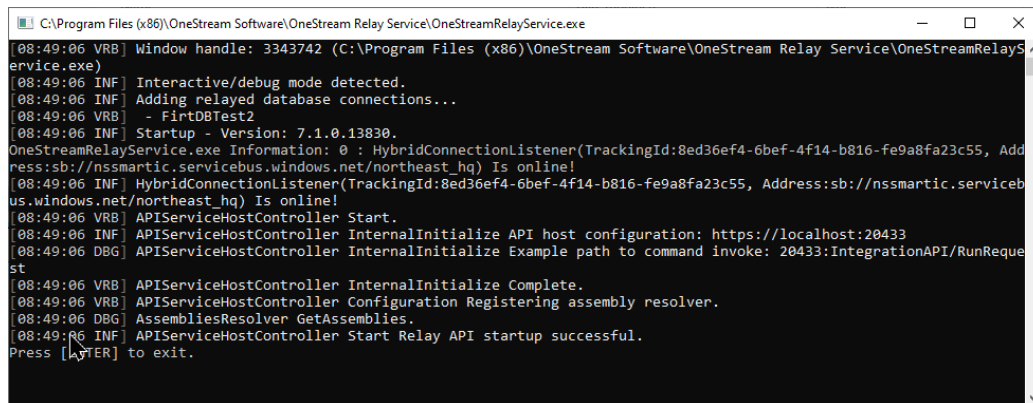
You should test your connection through a Data Adapter query to verify your access to Smart Integration connector. An alternate SQL Query to pulling the first 10-50 rows is sufficient. See [Data Adapters Example](#).

## Test the Gateway

1. You can test the gateway by double-clicking the *OneStreamGatewayService.exe* file located in the installation folder.

**NOTE:** The Smart Integration Connector Gateway Windows Service must be in a stopped state to run in the console for test purposes.

The following command window is displayed:



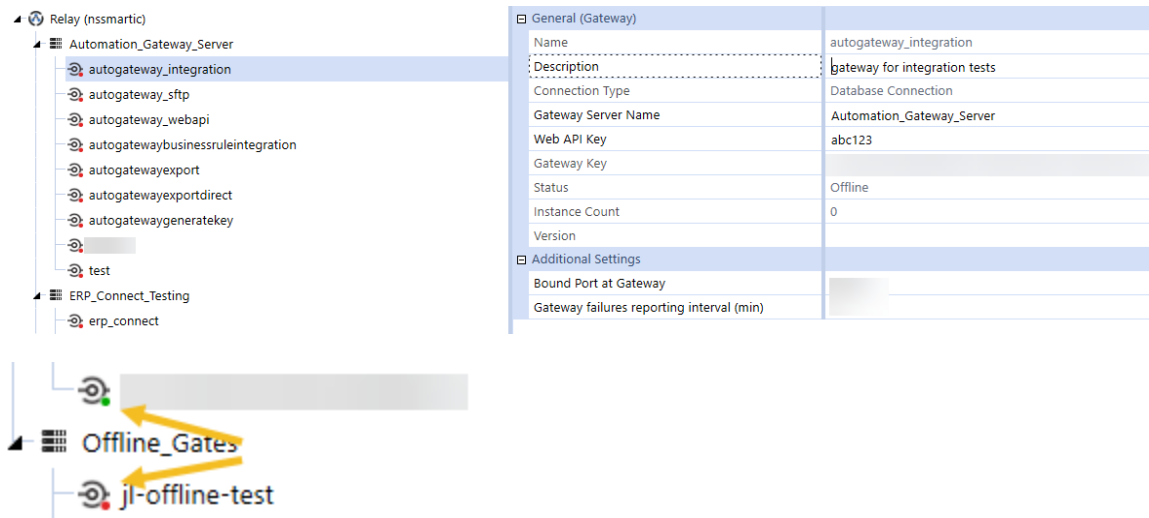
```
C:\Program Files (x86)\OneStream Software\OneStream Relay Service\OneStreamRelayService.exe
[08:49:06 VRB] Window handle: 3343742 (C:\Program Files (x86)\OneStream Software\OneStream Relay Service\OneStreamRelayService.exe)
[08:49:06 INF] Interactive/debug mode detected.
[08:49:06 INF] Adding relayed database connections...
[08:49:06 VRB] - FirtDBTest2
[08:49:06 INF] Startup - Version: 7.1.0.13830.
OneStreamRelayService.exe Information: 0 : HybridConnectionListener(TrackingId:8ed36ef4-6bef-4f14-b816-fe9a8fa23c55, Address:sb://nssmartic.servicebus.windows.net/northeast_hq) Is online!
[08:49:06 INF] HybridConnectionListener(TrackingId:8ed36ef4-6bef-4f14-b816-fe9a8fa23c55, Address:sb://nssmartic.servicebus.windows.net/northeast_hq) Is online!
[08:49:06 VRB] APIServiceHostController Start.
[08:49:06 INF] APIServiceHostController InternalInitialize API host configuration: https://localhost:20433
[08:49:06 DBG] APIServiceHostController InternalInitialize Example path to command invoke: 20433:IntegrationAPI/RunRequest
[08:49:06 VRB] APIServiceHostController InternalInitialize Complete.
[08:49:06 VRB] APIServiceHostController Configuration Registering assembly resolver.
[08:49:06 DBG] AssembliesResolver GetAssemblies.
[08:49:06 INF] APIServiceHostController Start Relay API startup successful.
Press [ENTER] to exit.
```

2. Correct any errors that are displayed in the command window.

**NOTE:** If the command window output does not proceed beyond the "APIServiceHostController Start Relay API startup successful." line, this indicates that the outbound traffic over port 443 to the Azure Relay is blocked. Open the port to resolve this issue.

3. In the OneStream Windows Application client, refresh Gateway Details from **System > Administration > Smart Integration Connector > Your gateway**.
  - You should see the **Instance Count** change from 0 to 1.
  - The **Status** change from **Offline** to **Online**. Additionally, status indicators turns green on the side menu if the **Gateway** is **Online**, red if the Gateway is **Offline**, and yellow if the **Gateway** is **Offline** but there is a newer version of the Local Gateway Server available. See the second screenshot under this step for a close up of the indicators.
  - The **Version** field will show the version of the running Smart Integration Connector Gateway.

## Setup and Installation

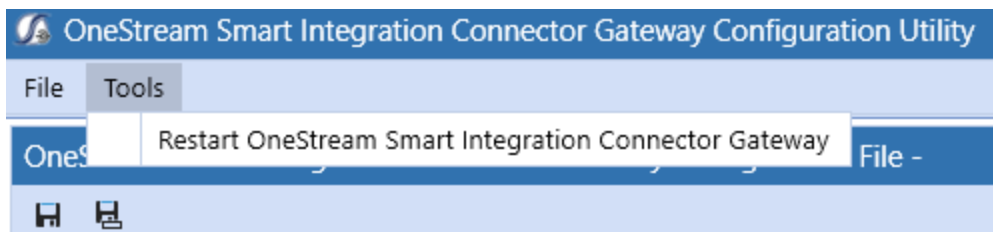


4. Press **Enter** twice on the keyboard to stop the service in the command window and then close the command window.

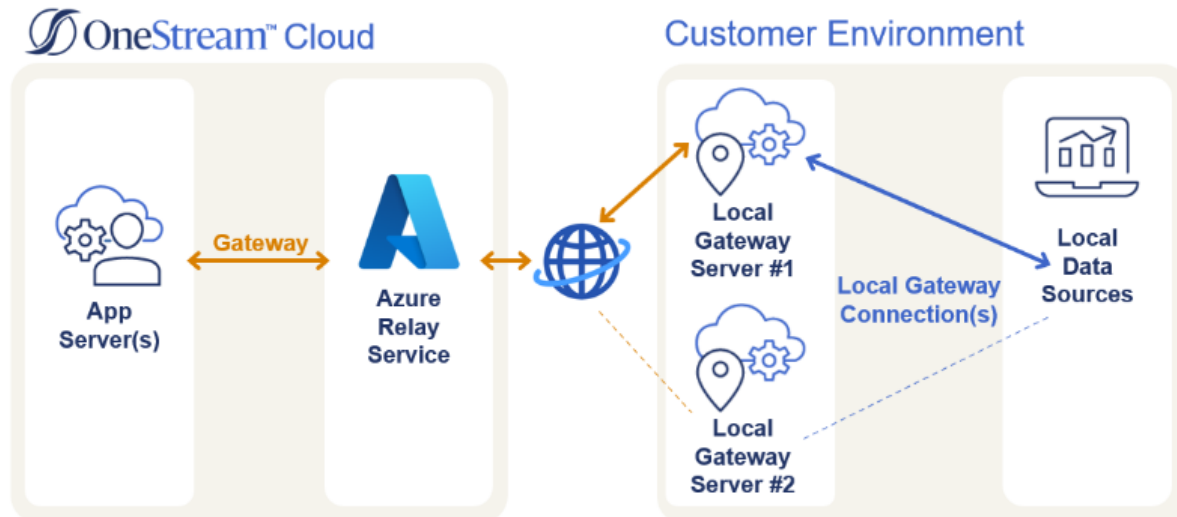
## Restart OneStream Smart Integration Connector Gateway

After communication has been verified, the following Windows Service needs to run in order to maintain communication with the OneStream Cloud instance. By default, these services are set to start after a Windows reboot. You can also manually start them using the Windows Service control manager or the command line using the net start/net stop commands. If you're having issues restarting the service, refer to [Troubleshooting](#).

1. Open the **OneStream Local Gateway Configuration**.
2. Click **Tools > Restart OneStream Smart Integration Connector Gateway**.



### Redundant and Fail-over Gateways



The Smart Integration Connector Local Gateway Server can be installed on a separate Windows Server to operate as a fail-over. The Local Gateway Server Gateway establishes connection to the Relay that becomes the ac/primary Local Gateway Server instance while the second Local Gateway Server environment remains idle until the primary goes offline. The second Local Gateway Server Gateway would be the fail-over in this scenario and automatically accept traffic if the primary server instance were to go offline.

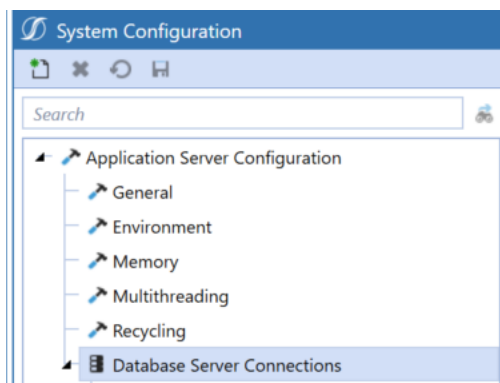
1.
  - a. Since the Data Source Connection Strings are encrypted, you will need to re-enter the connection string for each Data Source.
    - a. Click **Tools > Local Gateway Connections > Data Sources**.
    - b. Select a **Data Source** and select the **Connection String**.
    - c. Select **OK** to provide a new connection string.
    - d. Delete the encrypted text and replace it with a valid connection string from the primary server.
    - e. Select **OK** to encrypt the connection string and close the dialog box.
    - f. Repeat steps A through F for all the remaining data sources.

- g. Click **OK** to close the **Data Sources**.
- h. Click **OK** to close the **Local Gateway Connections**.
- i. Click **Save** to save the **Local Gateway Configuration**.
- j. Click **Yes** to restart the service.

## Define Custom Database Connections in OneStream System Configuration Setup

Now that the gateway is set up and communicating with the Smart Integration Connector Gateway, the final step is to set up the location of the remote data source in OneStream. To continue adding the Custom Database Connection, you must assign a user to the `ManageSystemConfiguration` role.

1. Go to **System > Administration > System Configuration**.
2. Select **Application Server Configuration > Database Server Connections**.



3. Select  **Create Item** to create a new **Custom** database server connection.

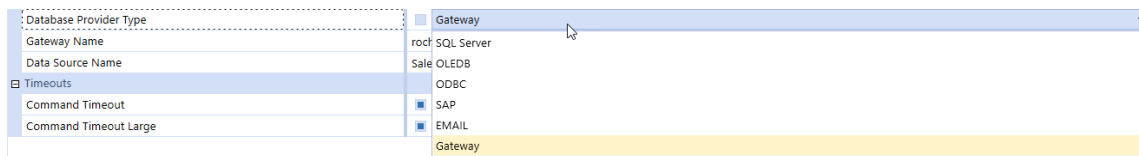
**NOTE:** If the only fields displayed are Name and External Database properties, verify that the current user is assigned to the `ManageSystemConfiguration` role.

4. Enter the **Name** of the **Database Server Connection**.
5. For **Database Provider Type**, select **Gateway**.



## Setup and Installation

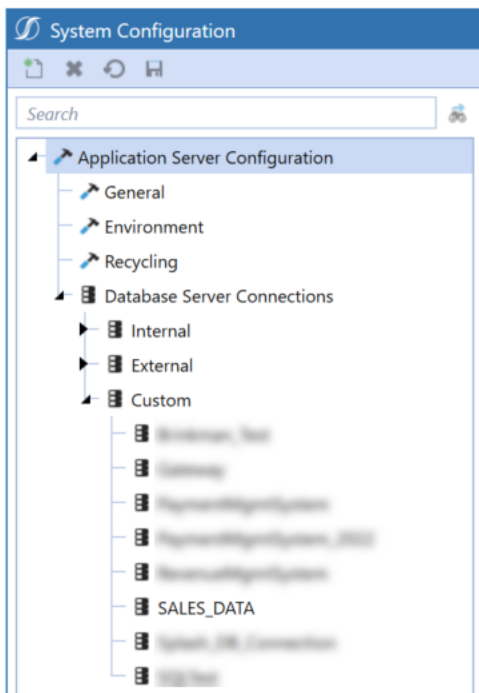
---



6. The Gateway Name drop-down menu will be populated with a list of configured Gateways. Select the Gateway.
7. After the Gateway is selected, the Data Source Name drop-down menu populates with a list of the Local Gateway Server Database Connections.
8. Select a Database Connection from the drop-down menu.

**NOTE:** If the remote data source is not displayed or the Gateway is offline, you can select Custom to allow the data source to be manually specified.

