Smart Integration Connector Guide

9.2.0 Release

Table of Contents

Revision History

> **IMPORTANT:** For best performance and reliability, upgrade Smart Integration Connector to match your OneStream version. While Smart Integration Connector Local Gateway Server supports the most recent previous version of OneStream, full functionality is not guaranteed.

Date	Release	Summary of Changes
09 December, 2025	9.2.0	Updated for release features, including the following enhancements: • Remote BR support for datasets that contain more than one datatable. • Support for SQL Table Editor write-backs to SQL Server without needing additional BRs. • Automatically create long running job processes using single BRAPI.

Date	Release	Summary of Changes
19 August, 2025	9.1.0	Updated for release features, including the following enhancements: • Removed WinSCP from the SIC Local Gateway Installer. Best practice is to use SSH.NET for SFTP transfers. See Support for SFTP. • Revised phrasing of "Database Connection" to "Gateway Connection" and adjusted related gateway terms to align with new user interface and tooltips. • Update Connection icons for Gateway Connections and Direct Connections. • Split "Use Smart Integration Connector" subheadings into separate sections for improved document navigation. • Move the following settings to Application Data Settings: ◦ "Allow Remote Code Execution" ◦ "Web API Bind Port"

Date	Release	Summary of Changes
29 April, 2025	9.0.0	Updated for release features, including the following enhancements: • Load Balanced Gateway Servers support queries over 1 million rows and Smart Integration Functions that query for long running jobs. • Enhanced debugging capabilities of Smart Integration Functions / Remote Business Rules. • Smart Integration Functions / Remote Business Rules support returning more than 1 million rows by default.
09 Dec, 2024	8.5.0	Updated for release features, including the following enhancements: • Added ability to mask and encrypt Configuration Parameter Values. • Connection strings greater than 245 characters now will encrypt. • Removed redundant copying of DLLs in the "Referenced Assembly" folder upon service startup. • Version number is now displayed in the title bar. • Added Frequently Asked Questions to the documentation. See Frequently Asked Questions.

Date	Release	Summary of Changes
22 Aug, 2024	8.4.0	**CAUTION:** Smart Integration Connector version 8.4 is required when running OneStream version 8.4. Updated for release features, including the following enhancements: • Improved performance and reliability of multi-threaded / parallel processing for larger payloads. • Streamlined process of setting up a redundant Gateway Server. • The active Gateway Server is now displayed within the Gateway Setup in the Windows App. • For Remote Business Rules, the number of rows returned per query threshold has increased to 5M / 5GB of data. • Added ability to check if the gateway is online via a BRAPI.

Date	Release	Summary of Changes
17 Mar. 2024	8.2.0	Updated for release features, including the following enhancements: • Query results that contain NULL values are now being returned. • Added ability to mask the password when creating a database connection string. • Queries that run longer than 10 minutes will consistently return data. • Improved the reliability of multi-threaded connections. • Smart Integration Connector Local Gateway Configuration Utility will automatically open the configuration file for non-default install locations. • DataTable / Datasets can now be sent via a Remote Business Rule.
21 Nov. 2023	8.1.0	Updated to add WebAPI examples.

Date	Release	Summary of Changes
17 Nov. 2023	8.1.0	Updated for release features, including the following enhancements: • Customers can test their Smart Integration Connector Gateways during set-up to ensure there is nothing blocking port 443. • The default Referenced Assemblies folder is in C:\Program Files\OneStream Software\OneStream Gateway\Referenced Assemblies. • The database connection strings in the OneStream Local Gateway Configuration are encrypted when saved. • Specific IPs or CIDRs (a range of IPs) can be whitelisted from the OneStream Windows Client Application. • The OneStream Local Gateway Configuration utility automatically opens the configuration file for the user.

Date	Release	Summary of Changes
21 Aug. 2023	8.0.0	With this release, Smart Integration Connector is a General Availability feature. Updated for release features, including the following enhancements: • The 2GB .NET limit and 1 million return rows is increased to 5GB and 5 million return rows. • Business rules decompress automatically.

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 9.2.0 Release.

  > **IMPORTANT:** For best performance and reliability, upgrade Smart Integration Connector to match your OneStream version. While Smart Integration Connector Local Gateway Server supports the most recent previous version of OneStream, full functionality is not guaranteed.

- Installing Smart Integration Connector Local Gateway Server in your environment.

Overview

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.

The goals for Smart Integration Connector are to establish all required data source connections without a 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.

The following sections explore customer use cases, benefits, common terms, and structural information to help you evaluate Smart Integration Connector for your business needs:

User Considerations

The Smart Integration Connector provides secure connectivity between OneStream SaaS environments and customer-managed data sources. This page explains when Smart Integration Connector is required, recommended, or prohibited, and outlines the steps to request the appropriate connection type.

Required Users

Smart Integration Connector is required if you are a commercial and non-DoD customer in the following cases.

- You use these data connection points: ODBC, SAP, Direct SQL adapters, FTP/SFTP, or similar connections.
- You are upgrading OneStream and have an existing VPN or ExpressRoute connection.

Recommended Users

Smart Integration Connector is strongly recommended if you are a commercial and non-DoD Government customer who uses any data connections. This recommendation ensures secure, reliable connectivity as you move forward with new or expanded integrations.

> **NOTE:** If you are a new customer using a DLL, you will need to implement Smart Integration Connector.

Prohibited Users

Smart Integration Connector is not approved for use by Department of Defense customers.

Alternatives

If you are not a Department of Defense customer and do not implement Smart Integration Connector, consider these alternatives:

- **IP Whitelisting**: See Networking and Whitelisting for details about this process.
- **Direct Connection**: Request a direct connection by opening a Support case at **Cloud Customer Portal** > **Open a Case** > **Customer Support/Remote Consulting**.

If you are a Department of Defense customer, use these alternatives:

- **Direct Connection**: Request a direct connection by opening a Support case at **Cloud Customer Portal** > **Open a Case** > **Customer Support/Remote Consulting**.
- **REST API**: Enable REST API integration by opening a Support case at **Cloud Customer Portal** > **Open a Case** > **Customer Support/Remote Consulting**.

Benefits

The table below details key benefits of implementing Smart Integration Connector.

	Benefits with Smart Integration Connector	Drawbacks without Smart Integration Connector
Change in IP Addresses	Prevents disruptions caused by OneStream IP changes because all connections originate from the customer's environment, eliminating the need to frequently update firewall rules	Can break customer-side firewall rules and require frequent updates, creating repeated maintenance and potential outages

	Benefits with Smart Integration Connector	Drawbacks without Smart Integration Connector
Networking	Stays fully within the customer's control, allowing them to manage security, routing, and access without relying on OneStream	Relies on OneStream-managed routing, access, and firewall dependencies, reducing autonomy and increasing coordination overhead
Credentials and Sensitive Information	Remains inside the customer's environment, reducing security risk and keeping sensitive authentication data out of vendor systems	Must be shared with OneStream, reducing customer ownership of sensitive authentication data
Changes and Updates	Can be self-serviced, enabling faster and more flexible changes	Cannot be self-serviced, requiring support tickets and longer turnaround times for updates

Common Terms

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.
OneStream Windows Application Server (App Server)	The application server executing all OneStream business logic and processing.
Connection (Gateway Connection, Direct Connection)	Connections define direct channels of integration between the OneStream Cloud and a local customer network. Connections are represented by a unique key and are configured for communication to an Azure Relay endpoint. Connections carry a 1:1 correlation to a local gateway on the SIC Local Gateway Server. The channel of communication is established from the OneStream connection and a local gateway is created in Smart Integration Connector Local Gateway Server.

Term	Definition
Gateway Server	A gateway server carries no unique technical definition or configuration address. It is a node in the tree control UI to organize connections and typically corresponds to an installed local gateway server name.
Custom Database Connections (System Configuration)	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 / Port Forwarding (e.g. SFTP, Web API)	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).

Term	Definition
Gateway Connection (e.g. DBs, DLL Integrations)	A gateway connection represents the ultimate datasource destination for Smart Integration Connector, and is typically associated with a local gateway connection that connects to 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.
Smart Integration Function (Remote Business Rule)	A Smart Integration Function (Remote Business Rule) is created in the Windows Desktop Client and compiled and executed on the local gateway server.
IPv4 Whitelist (Whitelisting)	Whitelisting can be applied to the Relay via IP addresses in the OneStream Windows Application client and also applied to your firewall via namespaces through your IT Admin.

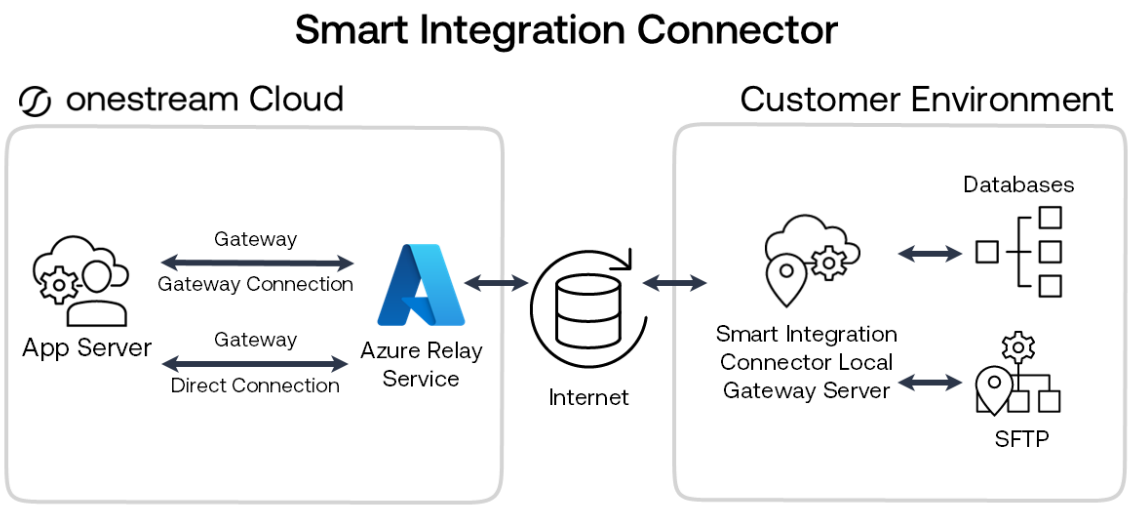
OneStream Local Gateway Configuration Terms

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.
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 gateway must be configured by importing the corresponding gateway definition exported from the OneStream Windows Application client.
Local Gateway Connections	Local gateway connections are typically database connections defined in the utility and confirm the connection between the local gateway and the local data sources.

Term	Definition
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 relationship between the Smart Integration Connector connection gateway and 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. All communication is encrypted end to end through TLS.

The components of the Smart Integration Connector are:

- OneStream Windows Application client

  Direct and Gateway Connections 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.**

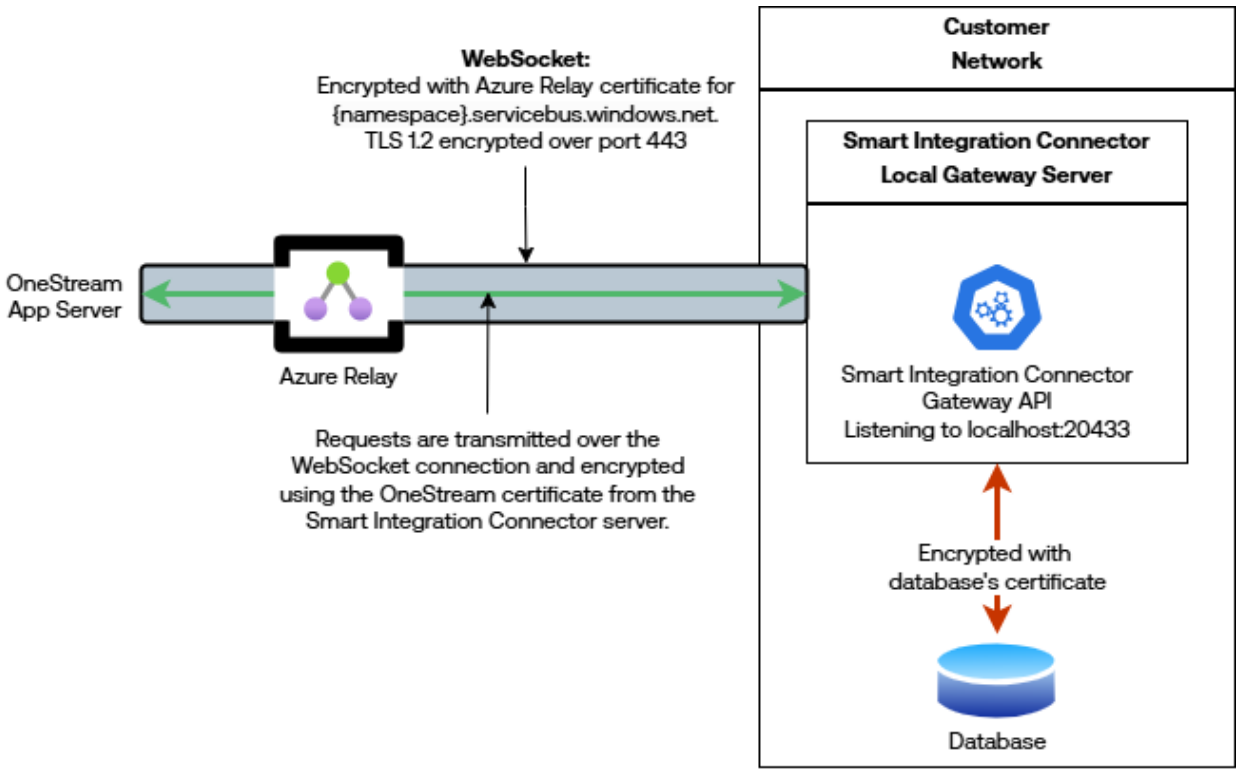
  > **NOTE:** The ManageSystemConfiguration role must be assigned to a user for this to be visible.

- OneStream Smart Integration Connector Local Gateway Server
    - Connection Settings provide the information to establish the relationship with the OneStream Windows Application. Connection Settings are exported from the connection 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.

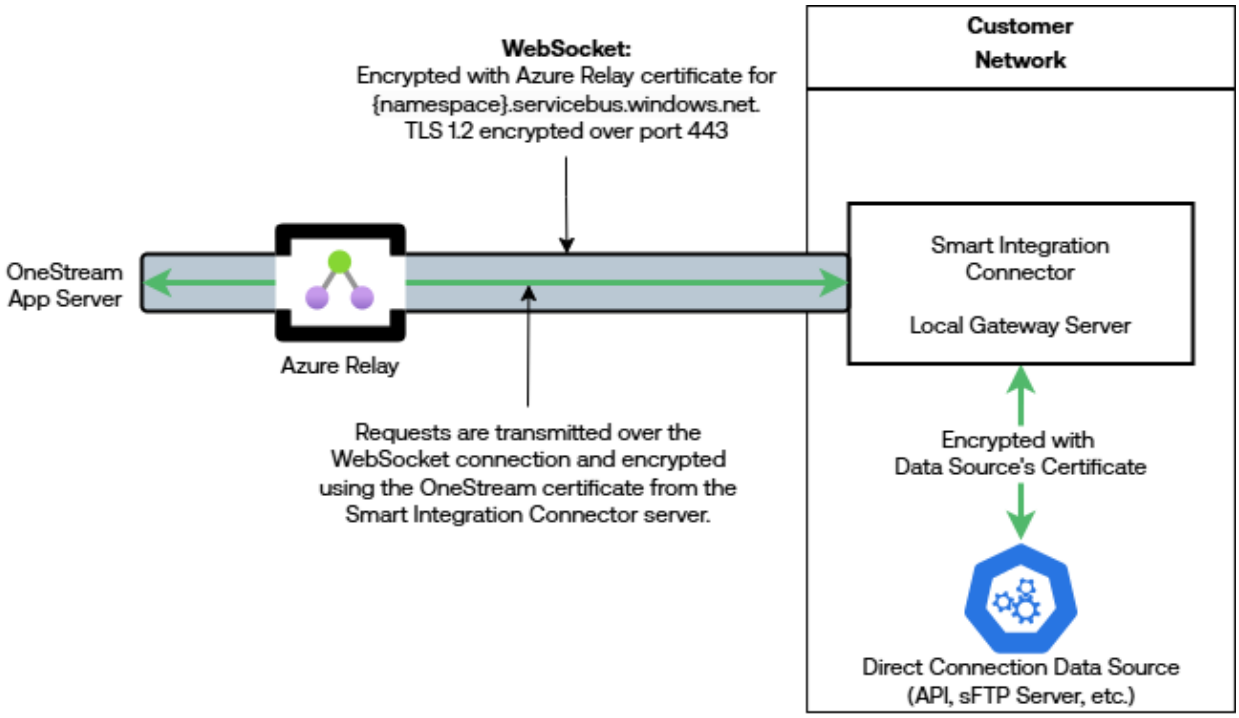
TLS/SSL Certificate

Communication between the OneStream Application Server(s) is encrypted end-to-end. For additional information about certificates and certificate errors, see Troubleshooting.

Gateway Connection example:



Direct Connection example:



> **NOTE:** Certificate errors in the OneStream Application Server caused by a domain name mismatch between the WebAPI domain name and OneStream hostname are ignored. This occurs because the business rule uses `localhost:{boundPort}` for the hostname and the response contains a certificate with a hostname specific to the API (for example, `someapi.org`).

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 connection.
- Multiple local gateways can be installed to establish global connectivity to data sources in different subnetworks.

- Local gateway configuration must align to the corresponding connection as defined in the OneStream Windows application. An export process from the OneStream Windows application connection user interface can assist with the alignment to ensure corresponding names and keys are identical.