9. Click **Save** to complete the configuration.
10. Verify the customer database connection is under **Custom**.



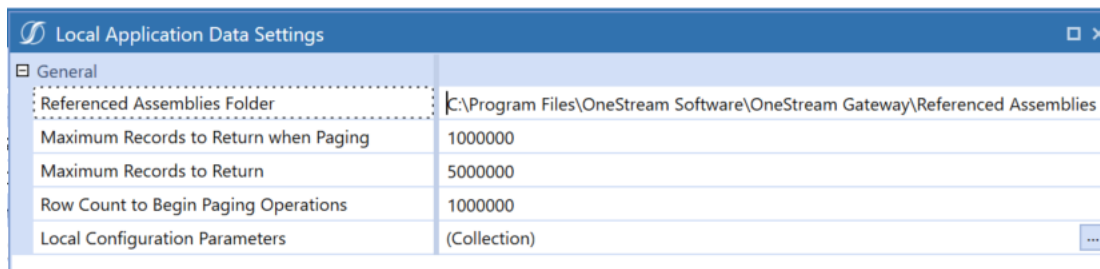
---

## Smart Integration Additional Settings

### Local Application Data Settings

Additional application configurations can be applied within the Local Application Data Settings.

Once you open a configuration file within the utility, open Local Application Data Settings.



Local Application Data Settings	
General	
Referenced Assemblies Folder	C:\Program Files\OneStream Software\OneStream Gateway\Referenced Assemblies
Maximum Records to Return when Paging	1000000
Maximum Records to Return	5000000
Row Count to Begin Paging Operations	1000000
Local Configuration Parameters	(Collection) <span>...</span>

You can:

- Reference a location to additional DLLs that will be used in remote business rules.
- Adjust the number of records returned. These are optional and are only defined if needed or if further tuning is necessary by a consultant or as instructed by Support.
- Store Configuration Parameters and associated values.

### Referenced Assemblies Folder

The Referenced Assemblies Folder specifies the location of customer-supplied DLLs that can be referenced when remote Smart Integration Functions are compiled and executed. The default value is C:\Program Files\OneStream Software\OneStream Gateway\Referenced Assemblies.

**NOTE:** If you are integrating with SAP, according to OneStream Platform version 8.0 and above, ERPConnect45.dll are not included by default. ERPConnect and supporting DLLs will need to be added to your Referenced Assemblies Folder. Refer to [Support for SAP Integration](#) .

This location path is used to locate Referenced Assemblies (DLLs) required to run remote Smart Integration Functions. You will need to add the DLL name to the Smart Integration Functions Referenced Assemblies property.

---

## Record Count Adjustments

### Maximum Records to Return when Paging

Defaults to 1000000 and defines the number of rows to return per page/block to OneStream APIs. This value is used only when greater than the "Row Count to Begin Paging Operations" rows are returned from a query. Example: If the query returns 3 million rows and Row Count to Begin Paging is set to 1 million, there would be 3 blocks of 1 million rows returned to OneStream.

**NOTE:** Maximum Records to Return when Paging, Maximum Records to Return, and Row Count to Begin Paging Operations are optional and should only be applied by a OneStream consultant or OneStream Support.

### Maximum Records to Return

Defaults to 5000000 and is the maximum number of rows that can be returned from any one query.

The maximum recommended number of records to return is 5 million and is the default. Additional RAM/CPU resources would be required on the Smart Integration Connector Gateway Server and on the remote database server to surface large quantities of data. If this limit is exceeded, you will receive a "Smart Integration Connector Remote Query" error.

**NOTE:** Maximum Records and Row Counts Settings: When large data volumes are returned (over 1000000 rows), to maintain performance and reliability, Smart Integration Connector automatically transfers the data in pages.

**NOTE:** Smart Integration Connector has a threshold limit of 5 million rows and 5GB.

**NOTE:** It is strongly recommended that you review any queries that return more than 1 million rows with your Database Administrator, because additional tuning may be required. Tuning these queries will improve performance, reduce resource usage, and make them more efficient.

---

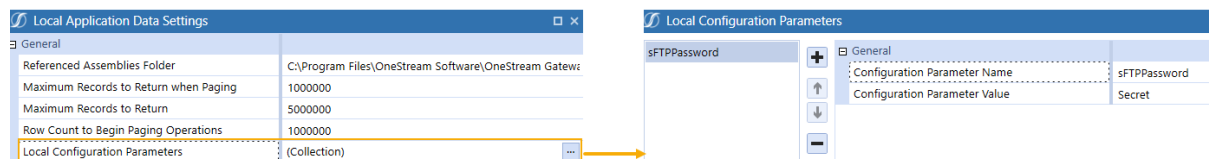
## Row Count to Begin Paging Operations

Defaults to 1000000 and is the number of rows returned before the dataset is returned through pages/blocks.

## Local Configuration Parameters

This is where you can set key value pairs, such as Web API keys, usernames, and passwords, that can be referenced from business rules. These key value pairs are defined as Configuration Parameter Name and Configuration Parameter Value.

For example, the **Configuration Parameter Name** is sftpPassword. Sensitive information, such as the password, is stored in the **Configuration Parameter Value** on the Local Gateway Server and does not need to be stored in the OneStream Windows Application.

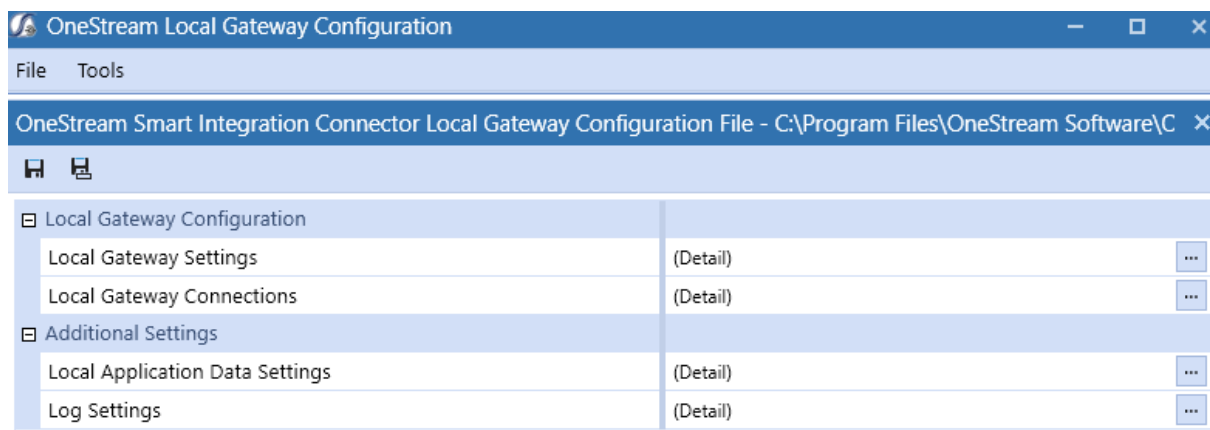


Then, in a business rule, you can reference the Configuration Parameter Name and do not need to know the password or other sensitive information that is stored in the Configuration Parameter Value. For example, in the following business rule the sftpPassword Configuration Parameter Name is referenced. The GetSmartIntegrationConfigValue API can be used in a Smart Integration Function to reference the Configuration Parameter Name, which may be needed in a business rule to access a local data source.

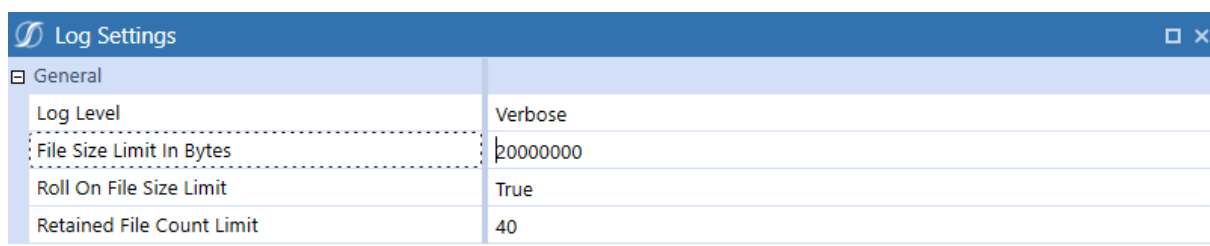
```
Dim passwordString As String = APILibrary.GetSmartIntegrationConfigValue("sftpPassword")
```

## Log Settings

The service uses Serilog for application-level logging and exposes options for controlling naming convention, growth limits, and retention details. For example you can change the verbosity of log messages by changing the **minimum-level** setting from Verbose to Informational. If a catastrophic error happens, you can check the Windows event logs to review the errors. You can edit the **Log Settings** from the **OneStream Local Gateway Configuration Utility**.



Click  to access **Log Settings**.



- **Log Level** descriptions:
  - **Verbose:** The noisiest level, rarely (if ever) enabled for a production application.
  - **Debug:** Used for internal system events that are not necessarily observable from the outside, but useful when determining how something happened.
  - **Information:** Used to describe things happening in the system that correspond to its responsibilities and functions. Generally, these are the observable actions the system can perform. This is recommended for production environments.
  - **Warning:** Service is degraded, endangered, or may be behaving outside of its expected parameters.
  - **Error:** Logging of situations where functionality is unavailable or a recoverable error condition occurred.

- 
- **Fatal:** Only the most critical level items would be logged, requiring immediate attention.
  - **File Size Limit in Bytes:** The maximum size for the log file, in bytes, before creating a new file for the day. The default is 20 MB.
  - **Roll On File Size Limit:** When a log file reaches the specified number of bytes, a new log file is generated.
  - **Retained File Count Limit:** Number of log files to retain. If logs do not exceed the limit in bytes (one file/day), this would allow for the configured value (with 40 days being the default) of log retention. If the Smart Integration Service is used heavily and log files are set to higher levels of verbosity, this could result in fewer days of log retention. Ensure that the growth rate and retention periods align with your organizational requirements.

The default location for log files is:

*%programdata%\OneStream Software\OneStreamGatewayService\Logs.*

# Advanced Networking and Whitelisting

## Whitelist the Azure Relay to your Firewall

You can whitelist the Azure Relay to your firewall through namespace or IP range.

1. To limit traffic from your Azure Relay namespace:
  - a. Add the namespace <\*.servicebus.windows.net> or go to **System > Administration > Smart Integration Connector** and click **Relay** at the top.
  - b. The Relay Name is the namespace of the Azure Relay. Add this namespace to your firewall rules to restrict traffic from this Azure Relay. The namespace should have a format similar to this example: **<somehost>.servicebus.windows.net**.

### OneStream Windows Client Application

General (Relay)	
Relay Name	nssmartic.servicebus.windows.net
IPv4 Whitelist	

### OneStream Local Gateway Configuration

Namespace	nssmartic.servicebus.windows.net
Remote Gateway Host	localhost
Bound Port at Gateway	20433

**NOTE:** The namespace will be different for your development and production environments.

2. (Optional) Additionally, you can limit traffic further from an IP address by following these Azure-specific instructions:
  - a. Whitelist all IP addresses returned by this [script](#).
  - b. Review these IP addresses periodically as Microsoft may change them. Published changes are found on the [Microsoft Community Hub](#).

# Use Smart Integration Connector

You can use Smart Integration Connector to access data from your Local Gateway Connection Data Source or through Direct Connections.

## Examples

### Data Adapters Example

1. Go to **Application > Presentation > Dashboards > Workspaces > [choose Workspace] > [choose Maintenance Unit] > Data Adapters**.
2. Create or select an existing data adapter.
3. Verify that the **Database Location** is **External** and the **External Database Connection** is the custom connection that you defined earlier.
4. Enter a valid SQL Query.

General (Data Adapter)	
Name	Northeast Sales Data Adapter
Description	
Maintenance Unit	Firt Maintenance Unit
Data Source	
Command Type	SQL
Database Location	External
External Database Connection	Northeast Sales
SQL Query	select * from [Sales_2022].[dbo].[NE_Sales]
Results Table Name	

5. Test  the data adapter and view the results.

### SQL Table Editor Example

The following use case describes how to send a query after establishing a connection.



## Use Smart Integration Connector

---

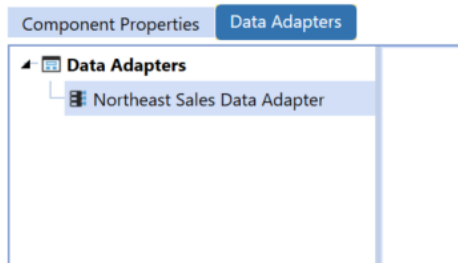
1. Go to **Application > Presentation > Dashboards > Workspaces > [choose Workspace] > [choose Maintenance Unit] > [choose Maintenance Unit] > Components > SQL Table Editor**.
2. Create or open a SQL Table Editor.
3. Verify the following:
  - **Database Location** is **External**,
  - **External Database Connection** is the custom connection that you defined earlier,
  - **Table Name** is defined as the table you want to return data from.

SQL Table Editor	
Database Location	External
External Database Connection	NE Sales Data
Schema Name	
Table Name	NorthEast_Sales

4. Open the associated dashboard and run the query. The OneStream Smart Integration Connector will connect to the external database. If it connects correctly, the query will populate.

## Grid View Example

1. Go to **Application > Presentation > Dashboards > Workspaces > [choose Workspace] > [choose Maintenance Unit] > [choose Maintenance Unit] > Components > Grid View**.
2. Create or open a grid view.
3. Configure the grid to use the data adapter.



4. Run the associated dashboard to see the data.

## Perform a Drill Back

The following snippet describes how to load data from a local gateway connection data source and how to perform a drill back. The example below has been updated from the Standard SQL Connectors business rule. If you do not have the Snippet Editor with the OneStream Application, you can find the Snippet Editor on the MarketPlace.