Requirements

This section contains information on required elements for Smart Integration Connector. Review these details before installation.

OneStream Smart Integration Connector Environment Setup

- Install compatible versions of the OneStream application and Smart Integration Connector Local Gateway Server.

  > **IMPORTANT:** For best performance and reliability, upgrade Smart Integration Connector to match yourOneStreamversion. While Smart Integration Connector Local Gateway Server supports the most recent previous version ofOneStream, full functionality is not guaranteed.

  - To install or upgrade OneStream to the latest version, see Setup and Installation.
  - To install or upgrade Smart Integration Connector Local Gateway Server to the latest version, see Upgrade Smart Integration Connector.

- Work with your IT team to install the latest version of the Smart Integration Connector Local Gateway Server in an appropriate environment.
  - Windows Server 2019+
  - .NET Framework 4.8
  - 2 newer generation x64 CPU cores (or equivalent virtual processors)

> **NOTE:** ARM processor compatibility testing is outside the current testing scope of Smart Integration Connector.

- Memory (RAM)
  - Minimum 16GB for queries / parallel jobs returning less than 1M / 3M rows or 1GB / 3GB of data respectively.
  - Minimum 32GB for queries / parallel jobs returning less than 5M / 15M rows or 5GB / 15GB of data respectively.

> **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. 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 Connection to be used as the baseline communication between OneStream Cloud and the Smart Integration Connector Local Gateway Server. See Create a Gateway Connection and Create a Direct Connection for more information.
- Be a OneStream administrator to configure corresponding data sources in the OneStream environment.

Installation and Setup

This section contains important details relating to the planning, configuring, and installation of Smart Integration Connector. Before installing, familiarize yourself with these details:

- Setup Overview
- Create a Gateway Connection
- Create a Direct Connection
- Export and Import the Connection Configuration
- Connect a Local Gateway to a Data Source
- Restart the Gateway
- Load Balanced Local Gateway Servers
- Define Custom Database Connections

Setup Overview

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 Smart Integration Connector connection in the OneStream Windows application to connect OneStream Cloud instance to a **Local Gateway on the Local Gateway Server**.

3. Export the connection configuration and import this configuration to the **Gateway Settings** in the **OneStream Local Gateway Configuration**.
4. For Database Connections:
    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:** 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.

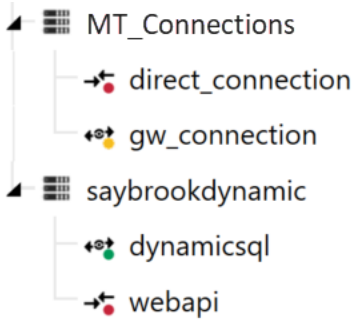
Smart Integration Connector Terms

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

Term	Definition
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.

Term	Definition
IPv4 Whitelist	Contains the list of IPs or CIDR addresses that are allowed to transfer data through Smart Integration Connector.
Name	The name of the connection. Names are completely arbitrary and typically refer to the region (North East) or data source such as (SAP). The name cannot be changed once created, and they must be unique across all environments—both development and production. You can delete an existing connection and recreate it with a new name.
Description	Text describing the role and purpose for the connection and the data sources to which it is connecting.
Gateway Server Name	Use for display and organizational purposes only. This is the name of the gateway server associated with the connection. You can select an existing gateway server or enter a new one.

Term	Definition
Web API Key (Gateway 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 connection 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.

Term	Definition
Status Indicators 	Status indicators in the list of connections provide a visual indication of the **Connection** status. • **Green:** The Connection is **Online**. • **Red:** The Connection is **Offline**. • **Yellow:** (Gateway Connections only) The Connection is **Online** and an update to the Local Gateway Server is available. **NOTE:** For **Direct Connections**, the yellow status is not displayed as these connections do not report a version number back to OneStream.
Instance Count	Displays the number of active gateways. Up to five active gateways per environment are supported. This can be increased by contacting Support.
Version (Gateway 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.

Term	Definition
Active Local Gateway Server Computer Name (Gateway Connections Only)	Displays the computer name of the first Local Gateway Server that connected to the Relay.
Bound Port at Gateway	Remote port bound to Gateway endpoint. Gateway Connections default to 20433 and should not be changed without consulting support. Direct Connections allow any port running on a remote host to be used. This port represents the well-known TCP service to expose from an on-premises host such as sFTP, which would equate to port 22.
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. For example: 172.168.4.7 or sftp.mycompany.com

Term	Definition
Bound Port in OneStream (Direct Connections Only)	The internal port that OneStream uses for secure communications with Direct Connections. The port must be globally unique for every connection in the deployment environment. The port number must be greater than 1024 and less than 65535. The recommended best practice is to automatically assign an available port number when the gateway is created. To automatically assign an unused port, select **(Auto Assigned)**. The port can be referenced in data management or business rules to directly access services such as sFTP and WebAPI.
Gateway failures reporting interval (min)	Minutes to wait between reporting connection failures into the OneStream Error Log. The default is five minutes and the max is 1440 minutes. If a connection is unreachable, an item is put in the error log using this interval value in minutes and the minutes can be adjusted.

Local Gateway Server Installation

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

1. Download the Smart Integration Connector installer (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. It is best practice not to rename, move, or change the location of the XFGatewayConfiguration.xml file. If the configuration file has to be opened from another location, then it will need to be opened from this other location again after the upgrade.

6. Save the configuration file.

7. Follow the dialog prompts and restart the service.

Create a Gateway Connection

Gateway Connections are used to connect OneStream to the Smart Integration Connector Local Gateway Server over the Azure Relay. At least one Gateway Connection is required for Smart Integration Connector to function properly. After the connection is created, you will need to copy the configuration to the Smart Integration Connector Local Gateway Server using the OneStream Local Gateway Configuration.

> **NOTE:** For descriptions of the fields in this procedure, see Smart Integration Connector Terms.

To create a Gateway Connection :

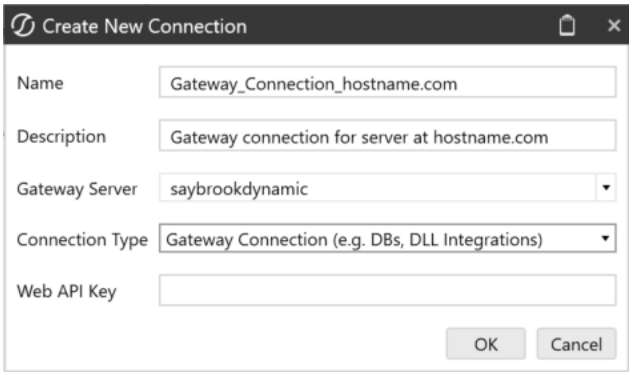
1. Go to **System** > **Administration** > **Smart Integration Connector**.
2. Click  **Create New Connection**.
3. Enter the **Name** and **Description**.

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

4. Select the **Gateway Server** from the drop-down, or enter a new Gateway Server name in the same field. If this is the first Connection being created, enter the name of the Gateway Server.

> **NOTE:** It is common practice to use the hostname or IP of your Smart Integration Connector Local Gateway Server as the "Gateway Server" name. The Gateway Server name is only used to organize servers when multiple Smart Integration Connector servers are set up.

5. From **Connection Type**, select **Gateway Connection**.



> **NOTE:** Each Gateway Server will only need one Gateway Connection and as such we recommend naming it "Gateway_Connection_" followed by the name used for "Gateway Server." This will differentiate the Gateway Connections in future steps.

6. The Web API Key is used as an additional layer of security when communicating with the Smart Integration Connector Local Gateway Server internal APIs.

> **NOTE:** WebAPI keys are not required, but are best practice to enhance security and can be modified or added at any time. 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 is suggested to use the WebAPI as a mechanism to enhance security on the remote network preventing unauthorized use of OneStream WebAPIs.

7. Copy the configuration to the Smart Integration Connector Local Gateway Server using the OneStream Local Gateway Configuration application. For details, see Export and Import the Connection Configuration.

Create a Direct Connection

A Direct Connection is a point-to-point channel to a specific remote network resource such as an sFTP server or Web API (including iPaaS services).

> **NOTE:** At least one gateway connection is required to use a Direct Connection because the gateway connection is used to monitor the availability of the remote Smart Integration Connector Gateway servers.
>
> The existence of a gateway connection does not necessarily mean it must be used or configured to a data source if only Direct Connections are desired.

> **NOTE:** For descriptions of the fields in this procedure, see Smart Integration Connector Terms.

To create a direct connection:

1. (Required) A Gateway Connection must be created before the Direct Connection is created. The Gateway Connection is used to monitor the availability of the remote Smart Integration Connector Gateway server. For details, refer to Create a Gateway Connection.
2. Go to **System** > **Administration** > **Smart Integration Connector**.
3. Click  **Create New Connection**.
4. Enter the **Name** and **Description**.

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

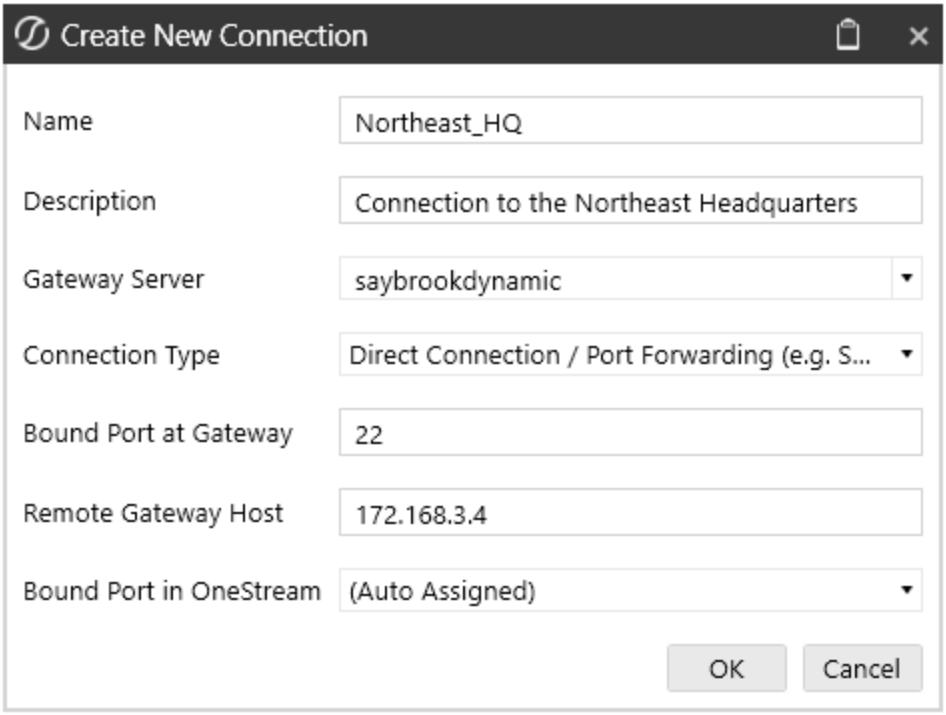
5. Select the **Gateway Server** from the drop-down, or enter a new Gateway Server name in the same field. If this is the first Connection being created, enter the name of the Gateway Server.

   > **NOTE:** It is common practice to use the hostname or IP of your Smart Integration Connector Local Gateway Server as the "Gateway Server" name. The Gateway Server name is only used to organize servers when multiple Smart Integration Connector servers are set up.

6. From **Connection Type**, select **Direct Connection / Port Forwarding**.
7. 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:** The remote service port is required to configure the connection and may require consultation with network or IT resources to obtain it. It is also required that any firewalls between the Local Gateway Server and the remote host allow traffic to the destination port specified.

8. Enter the **Remote Gateway Host**. 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.
9. For **Bound Port in OneStream**, select **(Auto Assigned)** (default and recommended setting) or **Enter Port Manually**. See Create a Direct Connection for additional information.
    - **(Auto Assigned)** (default and recommended setting) to allow the OneStream application to automatically assign an unused port number. When the Direct Connection is created, the port number is shown in the connection settings.
    - **Enter Port Manually**: Enter an unused port number. The port number must be greater than 1024 and less than 65535.
10. Click **OK**.
11. 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.



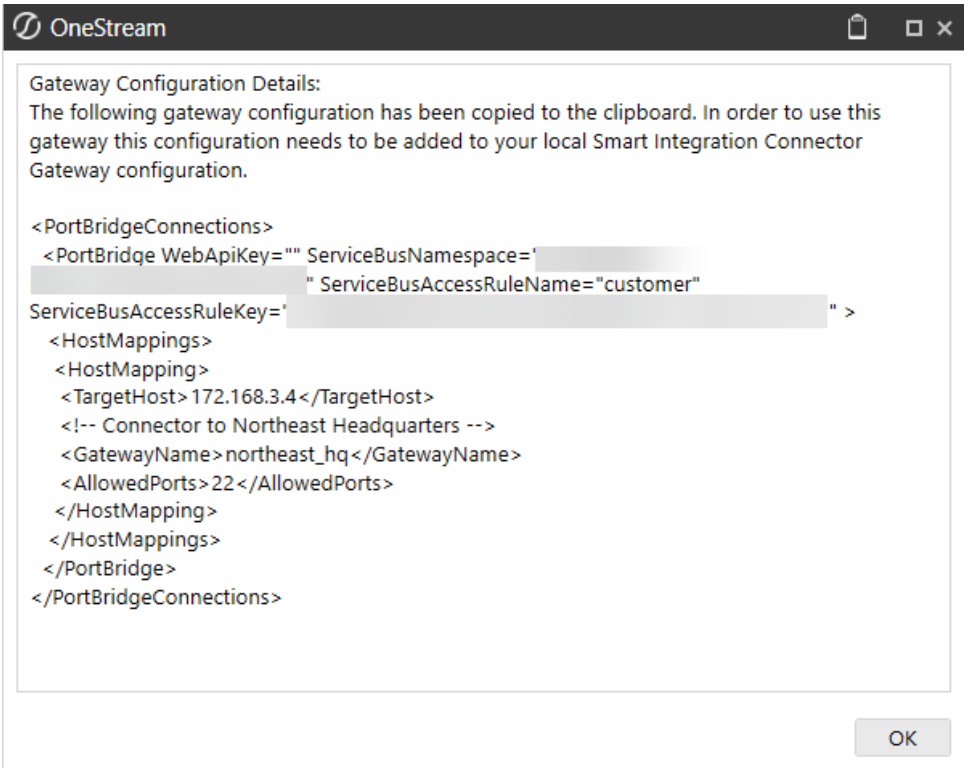
- Example: Remote SFTP server at 172.168.3.4 listening on port 22.
- Bound Port in OneStream defaulted to (Auto Assigned).
  Note that when **(Auto Assigned)** is used, the selected port number is available/displayed after saving and also surfaced in the OneStream Error Log.
- In OneStream Business Rules, you can access the remote host by connecting to localhost:"Bound Port in OneStream" portExample: `localhost:45000`.
- 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
```

Export and Import the Connection Configuration

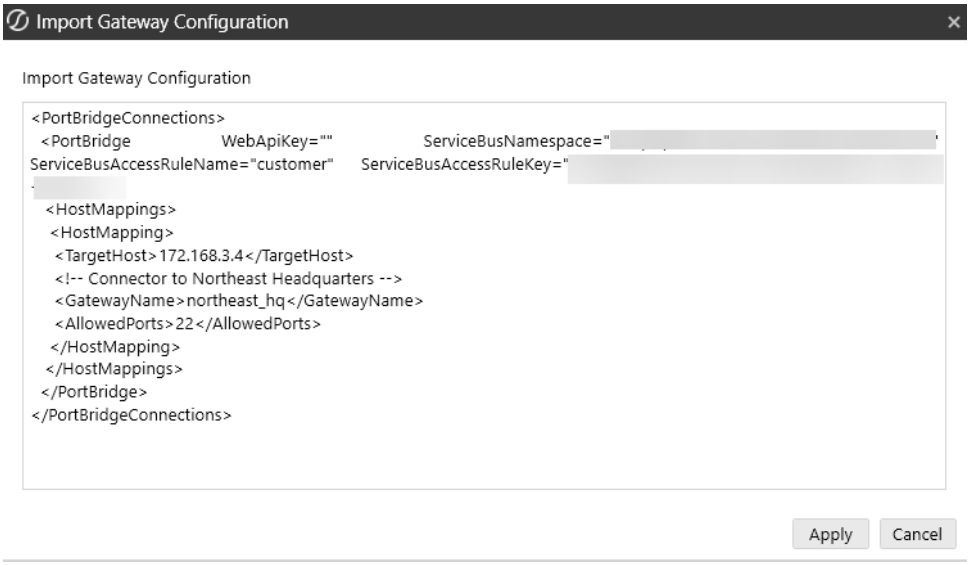
You must copy the connection configuration settings and paste them into your Smart Integration Connector Gateway to establish the connection.

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

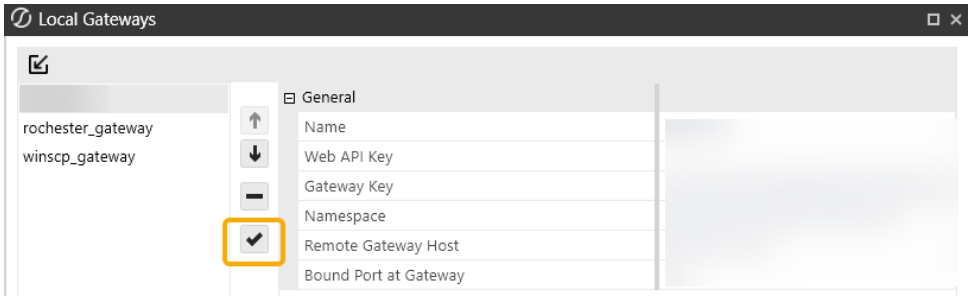


4. On your Windows Server, open the **OneStream Local Gateway Configuration**. This runs as administrator by default.
5. The existing **XFGatewayConfiguration.xml** opens by default.
6. Click ··· next to **Local Gateway Settings**.

7. Import **Import Gateway Configuration** to import the previously copied **Connection Configuration**.



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

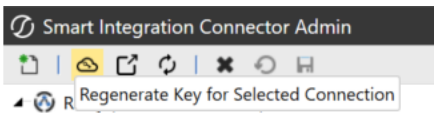


10. Click **OK** twice.
11. Save the configuration.
12. Click **Yes** to apply the changes and restart the Local Gateway Server.

New Key Generation

Smart Integration Connector administrators can rotate the 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 in the OneStream Windows Application to function properly.

1. Select an existing connection.
2. Click **Regenerate Key for Selected Connection**.

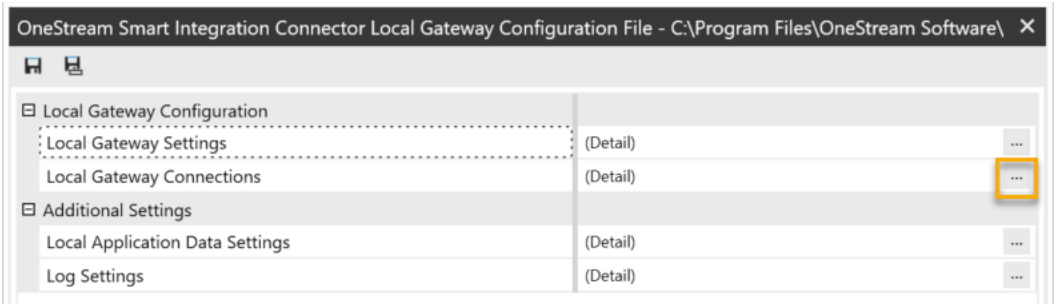


3. You must re-export your Connection Configuration and apply the new settings throughout the OneStream Local Gateway Configuration. See Export and Import the Connection Configuration.
4. Click **OK**.

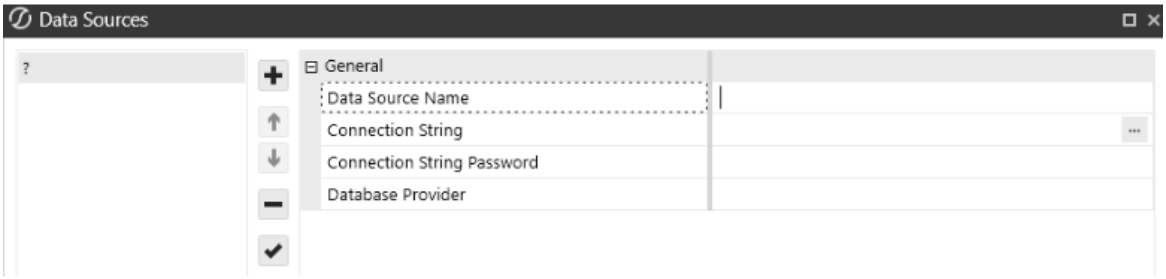
Connect a Local Gateway to a Data Source

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. The data source is configured using the Local Gateway Configuration Utility. The utility was installed as part of the Smart Integration Connector Local Gateway installation.

1. Start the **OneStream Local Gateway Configuration**.
2. The existing **XFGatewayConfiguration.xml** opens by default.
3. Click ··· **More** next to **Local Gateway Connections** to set up the **Data Sources** to local databases, APIs, or other on-premises resources.



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



5. If you have a password for the connection string, enter it in the **Connection String Password** field. The password is masked for security. Then, when you need to enter your connection string password, use the substitution variable: `|password|`
   Example: `Data Source=localhost;Initial Catalog=Sales_DB;Persist Security Info=True;User ID=sa;Password=|password|;`

6. Enter the **Data Source Name**, **Connection String**, and select a **Database Provider**.

   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

   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 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.

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

   > **IMPORTANT:** The connection strings below include user IDs and the password substitution variable. You can also use integrated security to remove plain text user IDs and passwords from connection strings in Smart Integration Connector. See Remove UserID and Passwords by Integrated Security.

Microsoft SQL Server

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

1. Click ··· **More** next to **Local Gateway Connections**.
2. Click  **Add Item** to add the data source.
3. **Data Source Name:** Northeast_Sales
4. **Connection String:**
   with UserID / Password:
   Server=localhost;Initial Catalog=Sales_DB;User ID=sa;Password=|password|;Max Pool Size=1000;Connect Timeout=60;
5. Enter your **Connection String Password**.

   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

6. From Database Provider, select **SqlClient Data Provider**.
7. Click  **Test Connection** to test the data source.
8. Click **OK** to save.

MySQL Data Provider

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

1. Click ··· **More** next to **Local Gateway Connections**.
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=|password|;database=gatewaymysql;
5. Enter your **Connection String Password**.

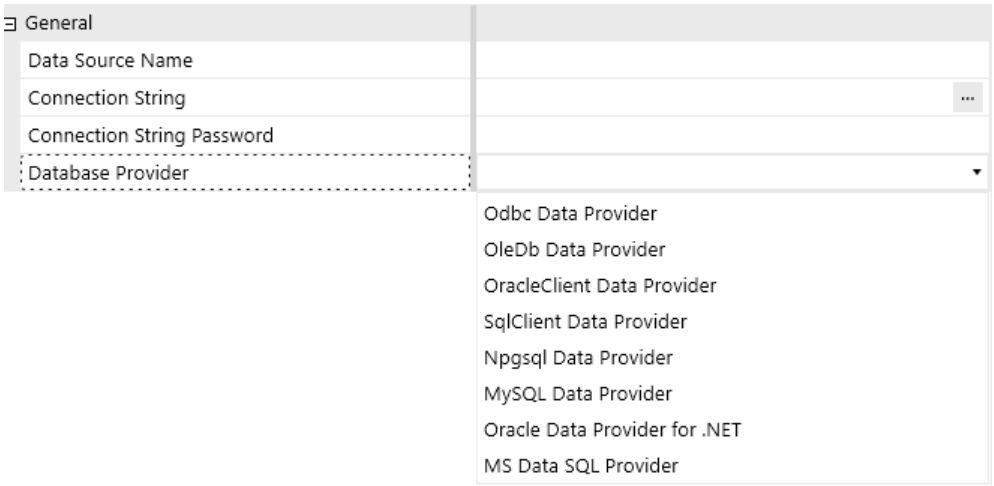
   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

6. From **Database Provider**, select **MySQL Data Provider**.
7. Click  **Test Connection** to test the data source.
8. 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 ODAC 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=|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=|password|;

OracleClient Database Provider

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

1. Click ··· **More** next to **Local Gateway Connections**.
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=|password|
5. Enter your **Connection String Password**.

   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

6. From **Database Provider**, select **OracleClient Data Provider**.

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

Oracle Data Provider for .NET

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

1. Click **More** next to **Local Gateway Connections**.
2. **Data Source Name:** Sales_SouthAmerica
3. **Connection String:**
   Data Source=oracletest;User Id=OneStream1;Password=|password|
4. Enter your **Connection String Password**.

   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

5. From **Database Provider**, select **Oracle Data Provider for .NET**.
6. Click **Add Item** to add a new data source.
7. Click **Test Connection** to test the data source.
8. Click **OK** to save.

PostgreSQL (Npgsql Data Provider)

Below is an example for setting up a PostGres database.

1. Click ··· **More** next to **Local Gateway Connections**.
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=|password|;
5. Enter your **Connection String Password**.

   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

6. From **Database Provider**, select **Npgsql Data Provider**.
7. Click **Test Connection** to test the data source.
8. Click **OK** to save.

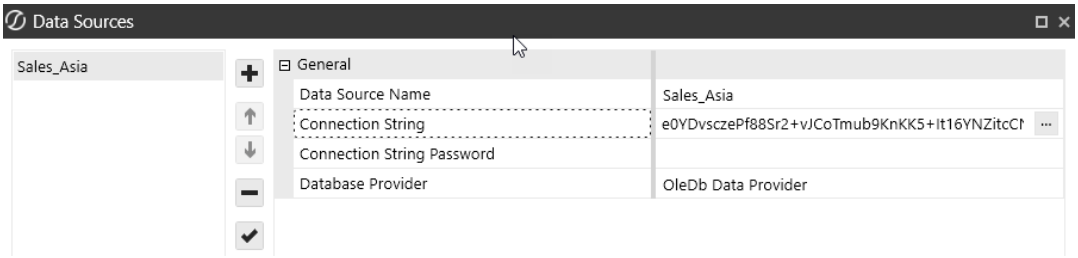
OleDb Data Provider

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

1. Click ··· **More** next to **Local Gateway Connections**.
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=|password|;
5. Enter your **Connection String Password**.

   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

6. From **Database Provider**, select **OleDb Data Provider**.
7. Click **Test Connection** to test the data source.
8. Click **OK** to save.



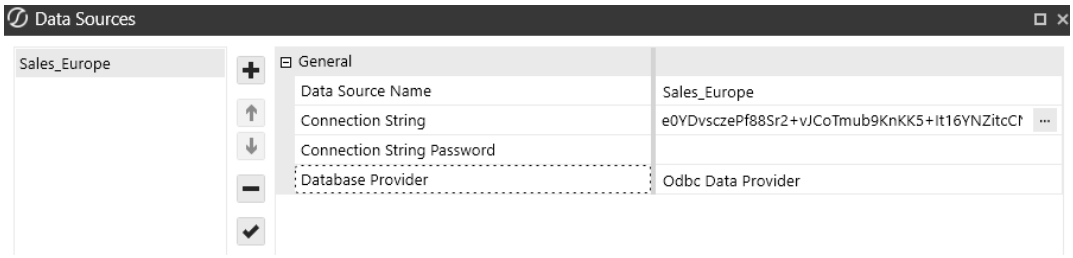
ODBC Data Provider

ODBC data sources can be defined (using a system DSN) to remove credentials from the configuration file. For ODBC connections, most ODBC drivers will allow you to set up a system DSN entry on the server, then the connection string in the gateway will be to only point to the DSN entry. See Administer ODBC data sources for more information. Below is an example for setting up an ODBC data source for Oracle.

1. Click ··· **More** next to **Local Gateway Connections**.
2. Click + **Add Item** to add the data source.
3. **Data Source Name:** Sales_Europe
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=|password|;
5. Enter your **Connection String Password**.

> **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

6. From **Database Provider** , select **Odbc Data Provider**.
7. Click **Test Connection** to test the data source.
8. Click **OK** to create the new source.
9. Click **Save**.

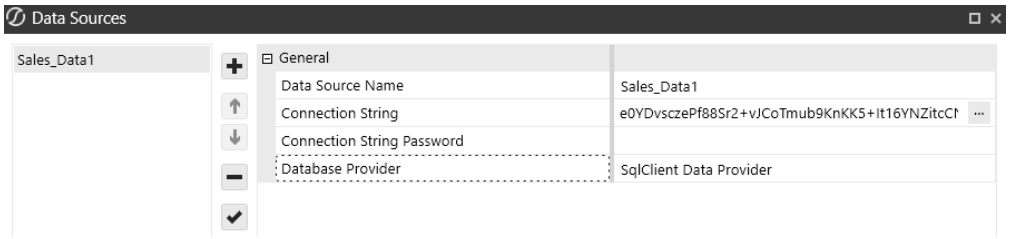


(Optional) Remove UserID and Passwords by Integrated Security

You can remove UserIDs 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 DSN) 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**.
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.

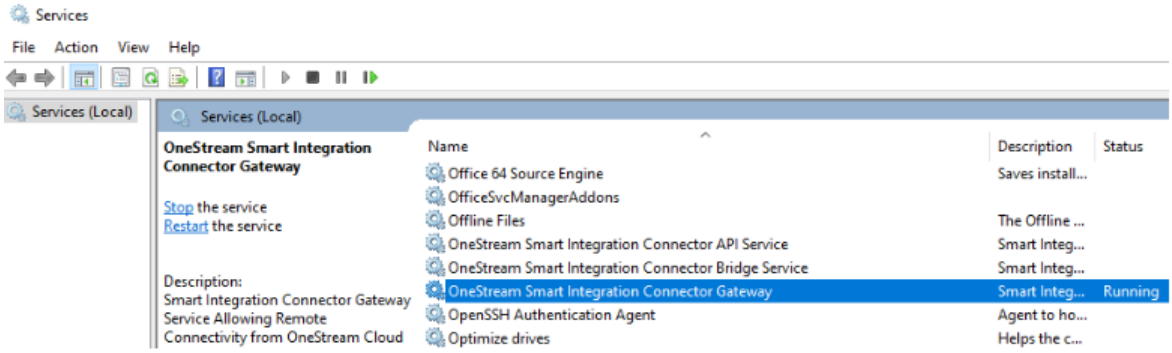
4. Click **OK**.
5. Save your **Data Source**.

Update Permissions on the OneStream Smart Integration Connector Gateway 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.

Smart Integration Connector remote query failed.
One or more errors occurred. (The SSL connection could not be established, see inner exception.)
One or more errors occurred. (The SSL connection could not be established, see inner exception.)