1. Download the Snippet Editor from the MarketPlace.
2. Navigate to **Application > Tools > Business Rules**.
3. Open **Connector**.
4. Navigate to **Snippets > SQL Connector > Standard SQL Connectors**.
5. Copy the Sample Business Rule.
6. Edit the query information. *Enter `dim ConnectionStringGateway As String = Your Connection information`.*

**NOTE:** This example assumes that you have completed the setup and installation process and configured a custom database connection in the System Configuration as a Gateway type. Refer to [Define Database Location in OneStream](#) for more information.

```
'Get the query information (prior to using the gateway)
Dim connectionString As String = GetConnectionString(si, globals, api)
'Get the query information (using the gateway)
Dim connectionString_gateway As String = GetConnectionString_Gateway(si, global3, api)|
```

7. Enter the connection name. In this example, “Northeast Sales” is the Gateway Connection Name as defined in the application configuration.

```
'Create a Connection string to the External Database (prior to using the gateway)
Private Function GetConnectionString(ByVal si As Sessioninfo, ByVal globals As BRGlobals,
ByVal api As Transformer) As String
    Try
        'Named External Connection
        '-----
        Return "Revenue Mgmt System"
    Catch ex As Exception
        Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
    End Try
End Function

'Create a Connection string to the External Database (using the Gateway)
Private Function GetConnectionStringGateway(ByVal si As Sessioninfo, ByVal globais As
BRGlobals, ByVal api As Transformer) As String
    Try
        'Named External Connection - Gateway
        '-----
        Return "Northeast Sales"
    Catch ex As Exception
        Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
    End Try
End Function
```

8. Enter the drill back information to your database.

```
If args.DrillCode.Equals(StageConstants.TransformationGeneral.DrillCodeDefaultValue,
StringComparison.InvariantCultureIgnoreCase) Then
    'Source GL Drill Down
    drillTypes.Add(New DrillBackTypeInfo(ConnectorDrillBackDisplayTypes.FileShareFile, New
NameAndDesc("InvoiceDocument", "Invoice Document")))
    drillTypes.Add(New DrillBackTypeInfo(ConnectorDrillBackDisplayTypes.DataGrid, New
NameAndDesc("MaterialTypeDetail", "Material Type Detail")))
    drillTypes.Add(New DrillBackTypeInfo(ConnectorDrillBackDisplayTypes.DataGrid, New
NameAndDesc("MaterialTypeDetail_Gateway", "Material Type Detail (Smart Integration)")))
```

9. Edit the level of drill back information returned.


**Example:** This example shows previously existing code that leverages a VPN based SQL connection and the Gateway based method shown in the second "Else If" block.

```
Else If args.DrillBackType.NameAndDescription.Name.Equals("MaterialTypeDetail",
StringComparison.InvariantCultureIgnoreCase) Then
    'Level 1: Return Drill Back Detail
    Dim drillBackSQL As String = GetDrillBackSQL_L1(si, globais, api, args)
    Dim drillBackInfo As New DrillBackResultInfo
    drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.OaDataGrid
    drillBackInfo.DataTable = api.Parser.GetXFDataTableForSQLQuery(si, DbProviderType.SqlServer,
connectionstring, True, drillBackSQL, False, args.PageSize, args.PageNumber)
    Return drillBackInfo

Else If args.DrillBackType.NameAndDescription.Name.Equals("MaterialTypeDetail_Gateway",
StringComparison.InvariantCultureIgnoreCase) Then
    'Level 1: Return Drill Back Detail
    Dim drillBackSQL As String = GetDrillBackSQL_L1(si, globais, api, args)
    Dim drillBackInfo As New DrillBackResultInfo
    drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.OaDataGrid
    drillBackInfo.DataTable = api.Parser.GetXFDataTableForSQLQuery(si, DbProviderType.Gateway,
connectionstring_gateway, True, drillBackSQL, False, args.PageSize, args.PageNumber)
    Return drillBackInfo
```

## Perform a Write Back

You can perform a write back using Smart Integration Connector leveraging the defined credentials to the local gateway datasource at the Smart Integration Connector Gateway. If the credentials have permission to insert, update, and/or delete records in a remote datasource, a OneStream business rule could be leveraged to write-back, update, and/or delete data as needed to support a financial process.

 **Example:** The following example shows how to insert rows and columns to a Smart Integration Connector remote database.

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports System.Windows.Forms
Imports Microsoft.VisualBasic
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Wcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine
```

```
Namespace OneStream.BusinessRule.Extender.SIC_BulkCopyExample
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args As ExtenderArgs) As Object
            Try
                ' SIC Gateway name
                Dim sicGatewayName As String = "jl-db-achqa1-gateway"

                ' SIC remote rule
                Dim sicRemoteRule As String = "SIC_Functions"

                ' SIC remote rule function
                Dim sicRemoteRuleFunction As String = "RunOperation"

                ' Create and populate DataTable
                Dim dt As New DataTable()
                dt.Columns.Add("Scenario", GetType(String))
                dt.Columns.Add("Time", GetType(String))
                dt.Columns.Add("Entity", GetType(String))
                dt.Columns.Add("Account", GetType(String))
                dt.Columns.Add("Amount", GetType(Double))
                dt.Rows.Add("Actual", "2023M3", "Houston Heights", "Net Sales", 100.25)
                dt.Rows.Add("Actual", "2023M3", "South Houston", "Net Sales", 1230.66)

                ' Compress data table before passing into remote business rule
                Dim dtCompress As CompressionResult = CompressionHelper.CompressJsonObject
(Of DataTable)(si, dt, XFCompressionAlgorithm.DeflateStream)

                Dim dtObj(2) As Object ' Create object to store arguments for remote business rule
                dtObj(0) = dtCompress ' compressed datatable
                dtObj(1) = "SIC_WriteBack" ' remote database table name
                dtObj(2) = "RevenueMgmt" ' remote data source name

                ' Execute remote business rule to bulk copy to target table
                Dim bulkRemoteResults As RemoteRequestResultDto
                =BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, sicRemoteRule,
                dtObj, sicGatewayName,sicRemoteRuleFunction,String.Empty, False, 600)

                ' Get result status
                If bulkRemoteResults.RemoteResultStatus <>
RemoteMessageResultType.RunOperationReturnObject Then ' Check if successful
                    ' Failed, do something
                    BRApi.ErrorLog.LogMessage(si,"Failed with status:" & bulkRemoteResults.
                    RemoteResultStatus.ToString)

                End If

                ' Get returned message
                Dim returnedMsg As String = CompressionHelper.InflateJsonObject(Of String)
                (si,bulkRemoteResults.resultDataCompressed)

                BRApi.ErrorLog.LogMessage(si,returnedMsg)

                Return Nothing
            Catch ex As Exception
```

## Use Smart Integration Connector

---

```
        Throw ErrorHandler.LogWrite(si, New XFXException(si, ex))
    End Try
End Function
End Class
End Namespace
```

The Extensibility Rule above calls the following Smart Integration Function:

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports System.Data.SqlClient
Imports OneStream.Shared.Common
Imports OneStreamGatewayService

Namespace OneStream.BusinessRule.SmartIntegrationFunction.SIC_Functions
    Public Class MainClass

        ' Function to bulk copy a compressed data table to a SQL database table
        ' Pass in compressed data table, database table name and data source name
        Public Shared Function RunOperation(dtCompress As CompressionResult,tablename As String,
            datasource As String) As String

            -----
            ' -----
            ' Get SQL connection string
            Dim connString As String = APILibrary.GetRemoteDataSourceConnection(datasource)

            ' Inflate compressed datatable
            Dim dt As DataTable = CompressionHelper.InflateJsonObject(Of DataTable)
            (New SessionInfo,dtCompress)

            If dt IsNot Nothing AndAlso dt.Rows.Count > 0 Then
                ' Check data table has been created and is populated

                ' Create sql connection to DWH
                Using sqlTargetConn As SqlConnection = New SqlConnection(connString)

                    sqlTargetConn.Open ' Open connection

                    Using bulkCopy = New SqlBulkCopy(sqlTargetConn)

                        bulkCopy.DestinationTableName = tableName ' DWH table
                        bulkCopy.BatchSize = 5000
                        bulkCopy.BulkCopyTimeout = 30

                        bulkCopy.WriteToServer(dt) ' Bulk copy data table to database table
                    End Using
                End Using
            End If
        End Function
    End Class
End Namespace
```

```
        End Using
    End Using

    Else
        Throw New Exception("Problem uncompressing data in SIC gateway")
    End If

    Return $"{dt.Rows.Count} rows bulk inserted into table {tableName}"

End Function

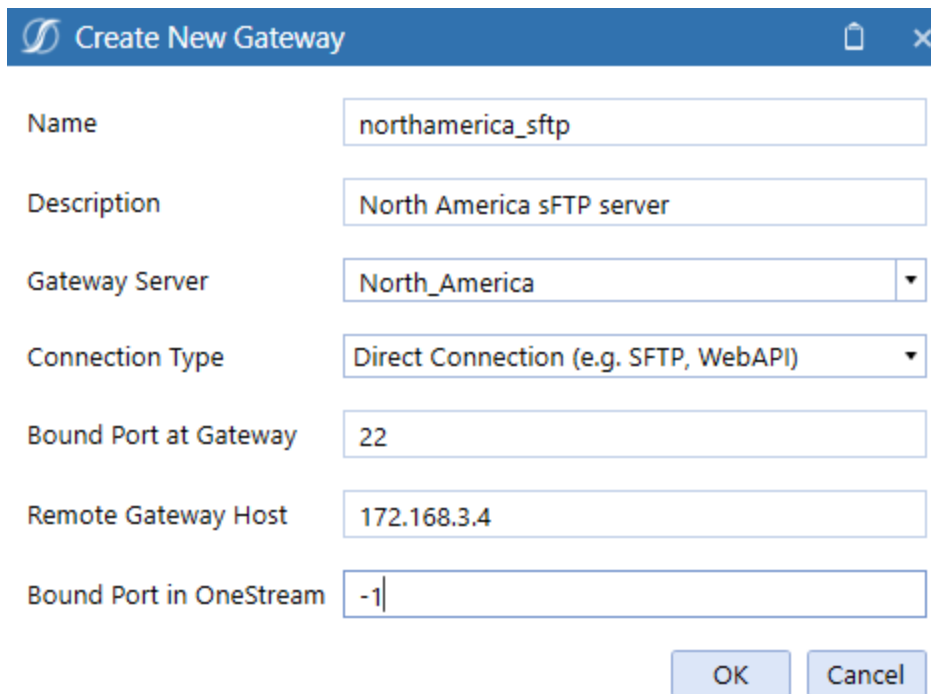
End Class
End Namespace
```

## Support for sFTP

Smart Integration Connector provides support for connecting to sFTP servers to send and retrieve files. Perform the steps in the following sections to establish a connection and then send and retrieve files.

**NOTE:** You must have an sFTP server available on a port. The port must be allowed for inbound and outbound connections on the Local Gateway Server. For this example, we have used port 22.

1. Login to OneStream.
2. Navigate to **System > Administration > Smart Integration Connector**.
3. Create a New **Gateway** and fill out all of the corresponding details for your Gateway and the Gateway Server.
4. From **Connection Type**, select Direct Connection (e.g., SFTP, WebAPI).
5. For **Bound Port at Gateway**, enter 22.
6. For **Remote Gateway Host**, enter the IP address or resolvable host name of the machine where your SFTP server is located.



**Create New Gateway**

Name: northamerica\_sftp

Description: North America sFTP server

Gateway Server: North\_America

Connection Type: Direct Connection (e.g. SFTP, WebAPI)

Bound Port at Gateway: 22

Remote Gateway Host: 172.168.3.4

Bound Port in OneStream: -1

OK Cancel

7. For **Bound Port in OneStream**, enter -1 to automatically assign an unused port number. You can also specify your own port number by entering a value greater than 1024 and less than 65535. It is recommended to use a higher value because it is less likely that number will be in use as this port number must be globally unique across all applications hosted on the OneStream servers.
8. Click **OK**.
9. Copy the **Gateway** to the **OneStream Smart Integration Connector Local Gateway Server Configuration**.
10. Save the Local Gateway Server configuration and restart the Smart Integration Connector Gateway service.

**Example:** Here is an example of how you can upload and download files through an SFTP extensibility rule.

**NOTE:** You will need to add WinSCPnet.DLL to your business rule Referenced Assemblies from the Properties tab in the business rule.



```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports System.Windows.Forms
Imports Microsoft.VisualBasic
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Wcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine
Imports WinSCP

Namespace OneStream.BusinessRule.Extender.SFTP_Example
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args As ExtenderArgs) As Object

            Try

                ' Setup the objects to read Gateway Details from BRAPIS
                Dim objGatewayDetails As GatewayDetails = BRApi.Utilities.GetGatewayConnectionInfo(si,
"WinSCP_Gateway")
                Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "SFTP_Password", Nothing, "rochester_
gateway",String.Empty,"SFTP_Password", False, 600)

                ' Setup session options
                Dim sessionOptions As New SessionOptions
                With sessionOptions
                    .Protocol = Protocol.Sftp
                    .HostName = "localhost" 'HostName in this instance is in reference to
OneStream and will always be localhost.
                    .UserName = "onestreamtest" 'sFTP server UserName
                    .Password = "*****" 'sFTP server Password
                    .Password = objRemoteRequestResultDto.ObjectResult ' This is the returned value
from the remote rule that obtains the customer controlled password
                    .PortNumber = objGatewayDetails.OneStreamPortNumber
                    'use BRAPI to populate Port Number and return the dynamically assigned value from OneStream
                    .SshHostKeyFingerprint = "*****" 'SSH Host Key from sFTP
                    host
                End With

                Using session As New Session
                    ' Connect
                    session.Open(sessionOptions)

                    ' Get the filepath
                    ' BatchHarvest in this example is File Share / Applicaitons / GolfStream / Batch /
Harvest
```

```
        Dim fileUPPath As String = BRAPi.Utilities.GetFileShareFolder(si,
FileShareFolderTypes.BatchHarvest, Nothing)
        Dim fileDNPath As String = BRAPi.Utilities.GetFileShareFolder(si,
FileShareFolderTypes.BatchHarvest, Nothing)

        ' Upload or download files
        Dim transferOptions As New TransferOptions
        transferOptions.TransferMode = TransferMode.Binary

        Dim transferResult As TransferOperationResult
        ' Upload
        fileUPPath = fileUPPath & "\SFTP_TEST_UPLOAD.txt"
        transferResult = session.PutFiles(fileUPPath, "/", False, transferOptions)

        'Throw on any error
        transferResult.Check()

        ' Download
        fileDNpath = fileDNPath & "\SFTP_TEST_DOWNLOAD.txt"
        transferResult = session.GetFiles("\SFTP_TEST_DOWNLOAD.txt", fileDNpath, False,
transferOptions)

        'Throw on any error
        transferResult.Check()

    End Using

    Return Nothing
Catch ex As Exception
    Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
Return Nothing
End Try

End Function
End Class
End Namespace
```

## Transferring Files from Local FileShare

You can use a Data Management job to move files Smart Integration Connector from a local FileShare. To do this, you build an extender business rule and call it through a data management job. This extender business rule will call a Smart Integration Function (remote function) and obtain the results.