The SSL connection could not be established, see inner exception.
Received an unexpected EOF or 0 bytes from the transport stream.

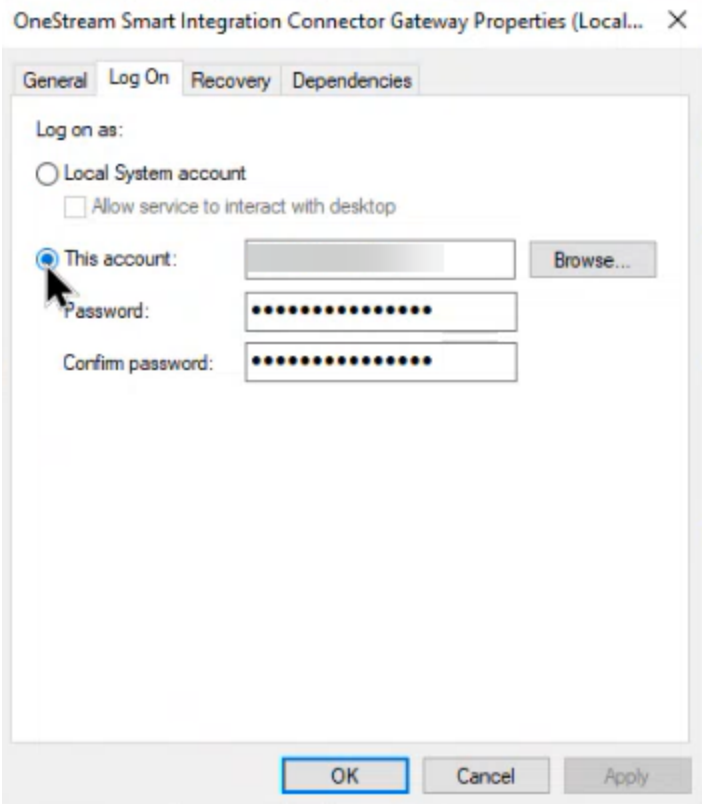
1. Open Windows Services.
2. Navigate to **OneStream Smart Integration Connector Gateway**. The service should be running.



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

> **IMPORTANT:** Before moving to the next step, ensure that you have the appropriate permissions and approvals from your IT Administrators to complete the Log On change. The service account used will require local Administrative rights to access resources on the Windows server, such as the machine certificate store and private keys used for encryption. This account will also require the appropriate permissions to access the database such as Microsoft SQL Server.

5. Change log on from **Local System account** to **This account** and enter your domain or login that has access to the data source. 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 about 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.

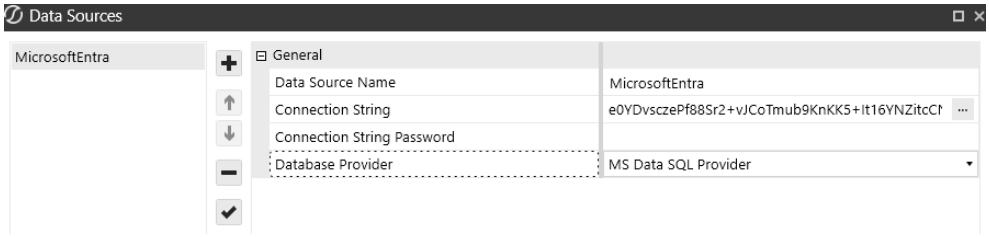
Microsoft Entra Authentication for Azure SQL

The ability to use Microsoft Entra using service principal authentication to access Azure SQL is supported.

1. Open your **OneStream Local Gateway Configuration**.
2. Open a **Local Gateway Connection**.
3. Enter a **Data Source Name** of **MicrosoftEntra**.
4. Navigate to the Connection String and enter a connection string. Example: Server=demo.database.windows.net; Authentication=Active Directory Service Principal; Encrypt=True; Database=testdb; User Id=AppId; Password=|password|;
5. Enter your **Connection String Password**.

   > **NOTE:** For security purposes, we recommend using the Connection String Password field and the substitution variable to ensure the password is not shown on screen. However, you can also embed the password directly within your connection string. For example: Server = localhost;Port=3306;uid=root;pwd=my_password;database=gatewaymysql;.

6. Select **MS Data SQL Provider** as your **Database Provider**.

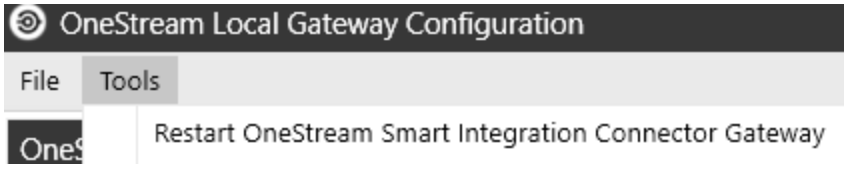


7. Click **Test Connection** to test the data source.
8. Click **OK**.
9. Click **Save**.

Restart the Gateway

After communication has been verified, the following Windows Service needs to run 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 are having issues restarting the service, see Troubleshooting.

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



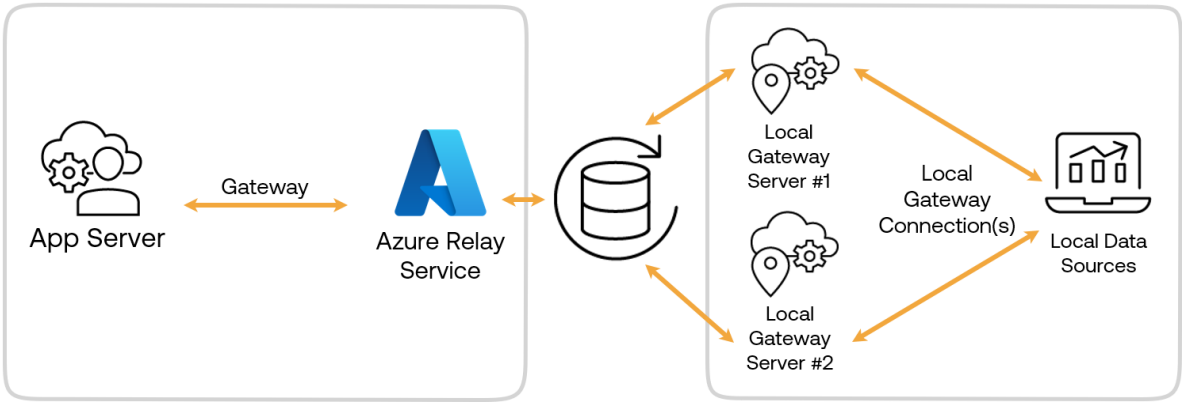
Load Balanced Local Gateway Servers

To create a load balanced environment, install the Smart Integration Connector Local Gateway Server on a separate Windows Server. Load balanced environments allow for faster overall results from data sources by:

- Distributing requests between servers
- Improving performance
- Providing a failover server

In a load balanced environment, consider the following:

- Incoming connections are balanced between the available Local Gateway Servers.
- The first Local Gateway Server to establish a connection to the Relay is displayed in the **Active Local Gateway Server Computer Name** field in the Smart Integration Connector Admin setup.
- The number of active gateways is displayed in the **Instance Count** field.



> **NOTE:** To set up local gateway servers for multiple environments, you must create a new server for each environment by following the steps in Local Gateway Server Installation.

Create a Load Balanced Local Gateway Server

To create a load balanced Local Gateway Server:

1. On the first Windows Server, complete installation on the initial Local Gateway Server and verify all data connections transfer data.
2. After all connections have been verified on the first Windows Server export the configuration.
    a. Open the **OneStream Local Gateway Configuration**.
    b. Go to **Tools** > **Export Configuration for Backup Gateway Server**.
    c. Choose the location and select Save.
3. On the second Windows Server in your environment, install the **OneStream Smart Integration Connector Local Gateway Server** (OneStreamSmartIntegrationConnectorGateway-#.#.#.#####.msi).

    > **NOTE:** If you are using custom DLLs, SAP, or referenced DLLs, you must copy the existing Referenced Assemblies Folder. Locations must be in sync and in the same location on the primary server. See Smart Integration Additional Settings.

4. On the second server, perform the following steps:

a. Open the **OneStream Local Gateway Configuration**.

b. Go to **Tools** > **Import Configuration from Primary Gateway Server**.

   i. Choose the location of the export file and select Open.

   > **CAUTION:** You will overwrite the existing local gateway configuration. If you use Connection String Passwords, you will need to reenter a connection string password.

   > **CAUTION:** If you installed a custom database driver, you must install the customer database driver on the backup gateway server.

   ii. Click **Local Gateway Connections**.

   iii. Select a **Data Source** and the **Connection String Passwords**.

   iv. Select **OK** to provide a new Connection string.

   v. Delete the encrypted text and replace it with a valid connection string from the primary server.

   vi. Select **OK** to encrypt the connection string and close the dialog box.

   vii. Repeat steps above for all the remaining data sources.

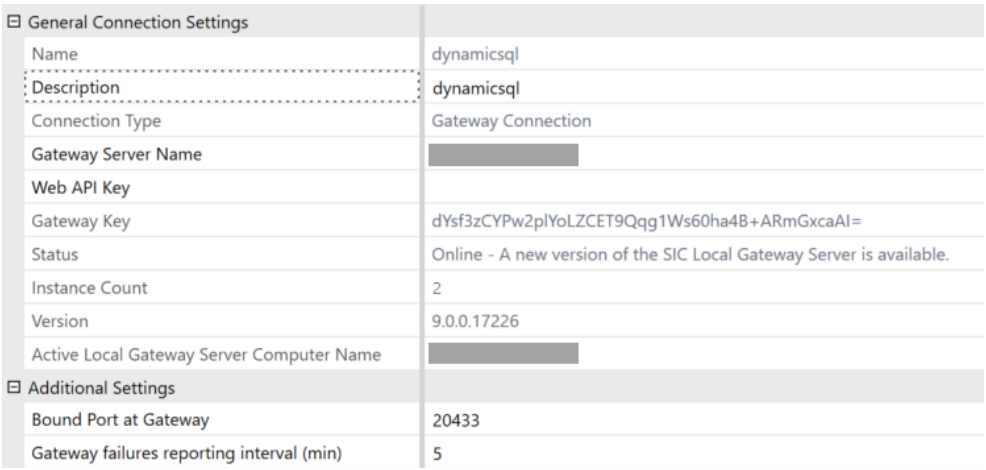
   viii. Click **OK** to close the **Local Gateway Connections**.

   ix. Click **Save** to save the **Local Gateway Configuration**.

   x. Click **Yes** to restart the service.

   xi. Test the Smart Integration Connector Local Gateway Server in OneStream.

5. Verify the **Instance Count** is **2** when both the initial and second servers are running in the OneStream Windows application. The first Local Gateway Server to establish a connection to the Relay is displayed in the **Active Local Gateway Server Computer Name** field

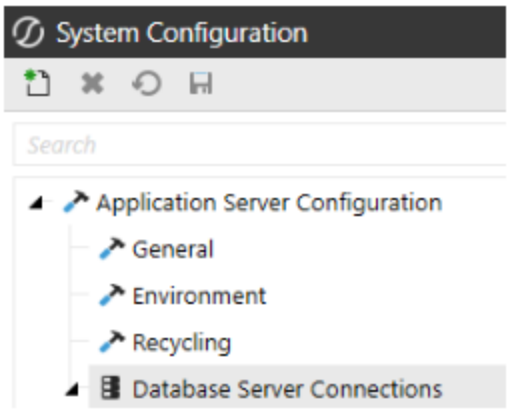


⊟ General Connection Settings	
Name	dynamicsql
Description	dynamicsql
Connection Type	Gateway Connection
Gateway Server Name	[redacted]
Web API Key	
Gateway Key	dYsf3zCYPw2plYoLZCET9Qqg1Ws60ha4B+ARmGxcaAI=
Status	Online - A new version of the SIC Local Gateway Server is available.
Instance Count	2
Version	9.0.0.17226
Active Local Gateway Server Computer Name	[redacted]
⊟ Additional Settings	
Bound Port at Gateway	20433
Gateway failures reporting interval (min)	5

Define Custom Database Connections

Now that the connection 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.

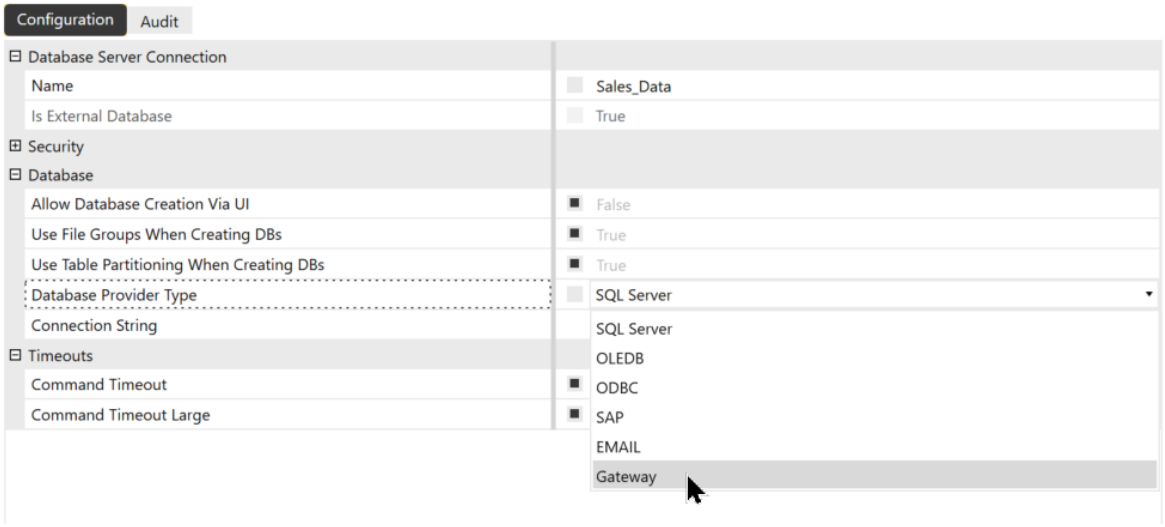
1. Go to **System** > **Administration** > **System Configuration**.
2. Select **Application Server Configuration** > **Database Server Connections**.



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.

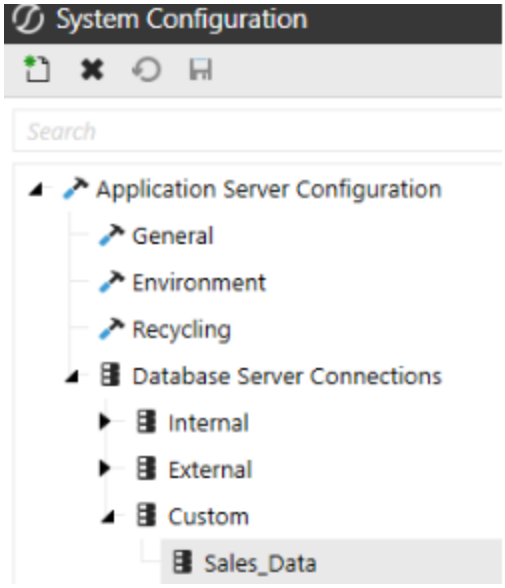
4. Enter the **Name** of the **Database Server Connection**.
5. For **Database Provider Type**, select **Gateway**.



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. It is advised to wait up to five minutes for the Gateway to populate first.

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


System Configuration
Search
Application Server Configuration
General
Environment
Recycling
Database Server Connections
Internal
External
Custom
Sales_Data

Upgrade Smart Integration Connector

The following section describes how to upgrade Smart Integration Connector.

> **IMPORTANT:** For best performance and reliability, upgrade Smart Integration Connector to match your OneStream version. While Smart Integration Connector Local Gateway Server supports the most recent previous version of OneStream, full functionality is not guaranteed.

Upgrade from:

- Private Preview versions 7.2, 7.3,
- Limited Availability version 7.4, or
- General Availability versions 8.x to 9.2

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 connections 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.

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 and sub folders before upgrading. Default is: `C:\Program Files\OneStream Software\OneStream Gateway\`.
5. Follow the steps in Installation and Setup to complete your upgrade.

> **NOTE:** If the upgrade process is interrupted or canceled, the Smart Integration Connector must be reinstalled.

If the Smart Integration Connector Windows Service was configured to start using a custom service account prior to upgrading, confirm that the service is set to start using the correct service account after the upgrade is completed.

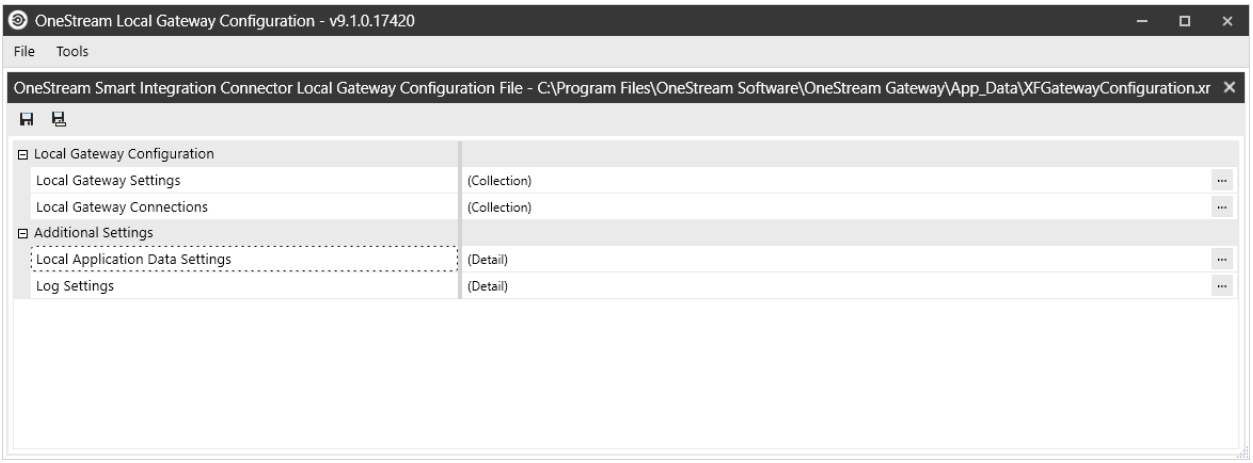
NOTE: For OneStream Local Gateway Server version 8.1 and above, the default location for Reference Assembly Folder is C:\Program Files\OneStream Software\OneStream Gateway\Referenced Assemblies.

Smart Integration Additional Settings

You can apply additional settings in the Smart Integration Connector Local Gateway Configuration file.

Additional settings include:

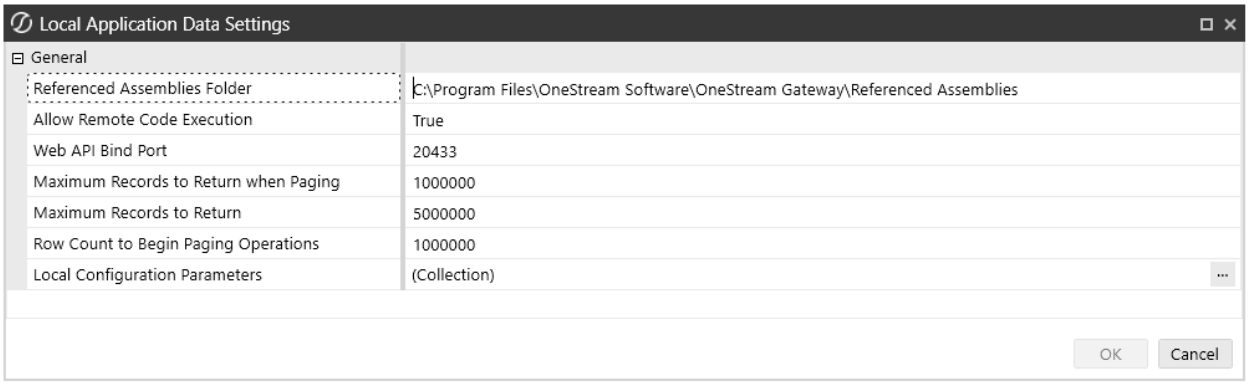
- Local Application Data Settings
- Log Settings



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.



You can:

- Reference a location to additional DLLs that will be used in remote business rules.
- Adjust the Maximum Records to Return. 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. You will need to add the DLL name to the Smart Integration Functions Referenced Assemblies property. The default value is C:\Program Files\OneStream Software\OneStream Gateway\Referenced Assemblies.

> **NOTE:** If you are integrating SAP with ERPConnect, add ERPConnect and the required DLLs to the Referenced Assemblies folder and C:\Windows\System32 folder per instructions. Refer to Support for DLL Migration.

Allow Remote Code Execution

The Smart Integration Connector Capabilities introduce additional business rule APIs (BRAPIs) 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 Web APIs hosted in your network. Set to True by default.

Web API Bound Port

The port that Smart Integration Connector uses to communicate with the internal API.

Maximum Records to Return when Paging

Defaults to 1,000,000 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 5,000,000 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 1,000,000 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 a best practice 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.

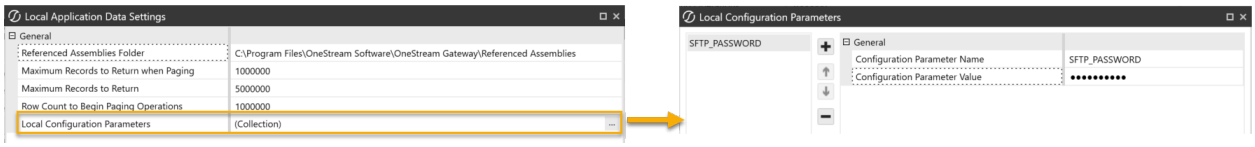
Row Count to Begin Paging Operations

Defaults to 1,000,000 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 SFTP_PASSWORD. 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.



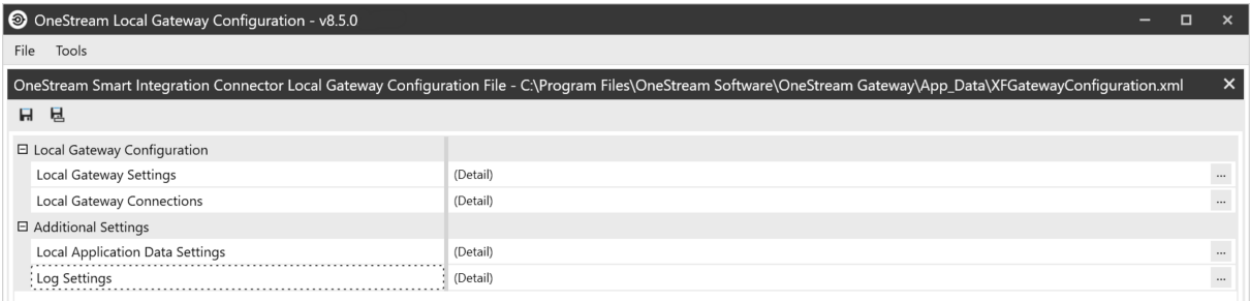
> **NOTE:** Configuration Parameter Values are masked and encrypted by default. When setting up the parameter, you will have the option of always showing the parameter in plain text.

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.

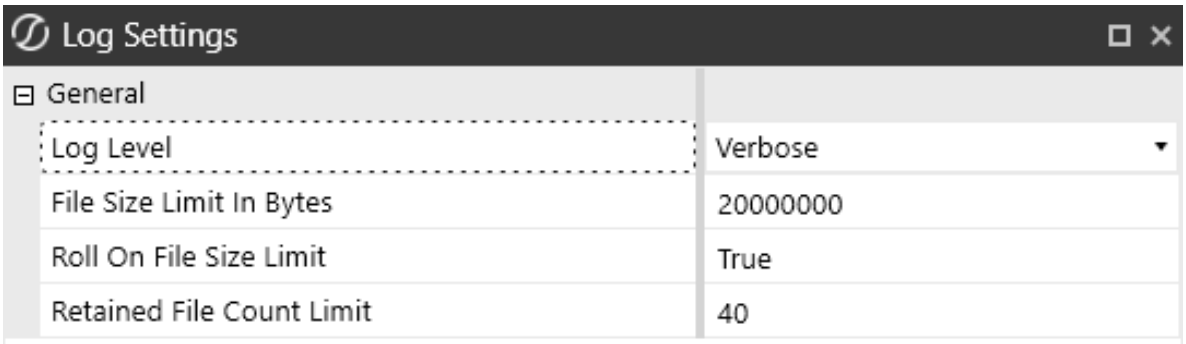
```
Dim passwordString As String = APILibrary.GetSmartIntegrationConfigValue("SFTP_PASSWORD")
```

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**.



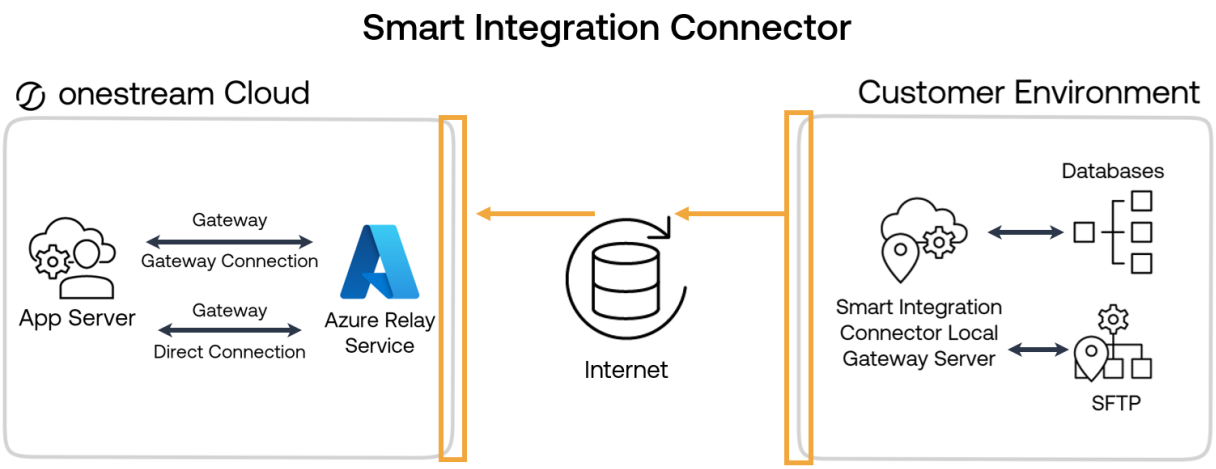
- **Log Level** descriptions:
  - **Verbose**: The noisiest level, rarely (if ever) enabled for a production application.
  - **Debug**: Used for internal system events that are not necessarily observable from the outside, but useful when determining how something happened.
  - **Information**: Used 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 and is the default setting upon installation.
  - **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.

> **NOTE:** The log file's output has been updated to reflect the enhanced performance and reliability of multithreaded or parallel processing for larger payloads since the Platform Version v8.4 update.

Networking and Whitelisting

Smart Integration Connector requires outbound traffic over port 443 to function. If you restrict outbound traffic over 443 then whitelisting outbound traffic to Azure Relay Service will be required. Smart Integration Connector does not require any inbound access rules to function.



These sections contain steps to restrict and whitelist traffic for Smart Integration Connector:

- Restrict Traffic to the Azure Relay
- Whitelist Outbound Traffic

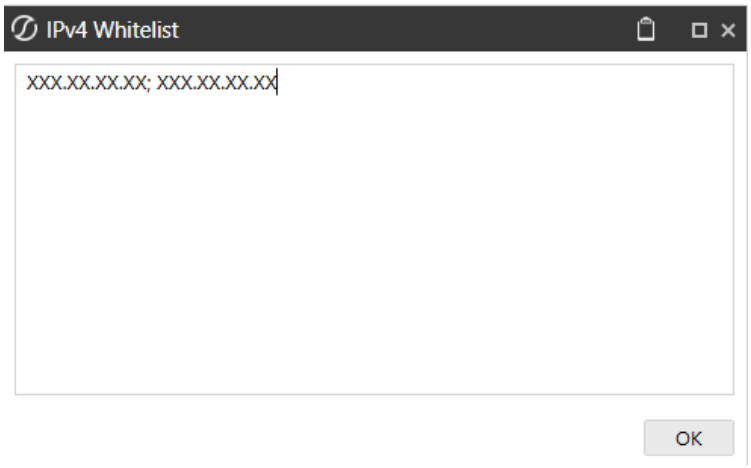
Restrict Traffic to the Azure Relay

You can block or restrict traffic to your Azure relay to only allow certain IP ranges to connect.

1. From the OneStream Windows Application client go to **System** > **Administration** > **Smart Integration Connector** > **Relay**.
2. Select **IPv4 Whitelist**.
3. Enter IPv4 compatible IP (XXX.XXX.XXX.XXX) or CIDR addresses (XXX.XXX.XXX.XXX/XX) separated by a semi colon in the **IPv4 Whitelist** dialog box.

   > **NOTE:** IPv6 addresses are not currently supported.

   > **NOTE:** Do not include any extra spaces for characters.

4. Restart your Local Gateway Service.

Whitelist Outbound Traffic

Allow outbound traffic using a wildcard domain to the Azure Relay Service (best practice). If the firewall does not allow wildcards, use the fully qualified domain names for your specific Azure Relay namespaces.

> **NOTE:** For additional information, see Azure Relay WCF and Hybrid Connections DNS Support.

Allow Traffic using Wildcard Domain (Best Practice)

To allow traffic using a wildcard domain (Microsoft Recommended best practice), add `*.servicebus.windows.net` to the firewall rules permitting port 443 outbound.

Allow Traffic using IP addresses (Not Recommended)

To allow traffic using fully qualified domain names to the firewall rules:

1. Look up the IP addresses used by the Azure Relay namespace. The IP addresses can be returned by using this script.
2. Add the IP addresses to the firewall rules permitting port 443 outbound.
3. Frequently monitor the IP addresses for changes. Update the IP addresses in the firewall rules when there are IP address changes. The IP addresses can be returned by using this script.

   > **NOTE:** Up to 20% of the IP address can change in the span of a month. To ensure that Smart Integration Connector continues to operate, you will need to frequently monitor if these IPs change and adjust your firewall accordingly.

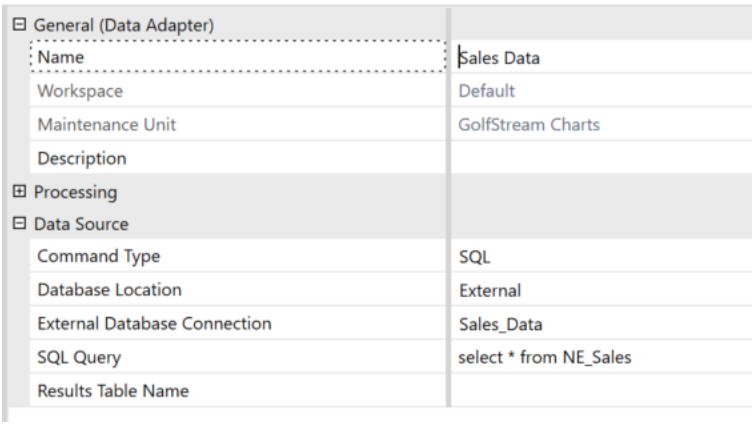
Use Smart Integration Connector

You can use Smart Integration Connector to access data from your Local Gateway Connection Data Sources or through Direct Connections. This section provides examples of how to use Smart Integration Connector:

- Data Adapters Example
- SQL Table Editor Example
- Grid View Example
- Perform a Drill Back
- Perform a Write Back
- Support for SFTP
- Transfer Files from Local FileShare
- Obtain Data through a WebAPI
- Send Emails through Direct Connections
- Support for DLL Migration

Data Adapters Example

1. Go to **Application** > **Presentation** > **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.

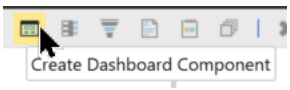


5. Click  **Test Data Adapter** to test the **Data Adapter** and view the results.

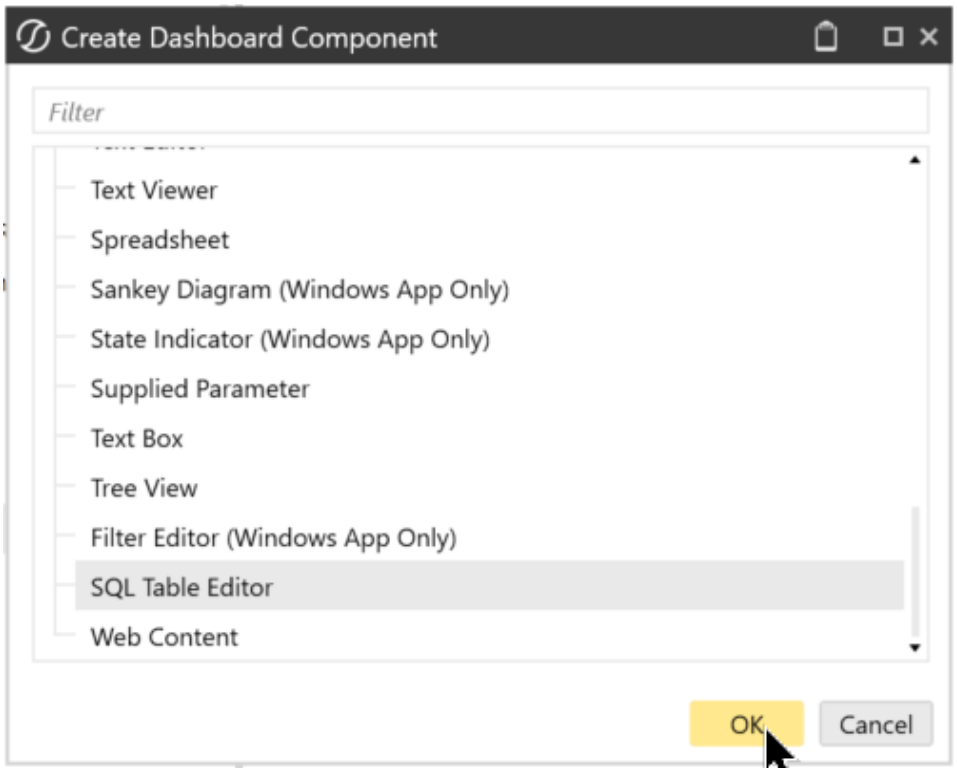
SQL Table Editor Example

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

1. Go to **Application** > **Presentation** > **Workspaces** > **[choose Workspace]** > **Maintenance Unit** > **[choose Maintenance Unit]** > **Components** > **SQL Table Editor**.
2. Create Dashboard Component or open a SQL Table Editor.



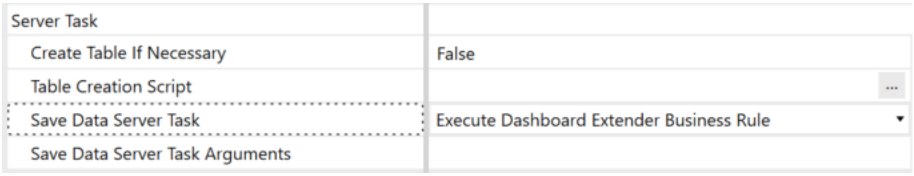
3. Choose SQL Table Editor and select OK.



4. 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.

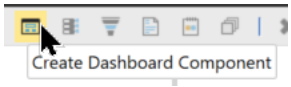
5. Open the associated dashboard and run the query. The OneStream Smart Integration Connector will connect to the external database. If it connects correctly, the query will populate.

> **NOTE:** If using SQL Server with SQL Table Editor, no business rule modifications are needed. If you plan on modifying data with SQL Table Editor using Smart Integration Connector to an Oracle or PostGres database, then you will need to write back data with a custom business rule using the **Execute Dashboard Extender Business Rule** feature under the **Save Data Server Task** action.

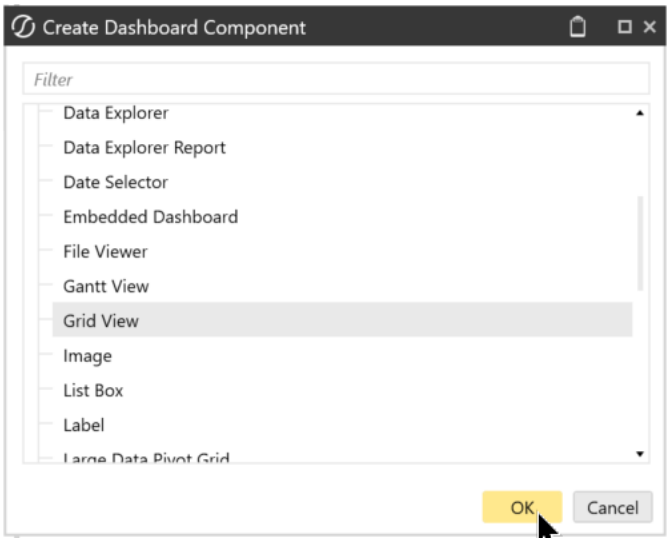


Grid View Example

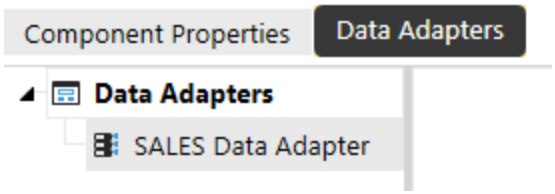
1. Go to **Application** > **Presentation** > **Workspaces** > **[choose Workspace]** > **Maintenance Unit** > **[choose Maintenance Unit]** > **Components** > **Grid View**.
2. Create Dashboard Component or open a grid view.



3. Choose Grid View and select OK.



4. Configure the grid to use the data adapter.



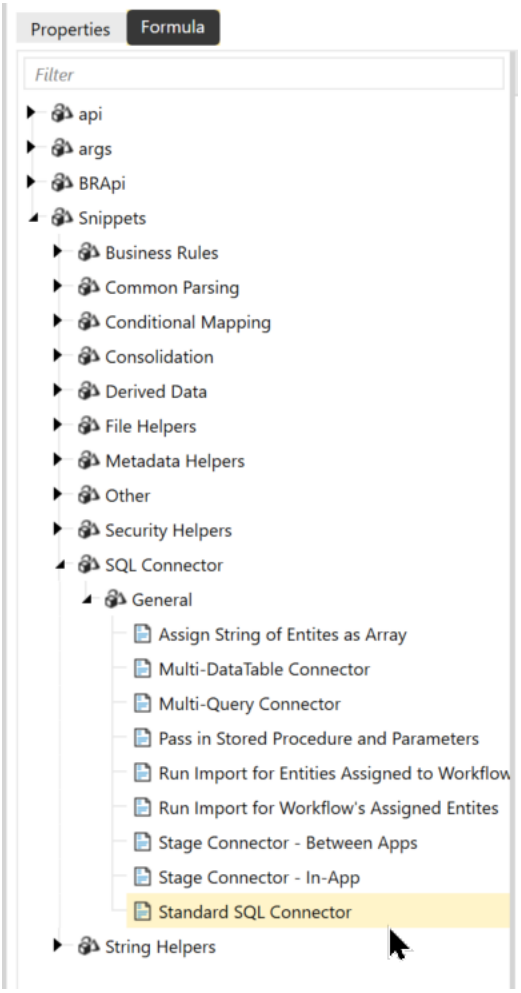
5. Run the associated dashboard to see the data.

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 Solution Exchange.

1. Download and install the Snippet Editor from Solution Exchange.
2. Navigate to **Application** > **Tools** > **Business Rules**.
3. Expand **Connector** and select a **Business Rule**.

4. Navigate to **Snippets** > **SQL Connector** > **Standard SQL Connectors**.



5. Copy the Sample Business Rule.

6. 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 GetConnectionString_Gateway(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
```