## Step 1 - Setup the Remote Server / Remote Share

To get started, setup the Smart Integration Function:

1. Navigate to **Application > Tools > Business Rules**.
2. Open the **Smart Integration Function** folder.
3. Create a new business rule (for example, TestFileRead) .
4. Copy and paste the following business rule code snippet.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;

namespace OneStream.BusinessRule.SmartIntegrationFunction.TestFileRead
{
    public class MainClass
    {
        public byte[] RunOperation(string year)
        {
            string fname = @"c:\temp\hw_" + year + ".csv";
            byte[] buffer = System.IO.File.ReadAllBytes(fname);
            return buffer;
        }

        public byte[] GetOtherFileData(string year)
        {
            string fname = @"c:\temp\zw_" + year + ".csv";
            byte[] buffer = System.IO.File.ReadAllBytes(fname);
            return buffer;
        }

        public bool DeleteOldData(string year)
        {
            string fname = @"c:\temp\zw_" + year + ".csv";
            try
            {
                System.IO.File.Delete(fname);
                return true;
            }
            catch (IOException)
            {
                return false;
            }
        }
    }
}
```

# Step 2 - Pull file from Extender Business Rule

1. Navigate to **Application > Tools > Business Rules**.
2. Open the **Extensibility Rules** folder.
3. Create a new business rule (for example, ProcessRemoteFileData) .
4. Copy and paste the following business rule code snippet.

```
Imports System
Imports System.Data
Imports System.Data.Common
Imports System.IO
Imports System.Collections.Generic
Imports System.Globalization
Imports System.Linq
Imports Microsoft.VisualBasic
Imports System.Windows.Forms
Imports OneStream.Shared.Common
Imports OneStream.Shared.Wcf
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Database
Imports OneStream.Stage.Engine
Imports OneStream.Stage.Database
Imports OneStream.Finance.Engine
Imports OneStream.Finance.Database

Namespace OneStream.BusinessRule.Extender.ProcessRemoteFileData
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args As ExtenderArgs) As Object
            Try
                Dim stepNumber As String = "1"

                If (Not args.NameValuePairs Is Nothing) Then
                    ' Extracting the value from the parameters collection
                    If (args.NameValuePairs.Keys.Contains("step")) Then
                        stepNumber = args.NameValuePairs.Item("step")
                    End If
                    BRApi.ErrorLog.LogMessage(si, "File Processing Step: " & stepNumber)
                End If

                Select Case stepNumber

                    Case Is = "1"
                        GetData(si)
                        Return Nothing

                    Case Is = "2"
                        CleanupData(si)
                End Select
            Catch ex As Exception
                BRApi.ErrorLog.LogMessage(si, "Error in ProcessRemoteFileData: " & ex.Message)
            End Try
        End Function
    End Class
End Namespace
```

```
Return Nothing

End Select

Catch ex As Exception
    Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
End Try

Return Nothing
End Function

Public Sub CleanupData(ByVal si As SessionInfo)

    Dim argTest(0) As Object
    argTest(0) = "2023"

    ' Here we are telling it to specifically call
    Dim objRemoteRequestResultDto As RemoteRequestResultDto =
    BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestFileRead", argTest, "gateway-jasonl-
    smartic", "DeleteOldData")
    If (objRemoteRequestResultDto.RemoteResultStatus =
    RemoteMessageResultType.RunOperationReturnObject) Then

        ' The delete method returns a true/false return type
        Dim result As Boolean
        ' ObjectResultValue introduced in v7.4 to simplify obtaining the return
        value from a method that doesn't return a
        ' Dataset/Datatable
        result = objRemoteRequestResultDto.ObjectResultValue

        Dim objRemoteRequestResultDtoCached As RemoteRequestResultDto =
        BRApi.Utilities.ExecRemoteGatewayCachedBusinessRule(si, "TestFileReadCache", argTest,
        "gateway-jasonl-smartic", String.Empty)

        BRApi.ErrorLog.LogMessage(si, "File Deleted: " & result)
    Else
        If (Not (objRemoteRequestResultDto.remoteException Is Nothing)) Then
            Throw ErrorHandler.LogWrite(si, New XFException(si,
            objRemoteRequestResultDto.remoteException))
        End If
    End If

End Sub

Public Sub GetData(ByVal si As SessionInfo)

    ' Demonstrating how to pass parameters
    ' We create an object array that matches the number of parameters
    ' To the remote function. In this case, we have 1 parameter that is a string
    Dim argTest(0) As Object
    argTest(0) = "2023"
```

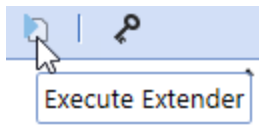
```
' This is where you can allow caching of the remote function. We are passing
in true at the end to force the cache to be updated
' We are also allowing the function to run for 90 seconds.
' String.empty means this will look for a remote function/method called
"RunOperation"
    Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestFileRead", argTest,
"ryantestconnection2", String.Empty, "TestFileRead", True, 90)
    If (objRemoteRequestResultDto.RemoteResultStatus =
RemoteMessageResultType.RunOperationReturnObject) Then
        Dim bytesFromFile As Byte()
        bytesFromFile = objRemoteRequestResultDto.ObjectResultValue
        Dim valueAsString As String = System.Text.Encoding.UTF8.GetString
(bytesFromFile)
        Return valueAsString
        bytesFromFile = Convert.FromBase64String
(objRemoteRequestResultDto.ObjectResultValue)
        'bytesFromFile = objRemoteRequestResultDto.ObjectResultValue

        Dim valueAsString As String = System.Text.Encoding.UTF8.GetString
(bytesFromFile)

        ' Do something with the files here....
        BRApi.ErrorLog.LogMessage(si, "File Contents: " & Left(valueAsString,10))
        ' We are saving the file into the OneStream Share here
        ' This is an option to allow other OneStream functions to process the data
        'Dim groupFolderPath As String =
FileShareFolderHelper.GetGroupsFolderForApp(si, True, AppServerConfig.GetSettings
(si).FileShareRootFolder, si.AppToken.AppName)
        Dim groupFolderPath As String = BRApi.Utilities.GetFileShareFolder(si,
FileShareFolderTypes.BatchHarvest, Nothing)
        Using sw As StreamWriter = New StreamWriter(groupFolderPath &
"\outputfile.csv")
            sw.Write(valueAsString)
            sw.Close()
        End Using
    Else
        If (Not (objRemoteRequestResultDto.remoteException Is Nothing)) Then
            Throw ErrorHandler.LogWrite(si, New XFException(si,
objRemoteRequestResultDto.remoteException))
        End If
    End If
End Sub

End Class
End Namespace
```

5. Test your Extender Business Rule via the Execute Extender button in the toolbar.



## Step 3 - Automate from Data Management / Task Scheduler

After the Extensibility Rule has been created and tested you can automate from a Data Management Job and associate Task Schedule. See [Task Scheduler](#) for more information.

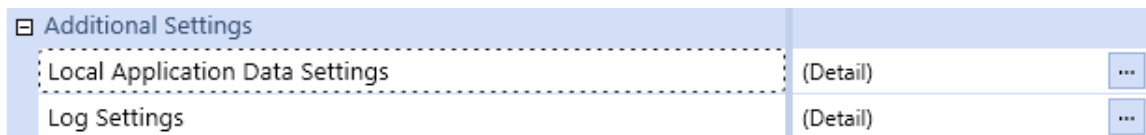
1. Navigate to **Application > Tools > Data Management**.
2. Create a new Data Management Group.
3. Enter the business rule.
4. Set the first Parameter to step=1.
5. Set the Parameters to step=2.
6. Create associated Task Schedule to run the Data Management job.

## Support for DLL Migration

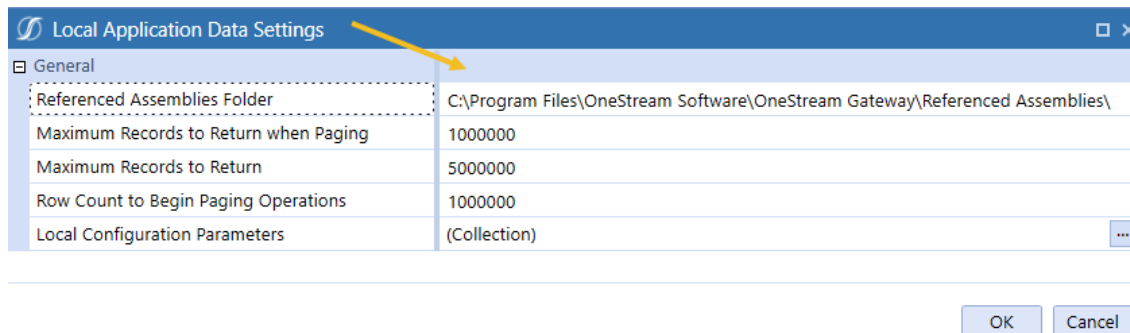
For OneStream Platform version 8.0 and above, all customer-supplied DLLs will be referenced through Smart Integration Connector. To use a DLL, copy the DLLs to the **Referenced Assemblies Folder** in the Local Gateway Server Utility and reference this DLL within your Smart Integration Function. See [Referenced Assemblies Folder](#).

To verify the Referenced Assemblies Folder path:

1. Open the **OneStream Local Gateway Configuration** and Run as Administrator.
2. Navigate to and open **Local Application Data Settings**.



3. The file path under **Referenced Assemblies Folder** opens to the default location.



4. Click the **OK** button.

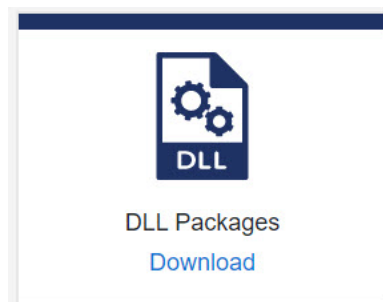
See the following SAP example for this process in use. See [Smart Integration Connector Settings](#) for more information on these fields.

## Support for ERPConnect (SAP)

As an alternative to creating a Local Gateway Connection to your SAP database, you can connect to SAP using third-party DLLs, such as ERPConnect###.dll. ERPConnect###.dll can be referenced using a Smart Integration Connector Remote business rule. Although ERPConnect45.dll can no longer enable a connection to SAP systems starting with Platform version 8.0, a newer version ERPConnectStandard20.dll is available through the download DLL Packages from the Platform page of the [Solution Exchange](#). ERPConnect requires additional libraries to be obtained from SAP as well, which can reside in the same reference assembly folder as ERPConnect.

To get started:

1. From the Platform page of the [Solution Exchange](#), download the DLL Packages, which contains the ERPConnectStandard20.dll file.





## Use Smart Integration Connector

---

2. Copy the ERPConnectStandard20.dll to your Referenced Assemblies Folder.
3. Install the required [Visual C++ 2013 Runtime](#).
4. From SAP, download and copy SAP NetWeaver RFC Library DLL (sapnwrfc.dll) and associated icudt50.dll, icuin50.dll, icuuc50.dll to your Referenced Assemblies Folder. See [Theobald Software ERPConnect Requirements](#) for additional information.
5. Modify your business rules to use the ERPConnectStandard20.dll.
6. Navigate to **Application > Tools > Business Rules**.
7. Expand the Smart Integration Function list.
8. Create a new **Smart Integration Function** or select an existing one.
9. Click the **Properties** tab.

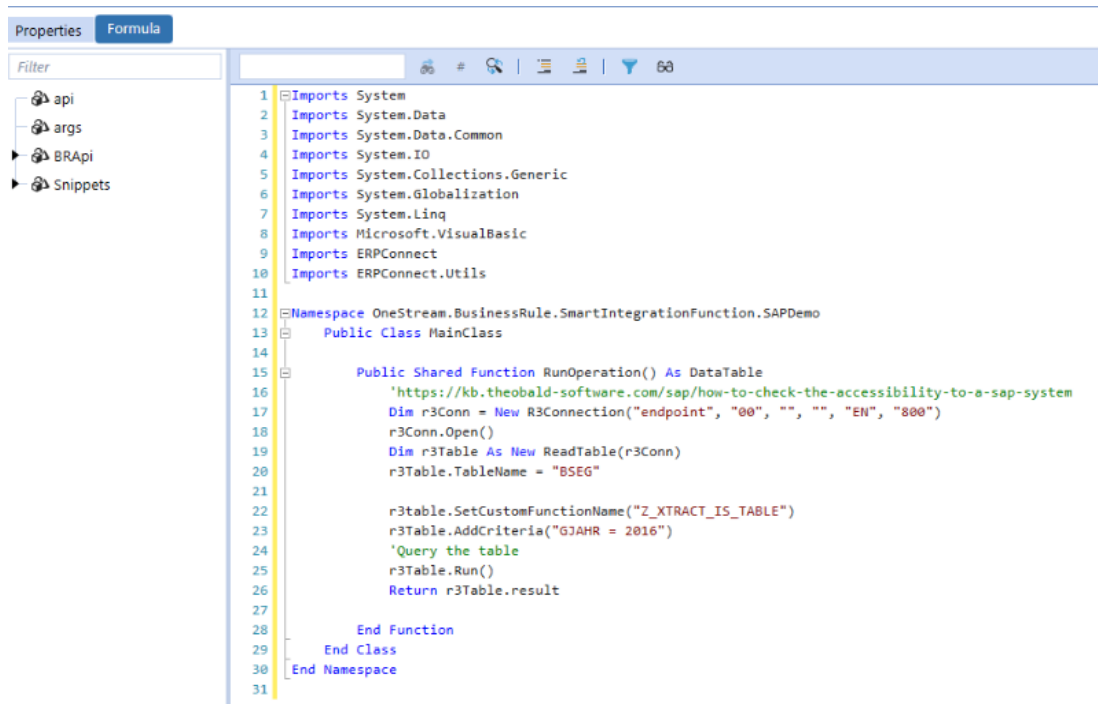
The screenshot shows the 'Properties' tab of a dialog box for a Smart Integration Function. The 'General' section is expanded, showing fields for Name, Type, Language Type, Referenced Assemblies, and Is Encrypted. The 'Security' section is also expanded, showing fields for Access Group and Maintenance Group. The 'Referenced Assemblies' field is highlighted with a dashed border.

Properties	
Formula	
General	
Name	SIC_SAP_TEST
Type	Smart Integration Function
Language Type	Visual Basic
Referenced Assemblies	ERPConnectStandard20.dll
Is Encrypted	False
Security	
Access Group	Administrators
Maintenance Group	Administrators

10. Enter **ERPConnectStandard20.dll** in the Referenced Assemblies field. The Smart Integration Connector Gateway server will attempt to locate this DLL in the previously defined folder: **Referenced BusinessRule AssemblyFolder**.
11. Add Imports for **ERPConnect** and **ERPConnect.Utils**.

## Use Smart Integration Connector

---



12. Verify you can compile the function on your Gateway.

You are now ready to add your custom code.

# Business Rules

The Smart Integration Connector Capabilities introduce additional business rule APIs (BR APIs) to allow for execution and management of remote business rules inside the context of the Smart Integration Connector gateway. These rules are transported using https to the Smart Integration Connector local gateway, compiled locally, executed, and the results returned to the caller for further processing. They provide a mechanism for complex drill backs, data processing scenarios or to invoke remote webAPIs hosted in your network.

**NOTE:** Gateways must have a local data source defined to invoke remote business rules.

There are two ways business rules can be used with the Smart Integration Connector Gateway:

- OneStream BR APIs interact with a specific local gateway and run on OneStream application servers.
- Business rules that reference DLLs that are only accessible by the Local Gateway Server. These BRs are compiled and executed on the local gateway (Remote Business Rules when creating them in the Windows Desktop Client).

In these scenarios, the local gateway must have the allowRemoteCodeExec setting configured to True to enable remote execution.

The BR APIs are outlined below:

<a href="#">ExecRemoteGatewayRequest</a>
<a href="#">ExecRemoteGatewayCachedBusinessRule</a>
<a href="#">ExecRemoteGatewayJob</a>
<a href="#">ExecRemoteGatewayBusinessRule</a>
<a href="#">GetRemoteDataSourceConnection</a>
<a href="#">GetRemoteGatewayJobStatus</a>

<a href="#">GetSmartIntegrationConfigValue</a>
<a href="#">GetGatewayConnectionInfo</a>
<a href="#">Incompatible Business Rules</a>

## ExecRemoteGatewayRequest

Initiates a request to a local gateway as specified in the remote request object. This request is dispatched to the Smart Integration Connector local gateway connection data source with the specified command remote invoked.

**NOTE:** This method is used for request and response type interactions to a remote endpoint that runs for three or less minutes. The default execution timeout is 90 seconds and can be overridden by setting the CommandTimeout property on the RemoteRequestDTO instance provided.

Parameter details:

- RemoteRequestDTO: Remote request object populated with the remote command and endpoint
- Returns: RemoteRequestResultDto - Result of execution including the status and any exceptions which may have occurred on the remote endpoint

Following is an example connector business rule that would run on the OneStream application server sending a remote request and block of code to a Local Gateway Connection:

```
Dim objxfRemoteRequestResultDto As RemoteRequestResultDto
Dim objxfRemoteRequest As New RemoteCodeRequestDto
' Indication the desire is to run a remote block of code
objxfRemoteRequest.connectionType = RemoteCommandType.RemoteCodeExec
' Name of the remote host to pass to
objxfRemoteRequest.gatewayHostforRequest = "testconnection"
Dim strCode As String
strCode = "using System;..." ' Valid block of C# or VB.NET code
objxfRemoteRequest.LanguageType = RemoteCodeLanguageType.CSHARP
objxfRemoteRequest.remoteCodeBlock = strCode
objxfRemoteRequestResultDto=BRApi.Utilities.ExecRemoteGatewayRequest(objxfRemoteRequest)
Dim xfDT = New XFDataTable(si,objxfRemoteRequestResultDto.resultSet,Nothing,1000)
```

This BR API can also be used to invoke arbitrary SQL commands against a Smart Integration Connector local gateway connection data source at your site:

```
Dim drillBackInfo As New DrillBackResultInfo
Dim drillBackSQL As String = "SELECT CustName, InvDesc, BomCode, UnitPrice, Units, Amount,
'BomDetail' As DrillTypeCode FROM InvoiceMaterialDetail WHERE
(InvYear = 2022) And (InvMonth = 'M3') And (PlantCode = 'H200') And (InvNo = 'I1-H200-AH2347-
2022M3')
And (ProdModel = 'P-Boy') And (DestinationCode = '1230') And (CustID = 'AH2347')ORDER BY BomCode"
Dim objxfRemoteRequestResultDto As RemoteRequestResultDto
Dim objxfRemoteRequest As New RemoteRequestDto
' Indicate this is a remote SQL command request
objxfRemoteRequest.connectionType = RemoteCommandType.SQLCommand
objxfRemoteRequest.GatewayRemotedBConnection = "RevenueMgmt" ' Name of the connection defined in the
remote endpoint
objxfRemoteRequest.gatewayHostforRequest = "testconnection" ' Name of the remote host to pass to
objxfRemoteRequest.RemoteCommand = drillBackSQL
objxfRemoteRequestResultDto=BRApi.Utilities.ExecRemoteGatewayRequest(objxfRemoteRequest)
Evaluate the results to determine if it was successful
If (objxfRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Success) Then
    Dim xFDT = New XFDataTable(si,objxfRemoteRequestResultDto.ResultSet,Nothing,1000)
    drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid
    drillBackInfo.DataTable = xFDT
    Return drillBackInfo
Else
    drillBackInfo.TextMessage = "Not Successful"
    drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.TextMessage
    Return drillBackInfo
End If
```

Remote function returning a datatable (VB.NET) without parameters:

```
'Here we are telling it to specifically call a remote Smart Integration Function
called
'GetDataFromDB at SIC Gateway called TestConnection with a method called RunOperation
Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "GetDataFromDB", Nothing, " TestConnection",
"RunOperation")
If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Success) Then
    If (objRemoteRequestResultDto.ResultType = RemoteResultType.DataTable) Then
        BRApi.ErrorLog.LogMessage(si, "Data Returned: " &
objRemoteRequestResultDto.ResultSet.Rows.Count)
    End If

    Else
        If (Not (objRemoteRequestResultDto.remoteException Is Nothing)) Then
            Throw ErrorHandler.LogWrite(si, New XFException(si,
objRemoteRequestResultDto.remoteException))
        End If
    End If
```

# ExecRemoteGatewayCachedBusinessRule

When a cache flag and key is provided to the ExecRemoteGatewayBusinessRule BR API, this method is used to invoke a previously cached method. This is intended to be used for high-frequency remote business rules to avoid the performance impact of recompiling a remote method on each invocation.

**NOTE:** Requires allowRemoteCodeExec = True on Smart Integration Connector local gateway. If the previously cached method is not invoked after 60 minutes, the remote cached method is purged.

Parameter details:

- *si*: SessionInfo object used to create connection objects
- *cachedFunctionKey*: Key of previously cached remote function to invoke
- *functionArguments*: Array of objects aligning to function / method parameters. Null / Nothing if there are none required
- *remoteHost*: Name of remote host to invoke operation. (Smart Integration Connector Local Gateway Name)
- *executionTimeOut*: Timeout (in seconds) on the remote job
- *Returns*: RemoteRequestResultDto - Result of execution including the status and any exceptions which may have occurred on the remote endpoint

Here is the rule in C#:

```
try
{
    // Caches a SIC BR called GetDataFromDB on SIC Gateway called TestConnection
    // and caches the function with the name GetDataFromDB with a cache key of GetDataFromDBCached
    RemoteRequestResultDto objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayBusinessRule
    (si, "GetDataFromDB", null, "TestConnection", "RunOperation", " GetDataFromDBCached ", false, 90);

    if (objRemoteRequestResultDto.RemoteResultStatus == RemoteMessageResultType.Success
        && objRemoteRequestResultDto.ResultSet != null
        && objRemoteRequestResultDto.ResultType == RemoteResultType.DataTable)
    {
        BRApi.ErrorLog.LogMessage(si, "Data Returned - Rows:" +
        objRemoteRequestResultDto.ResultSet.Rows.Count);
    }
    else
    {

```

```
        if (objRemoteRequestResultDto.RemoteException != null)
        {
            throw ErrorHandler.LogWrite(si, new XFException(si, objRemoteRequestResultDto.RemoteException));
        }
        else
        {
            BRApi.ErrorLog.LogMessage(si, "Remote Smart Integration Function Succeeded - no data/datatable returned");
        }
    }

    // Subsequent invocations of the remote BR can be run by specifying the endpoint and the cached key name
    RemoteRequestResultDto objRemoteRequestResultDtoCached =
    BRApi.Utilities.ExecRemoteGatewayCachedBusinessRule(si, " GetDataFromDBCached", null, "
    TestConnection ", 90);

    return null;
}
catch (Exception ex)
{
    throw ErrorHandler.LogWrite(si, new XFException(si, ex));
}
```

Here is the rule in VB.NET:

```
Dim argTest(0) As Object
argTest(0) = "2023"

'Here we are telling it to specifically call a remote Smart Integration function called
'TestFileread at a remote gateway called TestConnection and caching the compiled
'result as a key called TestFileReadCache with a 90 second timeout
Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestFileRead", argTest, " TestConnection ",
"DeleteOldData", "TestFileReadCache", false, 90)
If (objRemoteRequestResultDto.RemoteResultStatus =
RemoteMessageResultType.RunOperationReturnObject) Then

    'The delete method returns a true/false return type
    Dim result As Boolean
    ' ObjectResultValue introduced in v7.4 to simplify obtaining the return value from a method that
    ' doesn't return a
    ' Dataset/Datatable
    result = objRemoteRequestResultDto.ObjectResultValue

    ' Here we are invoking a compiled/cached method at a remote gateway called
    ' TestConnection using the key of TestFileReadCache with a 90 second timeout.
    Dim objRemoteRequestResultDtoCached As RemoteRequestResultDto =
    BRApi.Utilities.ExecRemoteGatewayCachedBusinessRule(si, "TestFileReadCache", argTest, "
    TestConnection ", 90)

    BRApi.ErrorLog.LogMessage(si, "File Deleted: " & result)
```

```
Else
    If (Not (objRemoteRequestResultDto.remoteException Is Nothing)) Then
        Throw ErrorHandler.LogWrite(si, New XFException(si,
objRemoteRequestResultDto.remoteException))
    End If
End If
```

## ExecRemoteGatewayJob

There may be instances where a remote operation on the Smart Integration Connector Local Gateway host would need to process and assemble data that may take several minutes to run. In this situation, you could use this BR API to queue and run a remote business rule in an asynchronous manner where the remote Smart Integration Connector Local Gateway host returns a Job ID (GUID) that can later be used to obtain the job's status or the results if the job is complete. When invoking this method, if the RemoteMessageResultStatus is returned as JobRunning (as shown in the example below), the RequestJobID is populated with the ID of the queued job that can later be used to obtain status.

**NOTE:** Requires allowRemoteCodeExec = True on Smart Integration Connector Local Gateway. There is a defined default limit of 30 minutes for remote jobs to execute before the job is cancelled, and an overloaded version of ExecremoteGatewayJob exists allowing the timeout to be provided, but can never exceed 4 hours. This is not configurable and if this timeout is reached, the status returned shows the timeout. If the result is not obtained within five minutes after the job completes (using the GetRemoteGatewayJobStatus BR API), the remote results are purged to ensure that result objects reclaim server memory on the Smart Integration Service host.

Here is a basic overview of invoking a remote job and displaying the returned remote Job ID in VB.NET.

**NOTE:** This is required to call back into GetRemoteJobStatus with the returned ID to obtain the result:

```
Dim drillBackInfo As New DrillBackResultInfo

Dim argInt As New XFSimpleObject(12)
argInt.Int32Value = 12
Dim argTest(1) As Object
argTest(0) = 100
argTest(1) = "test"

' Invoking a OneStream Business Rule as a remote job
```



```
Dim objRemoteRequestResultDto As RemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayJob(si,
"TestLongRunning", argTest, "ryantestconnection",String.Empty)
If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.JobRunning) Then
drillbackinfo.TextMessage = "Done " & objRemoteRequestResultDto.RequestJobID.ToString()
drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.TextMessage
Return drillBackInfo
End If
```

Here is the rule in VB.NET to invoke a job, obtain the job ID, and 'poll' until completion:

```
Try
Dim jobID As Guid
' Invoking a long-running Job with a Smart Integration Function called GetDataFromDB on SIC Gateway
called testConneciton
Dim objRemoteRequestResultDto As RemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayJob
(si, "GetDataFromDB", Nothing, "TestConnection",String.Empty)

' If Successful, the status is returned indicating the job is running with the job ID - Use this
ID to interrogate if the job is compleed.
If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.JobRunning)
Then
    jobID = objRemoteRequestResultDto.RequestJobID
    BRApi.ErrorLog.LogMessage(si, "Remote Job Queued and Running - JobID: " & jobID.ToString())

    ' Example waiting 20 seconds for job to complete
    For loopControl = 0 To 10
        System.Threading.Thread.Sleep(2000)
        Dim objJobStatus As RemoteJobStatusResultDto = BRApi.Utilities.GetRemoteGatewayJobStatus(si,
        JobID, " TestConnection ")

    If (objJobStatus.RemoteJobState = RemoteJobState.Running)
        BRApi.ErrorLog.LogMessage(si, "Remote Job Still running - JobID: " & jobID.ToString())
    Else If (objJobStatus.RemoteJobState = RemoteJobState.Completed)

        ' Checking the return type from the remote job
        If (Not objJobStatus.RemoteJobResult.ResultSet Is Nothing) Then
            Dim xfDT = New XFDataTable(si,objJobStatus.RemoteJobResult.ResultSet,Nothing,1000)
            BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Datatable Returned - JobID: " &
            jobID.ToString())
            Return Nothing
        Else If (Not objJobStatus.RemoteJobResult.ResultDataSet Is Nothing) Then
            Dim xfDT = New XFDataTable(si,objJobStatus.RemoteJobResult.ResultDataSet.Tables
            (0),Nothing,1000)
            BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Dataset Returned - JobID: " &
            jobID.ToString())
            Return Nothing
        Else If (Not objJobStatus.RemoteJobResult.ObjectResultValue Is Nothing) Then
            BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Object Returned - JobID:
            " & jobID.ToString())
            Return Nothing
        End If
    Else If (objJobStatus.RemoteJobState = RemoteJobState.JobNotFound)
        BRApi.ErrorLog.LogMessage(si, "Remote Job Not Found - JobID: " & jobID.ToString())
```

```
Return Nothing
Else If (objJobStatus.RemoteJobState = RemoteJobState.RequestTimeOut)
    BRApi.ErrorLog.LogMessage(si, "Remote Job Timed Out - JobID: " & jobID.ToString())
Return Nothing
Else If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Exception)
    BRApi.ErrorLog.LogMessage(si, "Exception During Execution of Job: " &
objRemoteRequestResultDto.RemoteException.ToString())
End If
Next

Else
    ' Exception occurred immediately during compile/initial run
    If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Exception)
        BRApi.ErrorLog.LogMessage(si, "Exception Executing Job: " &
objRemoteRequestResultDto.RemoteException.ToString())
    Else
        BRApi.ErrorLog.LogMessage(si, "General Job Execution Error - State: " &
objRemoteRequestResultDto.RemoteResultStatus.ToString())
    End If
End If