7. Enter the drill back information to your database.

```
If args.DrillCode.Equals(StageConstants.TransformationGeneral.DrillCodeDefaultValue,
StringComparison.InvariantCulturelgnoreCase) 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)")))
```

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

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 SQL remote database. Other types of databases (ODBC and OLEDB) are not compatible with this example.

```
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 = "Northeast_HQ"

                ' SIC remote rule
                Dim sicRemoteRule As String = "update_DB"
```

```
                ' 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)

                Dim dtObj(2) As Object ' Create object to store arguments for remote
business rule
                dtObj(0) = dt ' 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 = bulkRemoteResults.resultData

                BRApi.ErrorLog.LogMessage(si,returnedMsg)

                Return Nothing
            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(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(dt as DataTable,tablename As String,
        dataSource As String) As String
            ' ------------------------------------------------------------------------------
----------------
            ' Get SQL connection string
            Dim connString As String = APILibrary.GetRemoteDataSourceConnection(dataSource)

            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

            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.

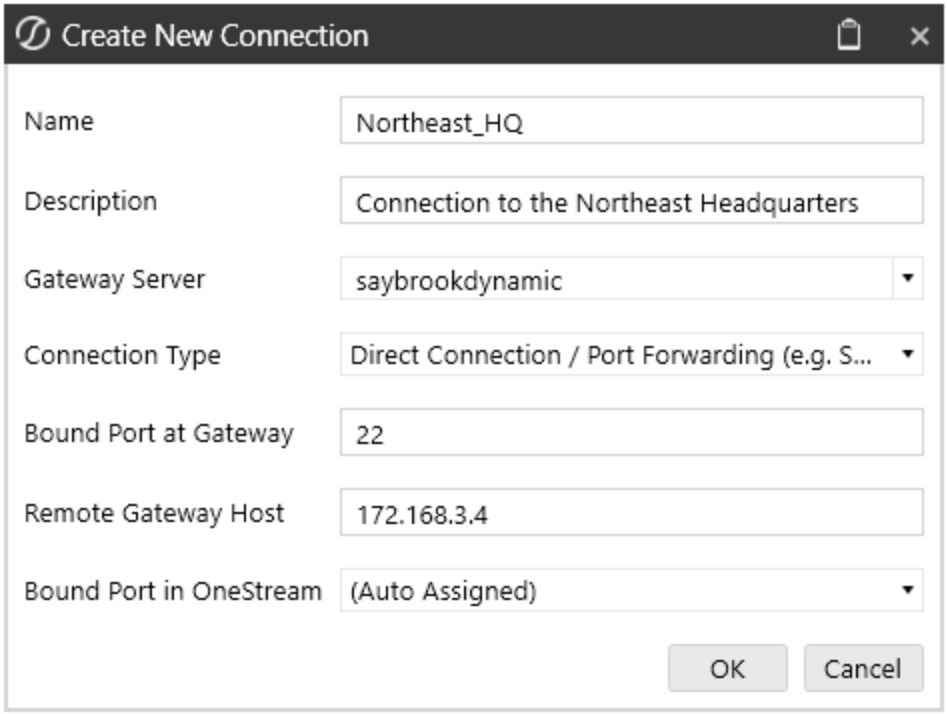
> **IMPORTANT:** It is best practice to utilize SSH.NET, which is included with Smart Integration Connector, for Secure File Transfer Protocol (SFTP) tasks.

> **IMPORTANT:** As of version 9.1, WinSCP is no longer included with Smart Integration Connector. For current WinSCP users, it is a recommended best practice to transition your SFTP operations to the SSH.NET library. If you want to continue to use WinSCP, you will need to add WinSCP to your referenced assemblies folder and reference WinSCP from your remote business rules. See Support for DLL Migration.

> **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. Log in to OneStream.
2. Navigate to **System** > **Administration** > **Smart Integration Connector**.
3. Create a New **Connection** and fill out all of the corresponding details for your Connection and the Gateway Server.
4. From **Connection Type**, select **Direct Connection / Port Forwarding**.
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.



7. For **Bound Port in OneStream**, select **(Auto Assigned)** (default and recommended setting) or **Enter Port Manually**. See Installation and Setup for additional information.
    - **(Auto Assigned)** (default and recommended setting) to allow the OneStream application to automatically assign an unused port number. When the Direct Connection is created, the port number is shown in the connection settings.
    - **Enter Port Manually**: Enter an unused port number. The port number must be greater than 1024 and less than 65535.
8. Click **OK**.
9. Copy the **Connection** 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.

C# SFTP Example

Below you can find the C# example for STFP.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using Microsoft.CSharp;
using OneStream.Finance.Database;
using OneStream.Finance.Engine;
using OneStream.Shared.Common;
using OneStream.Shared.Database;
using OneStream.Shared.Engine;
using OneStream.Shared.Wcf;
using OneStream.Stage.Database;
using OneStream.Stage.Engine;
using Renci.SshNet;
namespace OneStream.BusinessRule.Extender.SFTP_SSH_C
{
    public class MainClass
    {
        public object Main(SessionInfo si, BRGlobals globals, object api, ExtenderArgs args)
        {
            try
            {

                // ------------------------------------------------
                // SSH.NET EXAMPLES
                // ------------------------------------------------

                // Setup SSH.NET session options from values in Cloud Administration Tools
(CAT) Key Management - Secrets
                var username = BRApi.Utilities.GetSecretValue(si, "SFTP-UserName");
                var password = BRApi.Utilities.GetSecretValue(si, "SFTP-Password");
                var authenticationMethod = new PasswordAuthenticationMethod(username,
password);
                var connectionInfo = new ConnectionInfo("localhost", username,
authenticationMethod);
```

```
                // Get the filepath - BatchHarvest in this example is File Share /
Applications / GolfStreamDemo_v36 / Batch / Harvest
                var fileDNpath = BRApi.Utilities.GetFileShareFolder(si,
FileShareFolderTypes.BatchHarvest, null);
                var fileSFTPpath = Path.Combine(fileDNpath, "SFTP_TEST_DOWNLOAD_" +
DateTime.UtcNow.ToString("MM-dd-yyyy-HHmmss") + ".txt");
                var fileSCPpath = Path.Combine(fileDNpath, "SCP_TEST_DOWNLOAD_" +
DateTime.UtcNow.ToString("MM-dd-yyyy-HHmmss") + ".txt");
                // SFTP Example
                using (var sftpClient = new SftpClient(connectionInfo))
     {
         sftpClient.Connect();
                    using (var downloadStream = new FileStream(fileSFTPpath,
FileMode.OpenOrCreate, FileAccess.Write, FileShare.None))
                    {
     sftpClient.DownloadFile("SFTP_TEST_DOWNLOAD.txt", downloadStream);
                    }
     }
                // SCP Example
     using (var scpClient = new ScpClient(connectionInfo))
     {
     scpClient.Connect();
     scpClient.Download("SFTP_TEST_DOWNLOAD.txt", new FileInfo(fileSCPpath));
     }
                return null;
            }
            catch (Exception ex)
            {
                throw ErrorHandler.LogWrite(si, new XFException(si, ex));
            }
        }
    }
}
```

VB STFP Example

Below you can find the VB example for STFP.

```
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 Renci.SshNet
Namespace OneStream.BusinessRule.Extender.SFTP_SSH
    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
                ' ------------------------------------------------
                ' SSH.NET EXAMPLES
                ' ------------------------------------------------

                ' Setup SSH.NET session options from values in Cloud Administration Tools
(CAT) Key Management - Secrets
                Dim username As String = BRApi.Utilities.GetSecretValue(si, "SFTP-UserName")
                Dim password As String = BRApi.Utilities.GetSecretValue(si, "SFTP-Password")
                Dim authenticationMethod = New PasswordAuthenticationMethod(username,
password)
                Dim connectionInfo = New ConnectionInfo("localhost", username,
authenticationMethod)

                'Get the filepath - BatchHarvest in this example is File Share /
Applications / GolfStreamDemo_v36 / Batch / Harvest
                Dim fileDNPath As String = BRApi.Utilities.GetFileShareFolder(si,
FileShareFolderTypes.BatchHarvest, Nothing)
                Dim fileSFTPpath = Path.Combine(fileDNpath, "SFTP_TEST_DOWNLOAD_" &
DateTime.UtcNow.ToString("MM-dd-yyyy-HHmmss") & ".txt")
                Dim fileSCPpath = Path.Combine(fileDNpath, "SCP_TEST_DOWNLOAD_" &
DateTime.UtcNow.ToString("MM-dd-yyyy-HHmmss") & ".txt")

                ' SFTP Example
     Using sftpClient = New SftpClient(connectionInfo)
     sftpClient.Connect()
     Using downloadStream = New FileStream(fileSFTPpath, FileMode.OpenOrCreate,
FileAccess.Write, FileShare.None)
     sftpClient.DownloadFile("SFTP_TEST_DOWNLOAD.txt", downloadStream)
     End Using
     End Using

'                ' SCP Example
         Using scpClient As New ScpClient(connectionInfo)
                    scpClient.Connect()
                    scpClient.Download("SFTP_TEST_DOWNLOAD.txt", New FileInfo(fileSCPpath))
         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
```

Transfer 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 DeleteOldFileData(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.

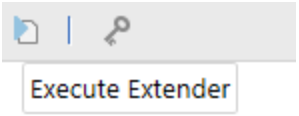
```
Imports System
Imports System.Data
Imports System.Data.Common
Imports System.IO
Imports System.Collections.Generic
Imports System.Globalization
Imports System.Linq
Imports Microsoft.VisualBasic
Imports System.Windows.Forms
Imports OneStream.Shared.Common
Imports OneStream.Shared.Wcf
Imports OneStream.Shared.Engine
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)
                        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,
"entergatewayname", "DeleteOldFileData")
                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,
"entergatewayname", 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,
"entertestconnection", 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.

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.

See Access Multiple WebAPIs for best practices on scenarios with multiple WebAPIs.

> **NOTE:** Data transferred over a Direct Connection to a WebAPI is transferred directly over HTTP(S) and not converted to parquet format. OneStream does not control the return format.

Host Headers

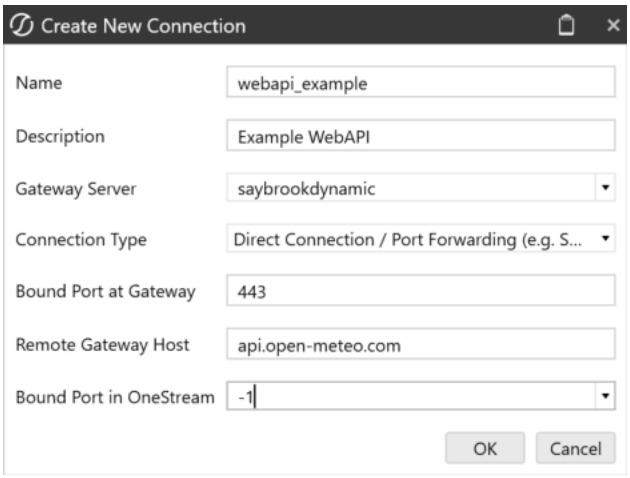
Host headers specify the domain name of the server that will receive the request. The Host header is defined in the Business Rule and includes the domain name of the target server and should match what the server expects in incoming requests (for example, api.example.com). See Troubleshooting for additional information.

```
// The header must be set or some connections may be refused.
internalHttpClient.DefaultRequestHeaders.Host = "api.example.com";
```

Access a Single WebAPI

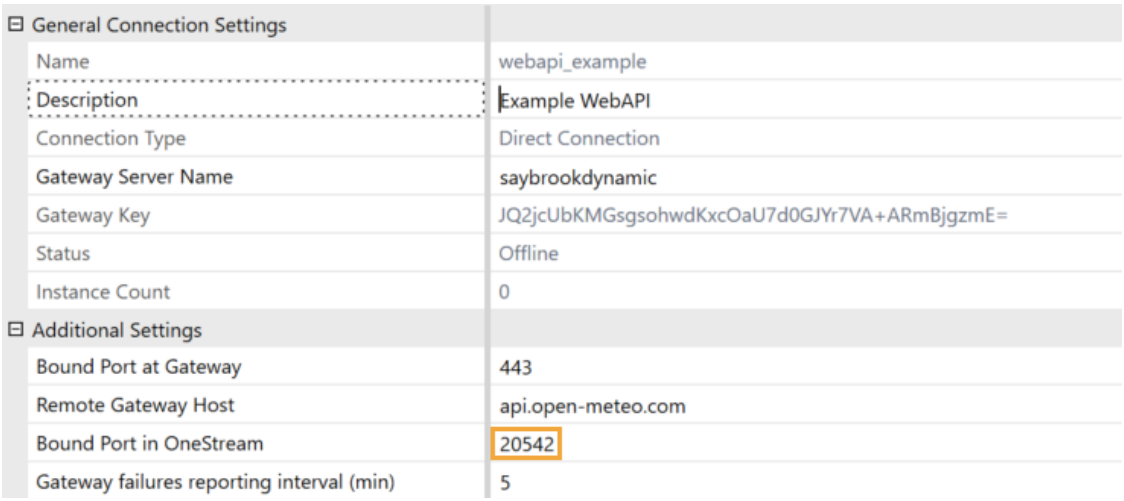
To set up a single WebAPI connection:

1. Set up a Direct Connection.



2. Export the Configuration and import to your Local Gateway Server. See the Installation and Setup section for more information on this process.
3. Refresh your connections and verify this new connection is online.

> **IMPORTANT:** Copy your **Bound Port in OneStream**. You will reference this later in the extensibility rule.



General Connection Settings	
Name	webapi_example
Description	Example WebAPI
Connection Type	Direct Connection
Gateway Server Name	saybrookdynamic
Gateway Key	JQ2jcUbKMGsgsohwdKxcOaU7d0GJYr7VA+ARmBjgzmE=
Status	Offline
Instance Count	0
Additional Settings	
Bound Port at Gateway	443
Remote Gateway Host	api.open-meteo.com
Bound Port in OneStream	20542
Gateway failures reporting interval (min)	5

4. Create the Extensibility Rule below:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using OneStream.Shared.Common;
using OneStream.Shared.Database;
using OneStream.Shared.Engine;
using OneStream.Shared.Wcf;
using System.Net;
using System.Net.Http;
using Newtonsoft.Json;
using System.Net.Http.Headers;

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

        public object Main(SessionInfo si, BRGlobals globals, object api, ExtenderArgs
args)
        {
            try
            {
                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("*/*"));

                // The header must be set or some connections maybe refused.
                internalHttpClient.DefaultRequestHeaders.Host = "api.example.com";

                // In this example, 20540 is the Bound Port in OneStream for the
Gateway being used.
                var stringTask = internalHttpClient.GetStringAsync
("https://localhost:20540/v1/forecast?latitude=40.73&longitude=-
73.94&daily=temperature_2m_max,temperature_2m_min&temperature_
unit=fahrenheit&timezone=America%2FNew_York");

                // Display the result in the exception dialog as an example.
                throw new Exception(stringTask.Result);
            }
            catch (Exception ex)
            {
                throw ErrorHandler.LogWrite(si, new XFException(si, ex));
            }
        }
    }
}
```

5. Compile and test the business rule. If the extensibility ran successfully, you should see the correct data that corresponds with the business rule in the Exception dialog box.

Access Multiple WebAPIs

If you are using more than one WebAPI, the best practice is to perform this process using a single connection and multiple remote Business Rules.

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 xfDT = 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;
 }
 }
}
```

Send Emails through Direct Connections

Prior to using this business rule, you must have your email server configured. You must establish a direct connection before sending email. See Single Web API Connection for more information on setting up an initial direct connection. The following business rule can send email from an Extender Business rule.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using Microsoft.CSharp;
using OneStream.Finance.Database;
using OneStream.Finance.Engine;
using OneStream.Shared.Common;
using OneStream.Shared.Database;
using OneStream.Shared.Engine;
using OneStream.Shared.Wcf;
using OneStream.Stage.Database;
using OneStream.Stage.Engine;
using System.Net.Mail;
using System.Net;
using System.Net.Security;
using System.Text.RegularExpressions;
using System.Security.Cryptography.X509Certificates;
namespace OneStream.BusinessRule.Extender.smtp_direct_test
{
    public class MainClass
    {
        public SessionInfo SI;
        private const string smtpHostName = "smtp.azurecomm.net"; // expected name to match
the cert.


        public object Main(SessionInfo si, BRGlobals globals, object api, ExtenderArgs args)
        {
 var client = new SmtpClient();
 var email = new MailMessage();
            try
            {
                SI = si;
                // Add custom validation callback to look for expected cert (Host will be
localhost, which causes this to fail without a custom callback)
                ServicePointManager.ServerCertificateValidationCallback +=
ValidationCallback;
```

```
      client.UseDefaultCredentials = false;
      client.Port = 20542;
      client.Host = "localhost";
                client.EnableSsl = true;
                client.Credentials = new System.Net.NetworkCredential("<UserName>",
"<Password>");

      email.From = new MailAddress("DoNotReply@domain.com");
      email.To.Add("test@onestreamsoftware.com");
      email.Subject = "Test from SIC Gateway";
      email.IsBodyHtml = false;
      email.Body = "Forwarded test from SIC Gateway";

      client.Send(email);

                return null;
            }
            catch (Exception ex)
            {
                throw ErrorHandler.LogWrite(si, new XFException(si, ex));
            }
            finally
            {
                // Remove the custom ValidationCallback. It's recommended to remove this
before any other network calls.
                ServicePointManager.ServerCertificateValidationCallback -=
ValidationCallback;
                email.Dispose();
                client.Dispose();
            }
        }

        public bool ValidationCallback(object sender, X509Certificate certificate, X509Chain
chain, SslPolicyErrors sslPolicyErrors)
        {
            var policyErrors = (sslPolicyErrors as SslPolicyErrors?) ??
SslPolicyErrors.None;
            var certSubject = certificate?.Subject ?? string.Empty;
            var certName = string.Empty;

            // Extract the certName from the certSubject
            string namePattern = @"CN=([^,]+)";
            var match = Regex.Match(certSubject, namePattern);
            if (match.Success)
            {
                certName = match.Groups[1].Value;

            }
            if (
                (policyErrors == SslPolicyErrors.RemoteCertificateNameMismatch ||
policyErrors == SslPolicyErrors.None)
                && certName == smtpHostName)
            {
                // verify the certName matches the expected smtpHostName. No other
SslPolicyErrors should be present.
                return true;
```

```
                }
                else
                {
                    return false;
                }
            }
        }
    }
```

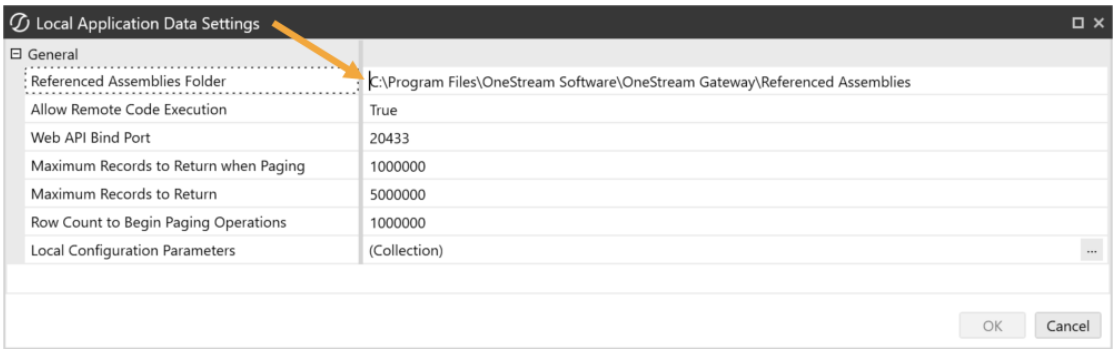
Support for DLL Migration

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 Additional Settings for more information on these fields.