Return Nothing
Catch ex As Exception
Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
End Try
```

Here is the rule in C# to invoke a job, obtain the job ID, and 'poll' until completion:

```
try
{
    Guid jobID;
    RemoteRequestResultDto objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayJob(si,
"GetDataFromDB", null, " TestConnection ",string.Empty);
    if (objRemoteRequestResultDto.RemoteResultStatus == RemoteMessageResultType.JobRunning)
    {
        jobID = objRemoteRequestResultDto.RequestJobID;
        BRApi.ErrorLog.LogMessage(si, "Remote Job Queued and Running - JobID: " + jobID.ToString());

        // Example waiting 20 seconds for job to complete
        for (int loopControl=0; loopControl<=10; loopControl++)
        {
            System.Threading.Thread.Sleep(2000);
            RemoteJobStatusResultDto objJobStatus = BRApi.Utilities.GetRemoteGatewayJobStatus(si,
jobID, " TestConnection");

            if (objJobStatus.RemoteJobState == RemoteJobState.Running)
            {
                BRApi.ErrorLog.LogMessage(si, "Remote Job Still running - JobID: " + jobID.ToString());
            }
            else if (objJobStatus.RemoteJobState == RemoteJobState.Completed)
            {
                // Checking the return type from the remote job
                if (objJobStatus.RemoteJobResult.ResultSet != null)
```

```
        {
            XFDataTable xfDT = new XFDataTable(si,objJobStatus.RemoteJobResult.ResultSet,null,1000);
            BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Datatable Returned - JobID: " +
jobID.ToString());
            return null;
        }
        else if (objJobStatus.RemoteJobResult.ResultDataSet != null)
        {
            XFDataTable xfDT = new XFDataTable
(si,objJobStatus.RemoteJobResult.ResultDataSet.Tables[0],null,1000);
            BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Dataset Returned - JobID:
" + jobID.ToString());
            return null;
        }
        else if (objJobStatus.RemoteJobResult.ObjectResultValue !=null)
        {
            BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Object Returned -
JobID: " + jobID.ToString());
            return null;
        }
        else if (objJobStatus.RemoteJobState == RemoteJobState.JobNotFound)
        {
            BRApi.ErrorLog.LogMessage(si, "Remote Job Not Found - JobID: " +
jobID.ToString());
            return null;
        }
        else if (objJobStatus.RemoteJobState == RemoteJobState.RequestTimeout)
        {
            BRApi.ErrorLog.LogMessage(si, "Remote Job Timed Out - JobID: " + jobID.ToString());
            return null;
        }
        else if (objRemoteRequestResultDto.RemoteResultStatus ==
RemoteMessageResultType.Exception)
        {
            BRApi.ErrorLog.LogMessage(si, "Exception During Exeuction of Job: "+
objRemoteRequestResultDto.RemoteException.ToString());
            return null;
        }
    }
}
// End for loop
}
else
{
    // Exception occurring immediately during compile/initial run
    if (objRemoteRequestResultDto.RemoteResultStatus == RemoteMessageResultType.Exception)
        BRApi.ErrorLog.LogMessage(si, "Exception Executing Job: " +
objRemoteRequestResultDto.RemoteException.ToString());
    else
        BRApi.ErrorLog.LogMessage(si, "General Job Execution Error - State: " +
objRemoteRequestResultDto.RemoteResultStatus.ToString());

    return null;
}
}
catch (Exception ex)
```

```
    {  
        throw ErrorHandler.LogWrite(si, new XFException(si, ex));  
    }  
    return null;
```

## ExecRemoteGatewayBusinessRule

This is a core BR API that can be used to remotely invoke Smart Integration functions on a specified remote Smart Integration Connector Local Gateway host. The Smart Integration Connector Local Gateway must have allowRemoteCodeExec set to True for this BR API to invoke an operation successfully, otherwise the Smart Integration Connector Local Gateway host returns a result indicating that remote code execution is disabled.

This method takes a previously authored Smart Integration function, written in VB.NET or C#, in the OneStream application and passes it to the remote host for execution. With this BR API, it is expected that remote calls should take no more than 2-3 minutes to return a result to the caller as this BR API will block until a result is returned. If longer running or sync operations are needed, consider using the execRemoteGatewayJob BR API.

**NOTE:** Requires allowRemoteCodeExec = True on Smart Integration Service

Parameter details:

- *si*: SessionInfo object used to create connection objects
- *brName*: Name of the locally defined (within the OneStream Application scope) Smart Integration function
- *functionArguments*: Array of objects aligning to function / method parameters. Null / Nothing if there are none required.
- *remoteHost*: Name of remote host to invoke operation. (Smart Integration Connector name)
- *functionName*: Name of the function in the Smart Integration function to invoke. If null or empty, a function/method with the name RunOperation is expected to exist within the authored code.
- (Optional) *cachedFunctionKey*: Name used to cache the remote function to avoid recompiling the function on a subsequent call. This is optional and if missing or null the function will not be cached.

- (Optional) *forceCacheUpdate*: Option indicating if a previously cached function should be replaced with this version. When true, and an existing function is found with a name specified in the *cachedFunctionKey* parameter, the BR is recompiled and recached. This is useful for situations where a remote function is cached and a change was made.
- *executionTimeout*: Timeout (in seconds) on the remote job (In 7.4, this is now an optional parameter and defaults to 90 seconds if the parameter is missing.)

Here is a C# drill-back example:

```
DrillBackResultInfo drillBackInfo = new DrillBackResultInfo();
DataTable dtf = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "SAP_Test", null, "gateway-SAP_
Europe", string.Empty).ResultSet;
var xfdt = new XFDDataTable(si, dtf, null, 1000);
drillBackInfo.DataTable = xfdt;
drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid; return drillBackInfo;
```

Here is a VB.NET drill-back example that invokes a remote business rule accepting 2 parameters:

```
Dim drillBackInfo As New DrillBackResultInfo
Dim argTest(1) As Object ' Creating an object array to package the method parameters
argTest(0) = 12 ' First parameter is an integer
argTest(1) = "test" ' Second parameter is a string

'Remote Smart Integration Function Signature: ' Public Shared Function RunOperation2(testval As
Integer, teststr As String) As ArrayList

'Invoking method RunOperation2 on endpoint testConnection passing in user defined parameters as an
array

Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule
(si, "TestValueTypeParam", argTest, "testConnection", "RunOperation2")

If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.RunOperationReturnObject)
Then
    Dim returnVal As System.Collections.ArrayList returnVal =
objRemoteRequestResultDto.ObjectResultValue

    'Simple demonstration without error checking to look at the first element of the arraylist
drillbackinfo.TextMessage =
"Completed! " & returnVal(0).ToString() drillBackInfo.DisplayType =
ConnectorDrillBackDisplayTypes.TextMessage
Return drillBackInfo
Else If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Success)
'Demonstrating a 'pattern' whereby the caller can verify what the type is that's returned and
handle properly.
Dim xfdt = New XFDDataTable(si,objRemoteRequestResultDto.resultSet,Nothing,1000)
drillBackInfo.DataTable = xfdt
drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid
Return drillBackInfo
```

## Business Rules

---

```
Else If (Not (objRemoteRequestResultDto.remoteException Is Nothing)) Then
Throw ErrorHandler.LogWrite(si, New XFException(si,objRemoteRequestResultDto.remoteException))
End If
Return Nothing
End If
```

Below is a TestFileRead Remote Business Rule function in C# Referenced by Examples Below:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Data.SqlClient;
using OneStreamGatewayService;

namespace OneStream.BusinessRule.SmartIntegrationFunction.GetDataFromDB
{
    public class MainClass
    {
        public DataTable RunOperation()
        {
            DataTable dataTableResults = new DataTable();
            string connectionString, sql;

            // The API Below is only available in 7.4 and allows the ability to
            // Obtain a remotely defined connection string.
            connectionString = APILibrary.GetRemoteDataSourceConnection("RevenueMgmt");

            SqlConnection conn;
            conn = new SqlConnection(connectionString).Open();
            sql = "Select * FROM InvoiceMaterialDetail";
            SqlCommand cmd = new SqlCommand(sql, conn);
            var dbreader = cmd.ExecuteReader();
            dataTableResults.Load(dbreader);
            return dataTableResults;
        }
    }
}
```

Remote function returning a scalar value (VB.NET) or object with parameters:

```
Dim argTest(0) As Object
argTest(0) = "2023"

'Here we are telling it to specifically call a remote Smart Integration Function called
TestFileRead at SIC Gateway
```

```
'called TestConnection with a method called DeleteOldData
Dim objRemoteRequestResultDto As
RemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestFileRead",
argTest, "TestConnection", "DeleteOldData")
If (objRemoteRequestResultDto.RemoteResultStatus =
RemoteMessageResultType.RunOperationReturnObject) Then
    'The delete method returns a true/false return type

    Dim result As Boolean
    'ObjectResultValue introduced in v7.4 to simplify obtaining the return
    'value from a method that doesn't return a Dataset/Datatable
    result = objRemoteRequestResultDto.ObjectResultValue

    'Previous to v7.4, The result was returned in a compressed format and it is required that
    'InflateJsonObject was invoked
    'result = CompressionHelper.InflateJsonObject(Of Object)
    '(si,objRemoteRequestResultDto.resultDataCompressed)

    BRApi.ErrorLog.LogMessage(si, "File Deleted: " & result)
Else
    If (Not (objRemoteRequestResultDto.remoteException Is Nothing)) Then
        Throw ErrorHandler.LogWrite(si, New XFException
            (si, objRemoteRequestResultDto.remoteException))
    End If
End If
End if
```

Remote function returning a scalar type/object (C#):

```
try
{
    object[] argTest = new object[1];
    argTest[0] = "2023";

    'Here we are telling it to specifically call a remote Smart Integration Function called
    TestFileRead at SIC Gateway
    'called TestConnection with a method called DeleteOldData

    RemoteRequestResultDto objRemoteRequestResultDto =
    BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestFileRead", argTest, "TestConnection",
    "DeleteOldData");

    if (objRemoteRequestResultDto.RemoteResultStatus ==
    RemoteMessageResultType.RunOperationReturnObject
    && objRemoteRequestResultDto.ObjectResultValue != null)

        bool result;
        if (bool.TryParse(objRemoteRequestResultDto.ObjectResultValue.ToString(), out result))

            BRApi.ErrorLog.LogMessage(si, "File Deleted: " + result.ToString());
        else
            BRApi.ErrorLog.LogMessage(si, "Returned a non-boolean value");
    else
        if (objRemoteRequestResultDto.RemoteException != null)
```

```
        throw ErrorHandler.LogWrite(si, new XFException(si,
objRemoteRequestResultDto.RemoteException));

    return null;
    catch (Exception ex)
        throw ErrorHandler.LogWrite(si, new XFException(si, ex))
    End Try
```

Remote function returning a datatable (C#):

```
try
{
    // Here we are telling it to specifically call a remote Smart Integration Function called
    GetDataFromDB at SIC Gateway called TestConnection with a method called RunOperation
    RemoteRequestResultDto objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayBusinessRule
    (si, "GetDataFromDB", null, "TestConnection", "RunOperation");

    if (objRemoteRequestResultDto.RemoteResultStatus == RemoteMessageResultType.Success
        && objRemoteRequestResultDto.ResultSet != null
        && objRemoteRequestResultDto.ResultType == RemoteResultType.DataTable)
    {

        BRApi.ErrorLog.LogMessage(si, "Data Returned - Rows:" +
objRemoteRequestResultDto.ResultSet.Rows.Count);
    }
    else
    {
        if (objRemoteRequestResultDto.RemoteException != null)
        {
            throw ErrorHandler.LogWrite(si, new XFException(si,
objRemoteRequestResultDto.RemoteException));
        }
        else
        {
            BRApi.ErrorLog.LogMessage(si, "Remote Smart Integration Function Succeeded - no
data/datatable returned");
        }
    }

    return null;
}
catch (Exception ex)
{
    throw ErrorHandler.LogWrite(si, new XFException(si, ex));
}
```

## GetRemoteDataSourceConnection

This remote business rule will return the connection string associated with a Local Gateway Configuration Data Source.



**NOTE:** Requires allowRemoteCodeExec = True on Smart Integration Local Gateway.

Parameter details:

- *Data Source*: The name of the Local Gateway Configuration Data Source.

Here is the rule in VB.NET :

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Namespace OneStream.BusinessRule.SmartIntegrationFunction.GetRemoteDataSource_VB
    Public Class MainClass
        Public Shared Function RunOperation() As DataTable
            Dim dataTableResults As New DataTable

            ' Get the remotely defined connection String
            Dim connectionString As String =
OneStreamGatewayService.APILibrary.GetRemoteDataSourceConnection("Sales_Data1");

            Dim conn As SqlConnection = New SqlConnection(connectionString)
            ' Insert custom code

            Return dataTableResults
        End Function
    End Class
End Namespace
```

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Data.SqlClient;

namespace OneStream.BusinessRule.SmartIntegrationFunction.GetRemoteDataSourceSample
{
    public class MainClass
    {
        public DataTable RunOperation()
        {
            DataTable dataTableResults = new DataTable();
        }
    }
}
```

```
// Get the remotely defined connection string
string connectionString =
OneStreamGatewayService.APILibrary.GetRemoteDataSourceConnection("Sales_Data1");

SqlConnection conn = new SqlConnection(connectionString);
// Insert custom code

return dataTableResults;
    }
}
}
```

## GetRemoteGatewayJobStatus

This BR API returns the status or the results of a previously remotely queued job invoked against a specified Smart Integration Connector Local Gateway host.

**NOTE:** Requires allowRemoteCodeExec = true on Smart Integration Service.

Parameter details:

- *si*: SessionInfo object used to create connection objects
- *JobID*: GUID of remote job ID returned upon successful call to ExecRemoteGatewayJob
- *remoteHost*: Name of remote host to invoke operation (Smart Integration Connector Name)

The sample below invokes a job as part of a data management job inside a OneStream extenderrule. The example demonstrates a simple Smart Integration Function that sleeps 2 seconds 1000 times in a loop simulating a long running task. The corresponding extender rule illustrates how this long running function can be invoked as a job, returning a job ID and subsequently polled until it's completed.