SAP Connections

Establish a connection to an SAP environment using the OneStream SAP Connector. The best practice is to use the SAP Connector when connecting to an SAP environment. If necessary, the legacy connection method that uses the third-party ERPConnect DLL can be used.

Connect with the SAP Connector (Best Practice)

The SAP Connector is available from the OneStream Solution Exchange. To connect with the SAP Connector, refer to the SAP Connector Guide.

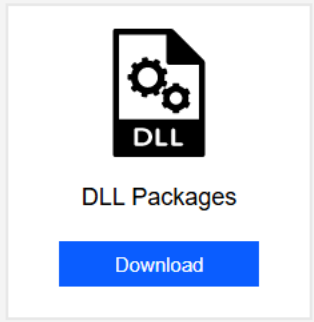
Connect with the ERPConnect (SAP)

As an alternative to using the SAP Connector, 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. 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.

For additional information, see the Theobald Software ERPConnect Help Center .

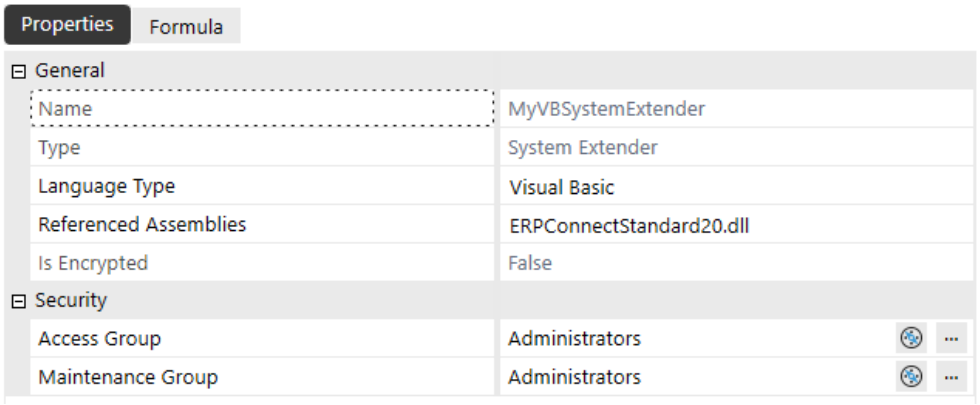
To get started:

1. From the Platform page of the Solution Exchange, download the DLL Packages, which contains the ERPConnectStandard20.dll file.



2. Extract the compressed zip file and then move the ERPConnectStandard20.dll to your Referenced Assemblies Folder.
3. Install the required Visual C++ Redistributable latest supported downloads.
4. Login to your sap.com account and then download SAP NetWeaver RFC Library DLL (sapnwrfc.dll) and associated icudt57.dll, icuin57.dll, icuuc57.dll files.
    - Copy SAP NetWeaver RFC Library DLL (sapnwrfc.dll) to the Referenced Assemblies folder.
    - Copy icudt57.dll, icuin57.dll, and icuuc57.dll to C:\Windows\System32.
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.



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**.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using ERPConnect;
using ERPConnect.Utils;
namespace OneStream.BusinessRule.SmartIntegrationFunction.ERP_Connect_Test
{
  public class MainClass
  {
    public const string UserName = "";
    public const string Password = "";
    public const string Host = "";
    public DataTable RunOperation()
    {
      using (R3Connection con = new R3Connection())
      {
   con.UserName = UserName;
   con.Password = Password;
   con.Language = "EN";
   con.Client = "800";
   con.Host = Host;
   con.SystemNumber = 00;
   con.Protocol = ClientProtocol.NWRFC; // Optional: If the NW RFC libraries are used.
   con.UsesLoadBalancing = false;
   con.Open();
   ReadTable table = new ReadTable(con);
   table.AddField("MATNR");
   table.AddField("MAKTX");
   table.WhereClause = "SPRAS = 'EN' AND MATNR LIKE '%23'";
   table.TableName = "MAKT";
   table.RowCount = 10;
   table.Run();
   return table.Result;
      }
    }
  }
}
```

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

Business Rules

The Smart Integration Connector Capabilities introduce additional business rule APIs (BRAPIs) 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 BRAPIs 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 BRAPIs are outlined below:

ExecRemoteGatewayBusinessRule
ExecRemoteGatewayCachedBusinessRule

ExecRemoteGatewayRequest
ExecRemoteGatewayJobAndWait
ExecRemoteGatewayJob
GetRemoteGatewayJobStatus
IsRemoteDtoSuccessful
GetGatewayConnectionInfo
GetSmartIntegrationConfigValue
GetRemoteDataSourceConnection
BRApi.Utilities.IsGatewayOnline
Check OneStream Version
Business Rules Compatibility

ExecRemoteGatewayBusinessRule

This is a core BRAPI 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 BRAPI 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 BRAPI, it is expected that remote calls should take no more than 2-3 minutes to return a result to the caller as this BRAPI will block until a result is returned. If longer running or sync operations are needed, consider using the ExecRemoteGatewayJobAndWait BRAPI.

> **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:

```
// ExecRemoteGatewayBusinessRule displaying results in drillback
var GatewayName = ""; // Name of the Gateway
var SICFunctionName = ""; // Name of the SIC Function to run
DrillBackResultInfo drillBackInfo = new DrillBackResultInfo();
DataTable dtf = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, SICFunctionName, null,
GatewayName, string.Empty).ResultSet;
var xfDT = new XFDataTable(si, dtf, null, 1000);
drillBackInfo.DataTable = xfDT;
drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid;
return drillBackInfo;
```

Here is a VB example:

```
' ExecRemoteGatewayBusinessRule displaying results in drillback
Dim GatewayName As String = "" ' Name of the Gateway
Dim SICFunctionName As String = "" ' Name of the SIC Function to run
Dim drillBackInfo As DrillBackResultInfo = new DrillBackResultInfo()
Dim dtf As DataTable = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, SICFunctionName,
Nothing, GatewayName, String.Empty).ResultSet
Dim xfDT As XFDataTable = new XFDataTable(si, dtf, Nothing, 1000)
drillBackInfo.DataTable = xfDT
drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid
Return drillBackInfo
```

Here is a C# drill-back example that invokes a remote business rule accepting 2 parameters:

```
// ExecRemoteGatewayBusinessRule Drillback example
var GatewayName = ""; // Name of the Gateway
var SICFunctionName = ""; // Name of the SIC Function to run
var RemoteMethodName = ""; // Name of the method inside the SIC Function that will be
called.
var drillBackInfo = new DrillBackResultInfo();
object[] argTest = new object[2]; // 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
```

```
var objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si,
SICFunctionName, argTest, GatewayName, RemoteMethodName);

if (objRemoteRequestResultDto.RemoteResultStatus ==
RemoteMessageResultType.RunOperationReturnObject)
{
 var returnVal = objRemoteRequestResultDto.ObjectResultValue as ArrayList;
 // 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.
 var xfDT = new XFDataTable(si, objRemoteRequestResultDto.ResultSet, null, 1000);
 drillBackInfo.DataTable = xfDT;
 drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid;
 return drillBackInfo;
}
else if (!(objRemoteRequestResultDto.RemoteException is null))
{
 throw ErrorHandler.LogWrite(si, new XFException(si,
objRemoteRequestResultDto.RemoteException));
}
```

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

```
' ExecRemoteGatewayBusinessRule Drillback example
Dim GatewayName As String = "" ' Name of the Gateway
Dim SICFunctionName As String = "" ' Name of the SIC Function to run
Dim RemoteMethodName As String = "" ' Name of the method inside the SIC Function that will
be called.
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, SICFunctionName, argTest, GatewayName,
RemoteMethodName)

If (objRemoteRequestResultDto.RemoteResultStatus =
RemoteMessageResultType.RunOperationReturnObject) Then
```

```
 Dim returnVal As ArrayList = 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 XFDataTable(si, objRemoteRequestResultDto.ResultSet, Nothing, 1000)
 drillBackInfo.DataTable = xfDT
 drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid
 Return drillBackInfo
Else If (Not (objRemoteRequestResultDto.remoteException Is Nothing))
 Throw ErrorHandler.LogWrite(si, New XFException(si,
objRemoteRequestResultDto.RemoteException))
End If
```

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

Here it is in C#:

```
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 DeleteOldFileData(string year)
 {
 string fname = @"c:\temp\zw_" + year + ".csv";
 try
```

```
 {
 System.IO.File.Delete(fname);
 return true;
 }
 catch (IOException ex)
 {
 return false;
 }
 }
 }
}
```

Here it is in VB:

```
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.TestFileRead
 Public Class MainClass
 Public Function RunOperation(ByVal year As String) As Byte()
 Dim fname As String = "c:\temp\hw_" & year & ".csv"
 Dim buffer As Byte() = System.IO.File.ReadAllBytes(fname)
 Return buffer
 End Function

 Public Function GetOtherFileData(ByVal year As String) As Byte()
 Dim fname As String = "c:\temp\zw_" & year & ".csv"
 Dim buffer As Byte() = System.IO.File.ReadAllBytes(fname)
 Return buffer
 End Function

 Public Function DeleteOldFileData(ByVal year As String) As Boolean
 Dim fname As String = "c:\temp\zw_" & year & ".csv"

 Try
 System.IO.File.Delete(fname)
 Return True
 Catch ex As IOException
 Return False
 End Try
 End Function
 End Class
End Namespace
```

Below is a remote business rule that queries a database and returns a datatable.

Here is the rule in C#:

```
// SIC Function referenced by other examples here
namespace OneStream.BusinessRule.SmartIntegrationFunction.GetDataFromDB
{
 public class MainClass
 {
 private const string DataSourceName = "";
 public DataTable RunOperation()
 {
 DataTable dataTableResults = new DataTable();
 string connectionString, sql;

 connectionString = OneStreamGatewayService.APILibrary.GetRemoteDataSourceConnection
(DataSourceName);

 SqlConnection conn;
 conn = new SqlConnection(connectionStringconn.Open());
 sql = ""; // Enter SQL Query here
 SqlCommand cmd = new SqlCommand(sql, conn);
 var dbreader = cmd.ExecuteReader();
 dataTableResults.Load(dbreader);
 return dataTableResults;
 }
 }
}
```

Here is the rule in VB:

```
' SIC Function referenced by other examples here
Namespace OneStream.BusinessRule.SmartIntegrationFunction.GetDataFromDB
 Public Class MainClass
 Private Const DataSourceName As String = ""

 Public Function RunOperation() As DataTable
 Dim dataTableResults As DataTable = New DataTable()
 Dim connectionString, sql As String
 connectionString = APILibrary.GetRemoteDataSourceConnection(DataSourceName)
 Dim conn As SqlConnection
 conn = New SqlConnection(connectionStringconn.Open())
 sql = "" ' Enter SQL Query here
 Dim cmd As SqlCommand = New SqlCommand(sql, conn)
 Dim dbreader = cmd.ExecuteReader()
 dataTableResults.Load(dbreader)
 Return dataTableResults
 End Function
 End Class
End Namespace
```

Here is an example of calling a TestFileRead remote business rule in C#.

```
// Here we are telling it to specifically call a remote Smart Integration Function called
TestFileRead at SIC Gateway
// called TestConnection with a method called DeleteOldFileData
var GatewayName = ""; // Name of the Gateway
var SICFunctionName = "TestFileRead"; // Name of the SIC Function from above example
var RemoteMethodName = "DeleteOldFileData"; // Name of the method inside the SIC Function
that will be called.
RemoteRequestResultDto objRemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, SICFunctionName, new object[] {"2024"},
GatewayName, RemoteMethodName);

if (objRemoteRequestResultDto.RemoteResultStatus ==
RemoteMessageResultType.RunOperationReturnObject &&
(objRemoteRequestResultDto.ObjectResultValue is 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;
```

Here is an example of calling a TestFileRead remote business rule in VB.NET.

```
'Here we are telling it to specifically call a remote Smart Integration Function called
TestFileRead at SIC Gateway
'called TestConnection with a method called DeleteOldFileData
Dim GatewayName As String = "" ' Name of the Gateway
Dim SICFunctionName As String = "TestFileRead" ' Name of the SIC Function from above example
Dim RemoteMethodName As String = "DeleteOldFileData" ' Name of the method inside the SIC
Function that will be called.
Dim argTest(0) As Object ' Creating an object array to package the method parameters
argTest(0) = "2024" ' First parameter is an integer

Dim objRemoteRequestResultDto As RemoteRequestResultDto =
```

```
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, SICFunctionName, argTest, GatewayName,
RemoteMethodName)
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
 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
```

Here's an example to call the remote BR called "GetDataFromDB" (C#):

```
// 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
var GatewayName = ""; // Name of the Gateway
var SICFunctionName = "GetDataFromDB"; // Name of the SIC Function from above example
var RemoteMethodName = "RunOperation"; // Name of the method inside the SIC Function that
will be called.
var objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si,
SICFunctionName, null, GatewayName, RemoteMethodName);

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");
 }
}
```

Here's an example to call the remote BR called "GetDataFromDB" (VB):

```
' 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 GatewayName As String = "" ' Name of the Gateway
Dim SICFunctionName As String = "GetDataFromDB" ' Name of the SIC Function from above
example
Dim RemoteMethodName As String = "RunOperation" ' Name of the method inside the SIC Function
that will be called.
Dim objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si,
SICFunctionName, Nothing, GatewayName, RemoteMethodName)

If objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Success AndAlso
objRemoteRequestResultDto.ResultSet IsNot Nothing AndAlso
objRemoteRequestResultDto.ResultType = RemoteResultType.DataTable Then
 BRApi.ErrorLog.LogMessage(si, "Data Returned - Rows:" &
objRemoteRequestResultDto.ResultSet.Rows.Count)
Else
 If objRemoteRequestResultDto.RemoteException IsNot Nothing Then
 Throw ErrorHandler.LogWrite(si, New XFException(si,
objRemoteRequestResultDto.RemoteException))
 Else
 BRApi.ErrorLog.LogMessage(si, "Remote Smart Integration Function Succeeded - no
data/datatable returned")
 End If
End If
```

ExecRemoteGatewayCachedBusinessRule

When a cache flag and key is provided to the ExecRemoteGatewayBusinessRule BRAPI, 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#:

```
// ExecRemoteGatewayCachedBusinessRule

// Execute and cache a remote SIC Function for later use
var GatewayName = ""; // Name of the Gateway
var SICFunctionName = ""; // Name of the SIC Function to run
var RemoteMethodName = ""; // Name of the method inside the SIC Function that will be called
var SICCachedFunctionName = ""; // Name of the cache key for this SIC Function, which can be
called on subsequent requests
RemoteRequestResultDto objRemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, SICFunctionName, null, GatewayName,
RemoteMethodName, SICCachedFunctionName, false, 90);

if (BRApi.Utilities.IsRemoteDtoSuccessful(si, objRemoteRequestResultDto)
 && 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, SICCachedFunctionName , null,
GatewayName, 90);
```

Here is the rule in VB.NET:

```
' ExecRemoteGatewayCachedBusinessRule

' Execute and cache a remote SIC Function for later use
Dim GatewayName As String = "" ' Name of the Gateway
Dim SICFunctionName As String = "" ' Name of the SIC Function to run
Dim RemoteMethodName As String = "" ' Name of the method inside the SIC Function that will
be called
Dim SICCachedFunctionName As String = "" ' Name of the cache key for this SIC Function,
which can be called on subsequent requests
Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, SICFunctionName, Nothing, GatewayName,
RemoteMethodName, SICCachedFunctionName, False, 90)