Here is the 'long running' Smart Integration Function in VB.NET:

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports System.Threading
```

```
Namespace OneStream.BusinessRule.SmartIntegrationFunction.LongRunningTest
    Public Class MainClass
        Public Shared Function RunOperation() As DataTable
            For i As Integer = 1 To 1000
                thread.Sleep (2000)
            Next

            Dim result As String
            result = OneStreamGatewayService.APILibrary.GetSmartIntegrationConfigValue("test")

            Dim Table1 As DataTable
            Table1 = New DataTable("TableName")

            Dim column1 As DataColumn = New DataColumn("SettingName")
            column1.DataType = System.Type.GetType("System.String")

            Table1.Columns.Add(column1)

            Table1.Rows.Add(result)
            Return Table1

        End Function
    End Class
End Namespace
```

It would be typical to invoke long running jobs as part of a Data management/Extender Rule and the code below is an example on how this could be accomplished in VB.NET:

```
Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Object, ByVal
args As ExtenderArgs) As Object
    Try
        Dim jobID As Guid
        Dim objRemoteRequestResultDto As RemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayJob
        (si, "LongRunningTest", Nothing, "testConnection",String.Empty)

        If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.JobRunning) Then
            jobID = objRemoteRequestResultDto.RequestJobID BRApi.ErrorLog.LogMessage
            (si, "Remote Job Queued and Running - JobID: " & jobID.ToString())
            'Example waiting 20 seconds for job to complete
            For loopControl = 0 To 10
                System.Threading.Thread.Sleep(2000)

            Dim objJobStatus As RemoteJobStatusResultDto = BRApi.Utilities.GetRemoteGatewayJobStatus(si, JobID,
            "testconnection2")
            If (objJobStatus.RemoteJobState = RemoteJobState.Running)
                BRApi.ErrorLog.LogMessage(si, "Remote Job Still running - JobID: " & jobID.ToString())
            Else If (objJobStatus.RemoteJobState = RemoteJobState.Completed)
                ' Checking the return type from the remote job
                If (Not objJobStatus.RemoteJobResult.ResultSet Is Nothing) Then
                    Dim xfDT = New XFDataTable(si,objJobStatus.RemoteJobResult.ResultSet,Nothing,1000)
                    BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Datatable Returned - JobID: " & jobID.ToString
                    ())
```

```
Return Nothing
Else If (Not objJobStatus.RemoteJobResult.ResultDataSet Is Nothing)
Then Dim xfDT = New XFDataTable(si,objJobStatus.RemoteJobResult.ResultDataSet.Tables
(0),Nothing,1000)
BRApi.LogError.LogMessage(si, "Remote Job Completed - Dataset Returned - JobID: " & jobID.ToString
()) Return Nothing
Else If (Not objJobStatus.RemoteJobResult.ObjectResultValue Is Nothing) Then
BRApi.LogError.LogMessage
(si, "Remote Job Completed - Object Returned - JobID: " & jobID.ToString())
Return Nothing
End If
Else If (objJobStatus.RemoteJobState = RemoteJobState.JobNotFound) BRApi.LogError.LogMessage
(si, "Remote Job Not Found - JobID: " & jobID.ToString()) Return Nothing
Else If (objJobStatus.RemoteJobState = RemoteJobState.RequestTimeout) BRApi.LogError.LogMessage
(si, "Remote Job Timed Out - JobID: " & jobID.ToString()) Return Nothing
Else If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Exception)
BRApi.LogError.LogMessage(si, "Exception During Execution of Job: " &
objRemoteRequestResultDto.RemoteException.ToString())
End If
Next
Else ' Exception occurring immediately during compile/initial run
If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Exception)
BRApi.LogError.LogMessage(si, "Exception Executing Job: " &
objRemoteRequestResultDto.RemoteException.ToString())
Else
BRApi.LogError.LogMessage(si, "General Job Execution Error - State: " &
objRemoteRequestResultDto.RemoteResultStatus.ToString())
End If
End If
Return Nothing
Catch ex As Exception
Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
End Try
End Function
```

## GetSmartIntegrationConfigValue

This BR API allows access to the Local Gateway Local Application Data Settings. Accessing the remotely stored secret or customer-defined configuration values is done using a new "Remote" equivalent of the BR API namespace. This feature can be used to:

- Reference configuration parameters in a remote business rule running on a Smart Integration Connector Local Gateway Server
- Store credentials to network resources allowing the developer of remote business rules to reference values stored in the configuration file instead of having them hard-coded and viewable by anyone with permission to edit a business rule.

These configuration values are defined and edited using the Smart Integration Connector Local Gateway Configuration Utility. The API used to obtain these values is demonstrated in the full business rule example below:

**NOTE:** Requires allowRemoteCodeExec = True on Smart Integration Local Gateway.

Here is the rule in C#:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Data.SqlClient;

namespace OneStream.BusinessRule.SmartIntegrationFunction.GetRemoteDataSourceSample
{
    public class MainClass
    {
        public DataTable RunOperation()
        {
            DataTable dataTableResults = new DataTable();

            // Get the remotely defined connection string
            string connectionString =
                OneStreamGatewayService.APILibrary.GetRemoteDataSourceConnection("Sales_Data1");

            SqlConnection conn = new SqlConnection(connectionString);
            // Insert custom code

            return dataTableResults;
        }
    }
}
```

Here is another example in VB.NET:

```
Imports System
Imports System.Data
Imports System.Data.Common
Imports System.IO
Imports System.Collections.Generic
Imports System.Globalization
Imports System.Linq
Imports Microsoft.VisualBasic
```

```
Imports OneStream.Shared.Wcf
Imports OneStreamGatewayService

Namespace OneStream.BusinessRule.SmartIntegrationFunction.SecretTester

Public Class MainClass
    Public Shared Function RunOperation() as bool
        Dim result As String
        ' APILibrary is the class containing new remote BRAPI methods
        ' GetSmartIntegrationConfigValue returns the string value of a found configuration
        ' element -- returns empty string if the specified key is not found
        result = APILibrary.GetSmartIntegrationConfigValue("test")
        return true
    End Function
End Class
End Namespace
```

## GetGatewayConnectionInfo

From a OneStream business rule, you can invoke this API to obtain gateway details such as:

- GatewayName: Name of the remote gateway
- GatewayVersion: Version of the Smart Integration Connector Gateway Service running on the remote host
- RemoteGatewayPortNumber: Bound Port at Gateway, the port of the remote service this direct connection is associated with.
- RemoteGatewayHost: Name of the remote host associated with the direct connection.
- OneStreamPortNumber: Bound Port in OneStream, the port number defined within OneStream that refers/maps to the specified direct connection.
- SmartIntegrationGatewayType: Type of the Smart Integration Connection (0=Database Connection, 1=Direct Connection)

This API is useful for direct connections where the port number is required before connecting to remote services such as sFTP or remote Web APIs because each endpoint defined in OneStream to Smart Integration Connector Local Gateways has a different port number and would need to be known by the business rule developer at design time. This API makes it easy to look up the remote port by knowing the name of the direct connection defined in OneStream. It returns other useful information outlined below:

Here is the rule in C#:

```
GatewayDetails gatewayDetailInformation = BRApi.Utilities.GetGatewayConnectionInfo(si,
"democonnection");
int oneStreamPortNumber = gatewayDetailInformation.OneStreamPortNumber;
```

```
Dim objGatewayDetails As GatewayDetails = BRApi.Utilities.GetGatewayConnectionInfo(si,
"democonnection")

sessionOpts.PortNumber = objGatewayDetails.OneStreamPortNumber
```

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports System.Windows.Forms
Imports Microsoft.VisualBasic
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Wcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine
Imports WinSCP

Namespace OneStream.BusinessRule.Extender.SFTP_Example
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args As ExtenderArgs) As Object

            Try

                ' Setup the objects to read Gateway Details from BRAPIs
                Dim objGatewayDetails As GatewayDetails = BRApi.Utilities.GetGatewayConnectionInfo(si,
"WinSCP_Gateway")
                Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule
                (si, "SFTP_Password", Nothing, "rochester_gateway", String.Empty, "SFTP_Password", False,
600)

                ' Setup session options
                Dim sessionOptions As New SessionOptions
                With sessionOptions
                    .Protocol = Protocol.Sftp
                    .HostName = "localhost"
```

```
'HostName in this instance is in refrence to OneStream and will always be localhost.
.UserName = "onestreamtest"
'sFTP server UserName
'.Password = "*****"
'sFTP server Password
.Password = CompressionHelper.InflateJsonObject(Of String)
(si,objRemoteRequestResultDto.resultDataCompressed)
'result of remote business rule to provide password from Local Gateway Server
'.PortNumber = 54321 'Bound Port in OneStream
.PortNumber = objGatewayDetails.OneStreamPortNumber 'use BRAPI to populate Port
Number
host
.SshHostKeyFingerprint = "*****" 'SSH Host Key from sFTP

End With

Using session As New Session
    ' Connect
    session.Open(sessionOptions)

    ' Get the filepath
    ' BatchHarvest in this example is File Share / Applicaitons / GolfStream / Batch /
Harvest
    Dim fileUPPath As String = BRAPi.Utilities.GetFileShareFolder(si,
FileShareFolderTypes.BatchHarvest, Nothing)
    Dim fileDNPath As String = BRAPi.Utilities.GetFileShareFolder(si,
FileShareFolderTypes.BatchHarvest, Nothing)

    ' Upload or download files
    Dim transferOptions As New TransferOptions
    transferOptions.TransferMode = TransferMode.Binary

    Dim transferResult As TransferOperationResult
    ' Upload
    fileUPpath = fileUPPath & "\SFTP_TEST_UPLOAD.txt"
    transferResult = session.PutFiles(fileUPpath, "/", False, transferOptions)

    'Throw on any error
    transferResult.Check()

    ' Download
    fileDNpath = fileDNPath & "\SFTP_TEST_DOWNLOAD.txt"
    transferResult = session.GetFiles("\SFTP_TEST_DOWNLOAD.txt", fileDNpath, False,
transferOptions)

    'Throw on any error
    transferResult.Check()

End Using

Return Nothing
Catch ex As Exception
    Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
Return Nothing
End Try

End Function
End Class
```



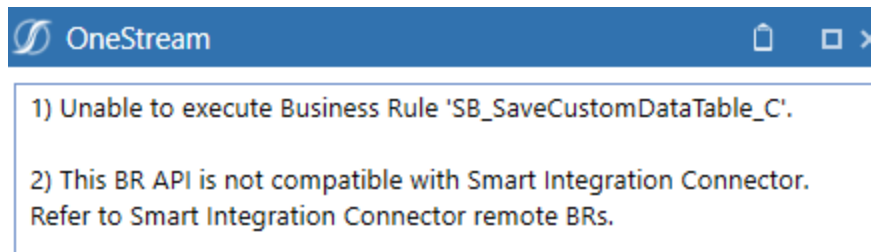
End Namespace

## Incompatible Business Rules

The following business rules are not compatible with Smart Integration Connector:

- CreateSAPConnection
- BRApi.Database.SaveCustomDataTable

If you attempt to use these business rules you will run into an error.



## Obtain Data through a WebAPI

In this scenario, you have a WebAPI (IPaaS integration or another accessible REST API) to obtain and pass back data to OneStream. You can use the following remote business rule in Smart Integration Connector to invoke the API. If you have results that are in JSON format, you can convert them to a data table and send them back to OneStream. If the data from the WebAPI is in JSON, you can process the data in Smart Integrator Connector. Additionally, you can send the raw data back as a string to a data management job for further testing.

Direct connections are preferred for this method and can be invoked using business rules within OneStream similar to the sFTP example provided above.

Use the following OneStream business rule to invoke the request.

```
Dim objRemoteRequestResultDto As RemoteRequestResultDto =  
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "RemoteWebAPISample", Nothing,  
"testconnection",String.Empty) If (objRemoteRequestResultDto.RemoteResultStatus =  
RemoteMessageResultType.Success) Dim xFDTable = New XFDataTable  
(si,objRemoteRequestResultDto.resultSet,Nothing,1000) End If
```

Use the following remote business rule to execute the request in C#

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using Newtonsoft.Json;
using System.Net.Http.Headers;

namespace OneStream.BusinessRule.SmartIntegrationFunction.RemoteWebAPISample
{
    public class MainClass
    {
        private static readonly HttpClient internalHttpClient = new HttpClient();

        static MainClass()
        {
            internalHttpClient.DefaultRequestHeaders.Accept.Clear();
            internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue
("application/json"));
            internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue
("application/x-www-form-urlencoded"));
            internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue
("application/octet-stream"));
            internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue
("text/plain"));
            internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue
("*/*"));
        }

        public DataTable RunOperation()
        {
            var stringTask = internalHttpClient.GetStringAsync
(https://localhost:44388/WeatherForecast);

            var msg = stringTask;
            DataTable dt = (DataTable)JsonConvert.DeserializeObject(stringTask.Result,
(typeof(DataTable)));

            return dt;
        }
    }
}
```

# Troubleshooting

This section provides help on addressing errors in Smart Integration Connector.

## Error Log

To view the error log, click **System > Logging > Error Log**.

Every five minutes, by default, the Smart Integration Connector tries to connect to an established Smart Integration Connector local gateway from each application server used in a deployment. If the gateway is unable to connect, it times out and adds an error to the error log. These errors are recorded in the OneStream error log along with other errors related to the OneStream application. You can configure the interval at which OneStream application servers monitor this gateway from 1 minute to 1440 minutes (1 day) to reduce the volume of logged failures for infrequently online test or validation environments.

**NOTE:** It is recommended to increase the time intervals for queries that run longer than five minutes. For example, if you have a query that runs ten minutes long, you need to set your time interval to above ten minutes (such as fifteen minutes). Time intervals can be adjusted from **System > Smart Integration Connector > Your connection > Gateway failures reporting interval (min)**.

Additional Settings	
Bound Port at Gateway	
Remote Gateway Host	localhost
Bound Port in OneStream	
Gateway failures reporting interval (min)	13

## Common Errors

## Memory Issues

If you receive any of the following errors, increase the memory in your Smart Integration Connector Local Gateway Server. For queries returning over 1 million records, 32 GB or more RAM is recommended.

- "Error while copying content to a stream. Received an unexpected EOF or 0 bytes from the transport stream."
- "An error occurred while sending the request. The response ended prematurely."

## Gateway Version is empty

If your gateway is reporting online, is of type "Database Connection" and the Version is empty, verify with your IT Admin that port 443 is fully open outbound between the SIC Local Gateway Server and the Azure Relay.

General (Gateway)	
Name	sql_testing_archqa1
Description	Testing SQL to ArchQA1 DB
Connection Type	Database Connection
Gateway Server Name	JL_VM_NEW
Web API Key	8675309
Gateway Key	8675309
Status	Online
Instance Count	1
Version	
Additional Settings	
Bound Port at Gateway	20433
Gateway failures reporting interval (min)	5

## Custom Data Source Names

You may not see the Data Source Names populate when setting up the custom connection with a new gateway. It is recommended to wait for five minutes from creating a new gateway to when you create the custom connection.

Data Source Name	
Timeouts	(Select One) Custom

## Array cannot be null Error

You receive the error: "Array cannot be null. (Parameter 'bytes') or "System.AggregateException - System.NullReferenceException: Object reference not set to instance of object"

**NOTE:** CompressionHelper.InflateJsonObject is now automatically executed as part of remote calls resulting in serialized .NET types returned from the Smart Integration Connector Gateway. Update any SIC related business rules accordingly.

Previously, it was required that a OneStream BR developer invoking a remote Smart Integration Function be aware of the data type returned and convert accordingly after the result is returned. An example where the returned result was a byte array involved code that appeared as follows:

### Example:

```
bytesFromFile = CompressionHelper.InflateJsonObject(Of System.Byte())
(si,objRemoteRequestResultDto.resultDataCompressed)
'The Smart Integration Connector Gateway now provides this type information back to OneStream
'and streamlines this conversion process using a newly added property called
'ObjectResultValue which is populated.
'When invoking the same operation shown above that previously required
'the type to be converted, a BR developer can do the following:
bytesFromFile = objRemoteRequestResultDto.ObjectResultValue
```

## Opening and Saving Configuration Errors

You may receive an error opening or saving your OneStream Local Gateway Configuration after installing Oracle Data Provider for .NET.

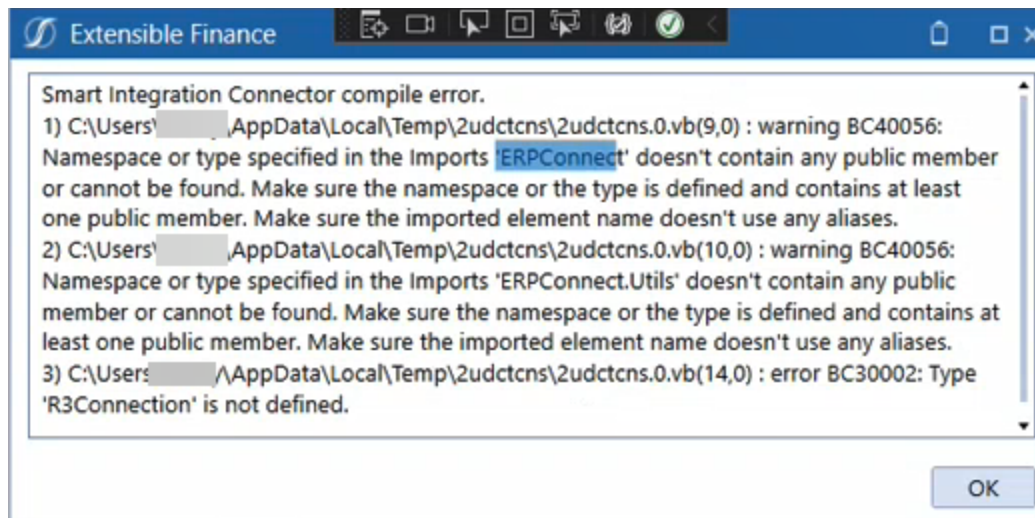
You must comment out the following line `<!--<add name="Oracle Data Provider for .NET" invariant="Oracle.DataAccess.Client" description=".Net Framework Data Provider for Oracle" type="Oracle.DataAccess.Client.OracleClientFactory, Oracle.DataAccess" />-->` when editing your OneStreamLocalGatewayConfiguration.exe.config to resolve this error.

Your configuration should look similar to this:

```
<DbProviderFactories>
  <add name="Npgsql Data Provider" invariant="Npgsql" description="Data Provider for
PostgreSQL" type="Npgsql.NpgsqlFactory, Npgsql" />
  <add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient" description=".Net Framework
Data Provider for MySQL" type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data" />
  <!--<add name="Oracle Data Provider for .NET" invariant="Oracle.DataAccess.Client"
description=".Net Framework Data Provider for Oracle"
type="Oracle.DataAccess.Client.OracleClientFactory, Oracle.DataAccess" />-->
```

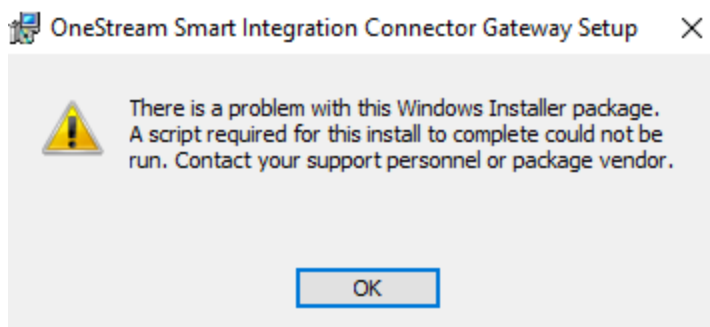
### Incorrect or Missing Library References

During compilation of remote business rules using .NET DLLs such as the ERPConnect Library to interface with SAP, incorrect or missing library references will result in an error similar (Smart Integration Connector compile error) to the image below.



### Script Error During Upgrade

During upgrades, you may run into the error "a script required for this install to complete could not be run." The action to resolve this error is to rerun the Smart Integration Connector installer. If you continue to see this error during upgrades, contact OneStream support.



# Smart Integration Functions with a DataTable/DataSet Parameter

When a Smart Integration Function is utilized for the purposes of performing write-backs or passing data from OneStream to a Smart Integration Connector Gateway environment having a parameter with a datatable or dataset data type, json serialization errors will be emitted. This is a known issue that is in active development. It's recommended to use alternative datatypes as an input parameter to these remote functions in the meantime.

```
Namespace OneStream.BusinessRule.SmartIntegrationFunction.DataTableTest
Public Class MainClass
    Public Shared Function RunOperation(ByVal dt As DataTable) As String
        Dim log As New System.Text.StringBuilder
        log.AppendLine(system.DateTime.Now.ToString & " successfully opened connection. Bulk
inserting " & dt.Rows.Count & " rows")

        For Each row As DataRow In dt.Rows
            log.AppendLine(row.ItemArray.ToString())
        Next

        Return log.ToString
    End Function
End Class
End Namespace
```

When this method is invoked with a datatable passed from a OneStream BR, an error as presented below will be emitted into the ErrorLog:

- *Description: 10/26/2023 3:54:48 PM - calling remote gateway BR.*
- *Exception = System.AggregateException: One or more errors occurred. (Serialization and deserialization of 'System.Type' instances are not supported. Path: \$.FunctionArguments.Columns.DataType.)*
- *---> System.NotSupportedException: Serialization and deserialization of 'System.Type' instances are not supported. Path: \$.FunctionArguments.Columns.DataType.*
- *---> System.NotSupportedException: Serialization and deserialization of 'System.Type' instances are not supported.*
- *at System.Text.Json.Serialization.Converters.UnsupportedTypeConverter`1.Write(Utf8JsonWriter writer, T value, JsonSerializerOptions options)*

- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.Metadata.JsonPropertyInfo`1.GetMemberAndWriteJson(Object obj, WriteStack& state, Utf8JsonWriter writer)*
- *at System.Text.Json.Serialization.Converters.ObjectDefaultConverter`1.OnTryWrite(Utf8JsonWriter writer, T value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWriteAsObject(Utf8JsonWriter writer, Object value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.Converters.IEnumerableConverter`1.OnWriteResume(Utf8JsonWriter writer, TCollection value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonCollectionConverter`2.OnTryWrite(Utf8JsonWriter writer, TCollection value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.Metadata.JsonPropertyInfo`1.GetMemberAndWriteJson(Object obj, WriteStack& state, Utf8JsonWriter writer)*
- *at System.Text.Json.Serialization.Converters.ObjectDefaultConverter`1.OnTryWrite(Utf8JsonWriter writer, T value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWriteAsObject(Utf8JsonWriter writer, Object value, JsonSerializerOptions options, WriteStack& state)*



- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.Converters.ArrayConverter`2.OnWriteResume(Utf8JsonWriter writer, TElement[] array, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonCollectionConverter`2.OnTryWrite(Utf8JsonWriter writer, TCollection value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWriteAsObject(Utf8JsonWriter writer, Object value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.Converters.ArrayConverter`2.OnWriteResume(Utf8JsonWriter writer, TElement[] array, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonCollectionConverter`2.OnTryWrite(Utf8JsonWriter writer, TCollection value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.Metadata.JsonTypeInfo`1.GetMemberAndWriteJson(Object obj, WriteStack& state, Utf8JsonWriter writer)*
- *at System.Text.Json.Serialization.Converters.ObjectDefaultConverter`1.OnTryWrite(Utf8JsonWriter writer, T value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.TryWrite(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*
- *at System.Text.Json.Serialization.JsonConverter`1.WriteCore(Utf8JsonWriter writer, T& value, JsonSerializerOptions options, WriteStack& state)*

## Data Returned as a String

Occasionally, data types can return as a string when you are expecting to see data in the original source format. Smart Integration Connector transfers data in Apache Parquet format from the Local Gateway Service to OneStream. If you are transferring a data type that is unsupported by parquet, the data converts and returns as string. You will need to add logic to re-convert the string to the desired and supported data type if needed.

In certain cases, if you receive the error "The method or operation is not implemented" then you can use a remote business rule to transfer data. This occurs when returning the varbinary(max) datatype.

## Null Columns are Not Returned

Columns that only contain NULL values are not returned as part of the query. The workaround is to use the function like NVL for Oracle or ISNULL for SQL Server to default data in the column before it is returned.

## Running Encrypted Smart Integration Connector Functions (Remote Business Rules)

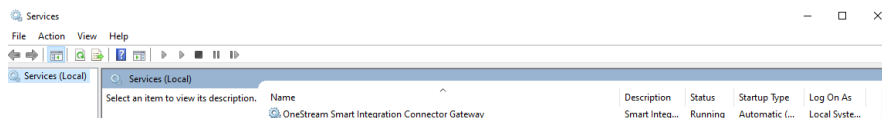
Encrypted Smart Integration Functions (Remote Business Rules) are not supported. This is in active development. Data will not be returned if you are trying to return data through an encrypted Smart Integration Function.

## Manual Start and Stop

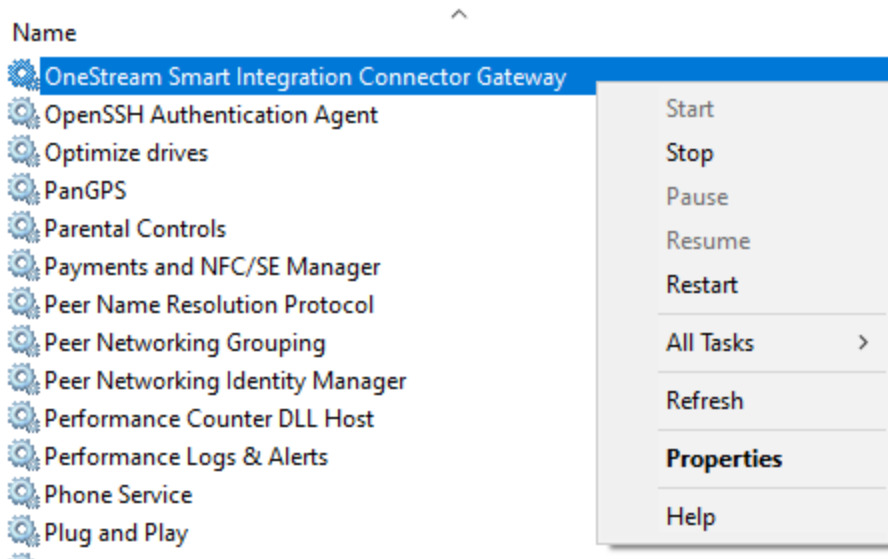
If you run into errors with the service, you may need to manually stop and restart the service. This can be accomplished in the GUI-based Services control manager as shown below or by using the command-line/PowerShell. The name of the service when using command line tools is "OneStreamSmartIntegration"

Using the Windows Service Control Manager:

1. Open **Services** from your Windows start menu.



2. Right-click on **OneStream Smart Integration Connector Gateway**.



3. Select **Stop**.
4. Right-click again and select **Start**.

Using an elevated command-prompt:

1. `net stop OneStreamSmartIntegration`
2. `net start OneStreamSmartIntegration`

Using an elevated PowerShell prompt:

1. `stop-service -ServiceName OneStreamSmartIntegration`
2. `start-service -ServiceName OneStreamSmartIntegration`

## Communication Error

If you see the following error in the Windows Service Log, it means that you have a mismatched WebAPIKey. This could occur if the WebAPI key is changed in OneStream and the configuration for the Smart Integration Local Gateway service is not exported from OneStream and re-imported into the Local Gateway Server service using the configuration utility.

*[14:13:36 INF] HTTP Request with invalid API key*

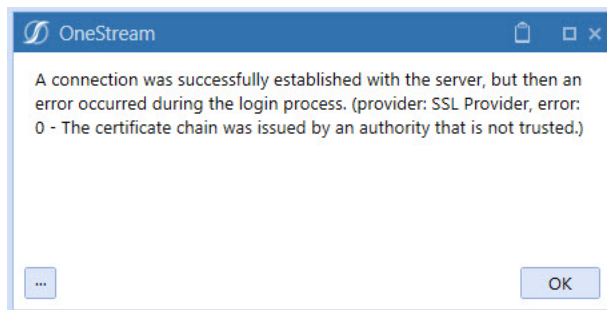
You can resolve this error by matching the WebAPIKey in the configuration utility.

**NOTE:** If the value is changed, you must restart the service.

### Trusted Certificate Chain

If you are using Smart Integration Functions and set the SQL Server connection string within the function, you may receive the following error:

A connection was successfully established with the server, but then an error occurred during the login process. (provider: SSL provider, error: 0 - The certificate chain was issued by an authority that is not trusted.)



To resolve this error, include **TrustServerCertificate=True**; to your connection string within the function.

### Gateway Unable to Connect

If your Gateway cannot connect, check your Smart Integration Connector error log for:

[2023-10-04 07:09:59 INF] Starting Listener for: <site name>.servicebus.windows.net

[2023-10-04 07:10:00 ERR] Unable to connect: Generic: Ip has been prevented to connect to the endpoint.

To resolve this issue, verify that the IP addresses in your Whitelisting to the Azure Relay is set up properly. See [Advanced Networking and Whitelisting](#).