If (BRApi.Utilities.IsRemoteDtoSuccessful(si, objRemoteRequestResultDto) AndAlso
objRemoteRequestResultDto.ResultType = RemoteResultType.DataTable) Then
 BRApi.ErrorLog.LogMessage(si, "Data Returned - Rows:" +
objRemoteRequestResultDto.ResultSet.Rows.Count)
Else
 If (objRemoteRequestResultDto.RemoteException IsNot Nothing) Then
 Throw ErrorHandler.LogWrite(si, New XFException(si,
objRemoteRequestResultDto.RemoteException))
 Else
 BRApi.ErrorLog.LogMessage(si, "Remote Smart Integration Function Succeeded - no
data/datatable returned")
 End If
End If

' Subsequent invocations of the remote BR can be run by specifying the endpoint and the
cached key name
Dim objRemoteRequestResultDtoCached As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayCachedBusinessRule(si, SICCachedFunctionName, Nothing,
GatewayName, 90)
```

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.

> **IMPORTANT:** This method is used for request and response type interactions to a remote endpoint that runs for three or less minutes. For jobs that run longer than 180 seconds, it is best practice to use the ExecRemoteGatewayJobAndWait BRAPI.

> **NOTE:** 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:

```
// ExecRemoteGatewayRequest for arbitrary code execution returning a DataTable
string GatewayName = "";
RemoteRequestResultDto objxfRemoteRequestResultDto;
RemoteCodeRequestDto objxfRemoteRequest = 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 = GatewayName;
var strCode = "using System;...."; // Valid block of C# or VB.NET code
objxfRemoteRequest.LanguageType = RemoteCodeLanguageType.CSHARP;
objxfRemoteRequest.RemoteCodeBlock = strCode;
objxfRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayRequest(objxfRemoteRequest);
var xfDT = new XFDataTable(si, objxfRemoteRequestResultDto.ResultSet, null, 1000);
```

Here is the example in VB:

```
' ExecRemoteGatewayRequest for arbitrary code execution returning a DataTable
Dim GatewayName As String = ""
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 = GatewayName
Dim strCode As String = "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 BRAPI can also be used to invoke arbitrary SQL commands against a Smart Integration Connector local gateway connection data source at your site:

```
/ ExecRemoteGatewayRequest for arbitrary SQL returning a DataTable
string SQL = ""; // SQL SELECT statement goes here
RemoteRequestResultDto objxfRemoteRequestResultDto;
RemoteRequestDto objxfRemoteRequest = new RemoteRequestDto();
// Indicate this is a remote SQL command request
objxfRemoteRequest.ConnectionType = RemoteCommandType.SQLCommand;
objxfRemoteRequest.RelayRemoteDBConnection = ""; // Name of the connection defined in the
remote endpoint
objxfRemoteRequest.GatewayHostForRequest = ""; // Name of the remote host to pass to
objxfRemoteRequest.RemoteCommand = SQL;
objxfRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayRequest(objxfRemoteRequest);
// Evaulate the results to determine if it was successful
if (objxfRemoteRequestResultDto.RemoteResultStatus == RemoteMessageResultType.Success)
{
 // Logic to use results in `objxfRemoteRequestResultDto.ResultSet`
}
else
{
 // Query failed. Add additional logic here to handle this case.
}
```

Here is the example in VB:

```
' ExecRemoteGatewayRequest for arbitrary SQL returning a DataTable
Dim SQL As String = "" ' SQL SELECT statement goes here
Dim objxfRemoteRequestResultDto As RemoteRequestResultDto
Dim objxfRemoteRequest As New RemoteRequestDto
```

```
' Indicate this is a remote SQL command request
objxfRemoteRequest.connectionType = RemoteCommandType.SQLCommand
objxfRemoteRequest.RelayRemoteDBConnection = "" ' Name of the connection defined in the
remote endpoint
objxfRemoteRequest.GatewayHostforRequest = "" ' Name of the remote host to pass to
objxfRemoteRequest.RemoteCommand = SQL
objxfRemoteRequestResultDto=BRApi.Utilities.ExecRemoteGatewayRequest(objxfRemoteRequest)
' Evaulate the results to determine if it was successful
If (objxfRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Success) Then
 ' Logic to use results in `objxfRemoteRequestResultDto.ResultSet`
Else
 ' Query failed. Add additional logic here to handle this case.
End If
```

Remote function returning a datatable (C#) without parameters:

```
// ExecRemoteGatewayBusinessRule
var GatewayName = ""; // Name of the Gateway
var SICFunctionName = ""; // Name of the SIC Function to run
var RemoteMethodName = ""; // Name of the method inside the SIC Function that will be called
var objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayBusinessRule(si,
GatewayName, null, SICFunctionName, RemoteMethodName);
if (BRApi.Utilities.IsRemoteDtoSuccessful(si, objRemoteRequestResultDto))
{
    if (objRemoteRequestResultDto.ResultType == RemoteResultType.DataTable)
    {
        BRApi.ErrorLog.LogMessage(si, "Data Returned: " +
objRemoteRequestResultDto.ResultSet.Rows.Count);
    }
}
else
{
    if (!(objRemoteRequestResultDto.RemoteException is null))
    {
        throw ErrorHandler.LogWrite(si, new XFException(si,
objRemoteRequestResultDto.RemoteException));
    }
}
```

Here is the example in VB:

```
' ExecRemoteGatewayBusinessRule

' Call a remote Smart Integration Function
Dim GatewayName As String = "" ' Name of the Gateway
Dim SICFunctionName As String = "" ' Name of the SIC Function to run
Dim RemoteMethodName As String = "" ' Name of the method inside the SIC Function that will
be called
```

```
Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, GatewayName, Nothing, SICFunctionName,
RemoteMethodName)
If (BRApi.Utilities.IsRemoteDtoSuccessful(si, objRemoteRequestResultDto)) 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
```

ExecRemoteGatewayJobAndWait

This BRAPI manages a long-running job process using other business rules. It is the preferred method for users looking to get a response from a remote gateway job without adding boilerplate polling logic to get a result. The BRAPI first performs ExecRemoteGatewayJob to start a new gateway job, then checks the status of that job by calling GetRemoteGatewayJobStatus.

Here is a basic overview of the long-running job process in C#.

```
// ExecRemoteGatewayJobAndWait basic example

var GatewayName = ""; // Name of the Gateway
var SICBRName = ""; // Name of the SIC BR
var SICFunctionName = ""; // Name of the SIC Function to run
var argTest = new object[2];
argTest[0] = 100; // Example first argument to SIC Function
argTest[1] = "test"; // Example second argument to SIC Function
var Timeout = 30; // Timespan in minutes until function will time out. Default is 30 mins.
Max is 4 hours.
var Polling = 60; // Timespan in seconds until function will check for an updated status.

RemoteJobStatusResultDto jobStatusResult = BRApi.Utilities.ExecRemoteGatewayJobAndWait(si,
SICBRName, argTest , GatewayName, SICFunctionName, Timeout , Polling);
if(jobStatusResult.RemoteJobState == RemoteJobState.Completed)
{
      RemoteRequestResultDto result = jobStatusResult.RemoteJobResult;
      // Perform logic on result.
}
```

Here is the basic example in VB:

```
' ExecRemoteGatewayJobAndWait basic example

Dim GatewayName As String = ""
Dim SICBRName As String = ""
Dim SICFunctionName  As String= ""
Dim argTest = New Object(1) {}
argTest(0) = 100
argTest(1) = "test"
Dim Timeout As Integer = 30
Dim Polling As Integer = 60
Dim jobStatusResult As RemoteJobStatusResultDto =
BRApi.Utilities.ExecRemoteGatewayJobAndWait(si, SICBRName, argTest, GatewayName,
SICFunctionName, Timeout, Polling)

If jobStatusResult.RemoteJobState = RemoteJobState.Completed Then
      Dim result As RemoteRequestResultDto = jobStatusResult.RemoteJobResult
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 BRAPI 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 BRAPI), the remote results are purged to ensure that result objects reclaim server memory on the Smart Integration Service host.

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

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

```
// ExecRemoteGatewayJob basic example

var GatewayName = ""; // Name of the Gateway
var SICFunctionName = ""; // Name of the SIC Function to run
var argTest = new object[2];
argTest[0] = 100; // Example first argument to SIC Function
argTest[1] = "test"; // Example second argument to SIC Function

// Invoking a OneStream SIC Function Business Rule as a remote job
var objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayJob(si, SICFunctionName,
argTest, GatewayName, String.Empty);
if (objRemoteRequestResultDto.RemoteResultStatus == RemoteMessageResultType.JobRunning)
{
 // Logic to wait for job to complete
}
```

Here is the basic example in VB:

```
' ExecRemoteGatewayJob basic example

Dim GatewayName As String = "" ' Name of the Gateway
Dim SICFunctionName As String = "" ' Name of the SIC Function to run
Dim argTest(1) As Object
argTest(0) = 100 ' Example first argument to SIC Function
argTest(1) = "test" ' Example second argument to SIC Function
```

```
' Invoking a OneStream SIC Function Business Rule as a remote job
Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayJob(si, SICFunctionName, argTest, GatewayName,
String.Empty)
If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.JobRunning) Then
 ' Logic to wait for job to complete
End If
```

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

```
// ExecRemoteGatewayJob with polling

var jobID = new Guid();
var GatewayName = ""; // Name of the Gateway
var SICFunctionName = ""; // Name of the SIC Function to run

// Invoke a long-running Job with a Smart Integration Function
var objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayJob(si, GatewayName, null,
SICFunctionName, String.Empty);

// If Successful, the status is retuned indicating the job is running with the job ID. Use
this ID to interrogate if the job is compleed.
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 (var loopControl = 0; loopControl < 10; loopControl++)
 {
 System.Threading.Thread.Sleep(2000);
 var objJobStatus = BRApi.Utilities.GetRemoteGatewayJobStatus(si, jobID, GatewayName);

 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 is null))
 {
 var 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 is null))
 {
 var 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.ResultDataCompressed is null))
 {
 BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Object Returned - JobID: " +
jobID.ToString());
 var value = CompressionHelper.InflateJsonObject<String>(si,
objJobStatus.RemoteJobResult.ResultDataCompressed);
 BRApi.ErrorLog.LogMessage(si, value);
 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 Execution of Job: " +
objRemoteRequestResultDto.RemoteException.ToString());
 }
 }
}
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());
 }
}

return null;
```

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

```
' ExecRemoteGatewayJob with polling

Dim jobID As Guid
Dim GatewayName As String = "" ' Name of the Gateway
```

```
Dim SICFunctionName As String = "" ' Name of the SIC Function to run

' Invoke a long-running Job with a Smart Integration Function
Dim objRemoteRequestResultDto As RemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayJob(si, GatewayName, Nothing, SICFunctionName,
String.Empty)

' If Successful, the status is retuned 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, GatewayName)

 If (objJobStatus.RemoteJobState = RemoteJobState.Running) Then
 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 IsNot Nothing) Then
 Dim xfDT As XFDataTable = 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 As XFDataTable = 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 objJobStatus.RemoteJobResult.ResultDataCompressed IsNot Nothing Then
 BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Object Returned - JobID: " &
jobID.ToString())
 Dim value As String = CompressionHelper.InflateJsonObject(Of String)(si,
objJobStatus.RemoteJobResult.ResultDataCompressed)
 Brapi.ErrorLog.LogMessage(si, value)
 Return Nothing
 End If
 Else If (objJobStatus.RemoteJobState = RemoteJobState.JobNotFound) Then
 BRApi.ErrorLog.LogMessage(si, "Remote Job Not Found - JobID: " & jobID.ToString())
 Return Nothing
 Else If (objJobStatus.RemoteJobState = RemoteJobState.RequestTimeOut) Then
 BRApi.ErrorLog.LogMessage(si, "Remote Job Timed Out - JobID: " & jobID.ToString())
 Return Nothing
 Else If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Exception)
Then
 BRApi.ErrorLog.LogMessage(si, "Exception During Exeuction of Job: " &
objRemoteRequestResultDto.RemoteException.ToString())
 End If
 Next
Else
 ' Exception occurred immediately during compile/initial run
 If (objRemoteRequestResultDto.RemoteResultStatus = RemoteMessageResultType.Exception) Then
```

```
 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
```

GetRemoteGatewayJobStatus

This BRAPI 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 extender rule. 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.

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 C#:

```
// Invoke long running job as part of a Data management/Extender rule
public object Main(SessionInfo si, BRGlobals globals, object api, ExtenderArgs args)
```

```
{
 Guid jobID;
 RemoteRequestResultDto objRemoteRequestResultDto = BRApi.Utilities.ExecRemoteGatewayJob(si,
"LongRunningTest", null/* TODO Change to default(_) if this is not a reference type */,
"testConnection", string.Empty);

 if ((objRemoteRequestResultDto.RemoteResultStatus == RemoteMessageResultType.JobRunning))
 {
 jobID = objRemoteRequestResultDto.RequestJobID;
 BRApi.ErrorLog.LogMessage(si, "Remote Job Queued and Running - JobID: " + jobID.ToString
());

 for (var loopControl = 0; loopControl <= 10; loopControl++)
 {
 System.Threading.Thread.Sleep(2000);
 RemoteJobStatusResultDto objJobStatus = 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 (!(objJobStatus.RemoteJobResult.ResultSet == null))
 {
 var 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))
 {
 var xfDT = new XFDataTable(si, objJobStatus.RemoteJobResult.ResultDataSet.Tables[0], null,
1000);
 BRApi.ErrorLog.LogMessage(si, "Remote Job Completed - Dataset Returned - JobID: " +
jobID.ToString());
 }
 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());
 }
 }
 else 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;
}
```

Here is the example in VB:

```
' Invoke long running job as part of a Data management/Extender rule
Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args As ExtenderArgs) As Object
 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.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 Exeuction of Job: " &
objRemoteRequestResultDto.RemoteException.ToString())
 End If
 Next
 Else ' Exception occuring 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
End Function
```

IsRemoteDtoSuccessful

Use the IsRemoteDtoSuccessful method to integrate debugging into business rules. It validates a successful request is received prior to further processing.

IsRemoteDtoSuccessful method:

```
BRApi.Utilities.IsRemoteDtoSuccessful(SessionInfo, RemoteRequestResultDto)
```

Parameters:

- si: SessionInfo object used to create connection objects
- RemoteRequestResultDto - Result of execution including the status and any exceptions which may have occurred on the remote endpoint
- Return: RemoteRequestResultDto - Returns True or False.

```
 public class MainClass
 {
  public object Main(SessionInfo si, BRGlobals globals, object api, ExtenderArgs args)
  {
   var GatewayName = "apgate"; // Name of the Gateway
   var RemoteMethodName = "RunOperation"; // Name of the method inside the SIC Function that
will be called.
   //DataTable non null dto and remote exception
   //resultdtovaluetype is not null and there are remote exception
   RemoteRequestResultDto objRemoteRequestResultDto0 =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnNonNullRemoteEx", null,
GatewayName, RemoteMethodName);
   try
   {
    bool ret = false;
    ret = BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDto0);
   }
   catch (Exception ex)
   {
    BRApi.ErrorLog.LogMessage(si, "ResultDto execption: "
+  ex.InnerException.Message.ToString());
   }
   //DataTable
   //Failed - resultdtovaluetype is null and no remote exceptions
   RemoteRequestResultDto objRemoteRequestResultDto =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnDataTable", new object[]
{true}, GatewayName, RemoteMethodName);
   BRApi.ErrorLog.LogMessage(si,"IsRemoteDtoSuccess.DataTable False = " +
BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDto).ToString());
   //Success - resultdtovaluetype is not null and no remote exceptions
   RemoteRequestResultDto objRemoteRequestResultDtoA =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnDataTable", new object[]
{false}, GatewayName, RemoteMethodName);
   BRApi.ErrorLog.LogMessage(si,"IsRemoteDtoSuccess.DataTable True = " +
BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDtoA).ToString());
   //DataSet
   //Failed - resultdtovaluetype is null and no remote exceptions
   RemoteRequestResultDto objRemoteRequestResultDto1 =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnDataset", new object[] {true},
GatewayName, RemoteMethodName);
   BRApi.ErrorLog.LogMessage(si,"IsRemoteDtoSuccess.Dataset False = " +
BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDto1).ToString());
   //Success - resultdtovaluetype is not null and no remote exceptions
   RemoteRequestResultDto objRemoteRequestResultDto1A =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnDataset", new object[] {false},
GatewayName, RemoteMethodName);
   BRApi.ErrorLog.LogMessage(si,"IsRemoteDtoSuccess.Dataset True = " +
BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDto1A).ToString());
   //Value
   //Failed - resultdtovaluetype is null and no remote exceptions
   RemoteRequestResultDto objRemoteRequestResultDto2 =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnValue", new object[] {true},
GatewayName, RemoteMethodName);
   BRApi.ErrorLog.LogMessage(si,"IsRemoteDtoSuccess.ObjectValue False = " +
BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDto2).ToString());
   //Success - resultdtovaluetype is not null and no remote exceptions
   RemoteRequestResultDto objRemoteRequestResultDto2A =
```

```
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnValue", new object[] {false},
GatewayName, RemoteMethodName);
   BRApi.ErrorLog.LogMessage(si,"IsRemoteDtoSuccess.ObjectValue True = " +
BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDto2A).ToString());
   //Exception
   //Failed - resultdtovaluetype is null and there are remote exceptions
   try
   {
    RemoteRequestResultDto objRemoteRequestResultDto3 =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "TestReturnException", null, GatewayName,
RemoteMethodName);
    BRApi.Utilities.IsRemoteDtoSuccessful(si,objRemoteRequestResultDto3);
   }
   catch (Exception ex)
   {
    BRApi.ErrorLog.LogMessage(si, ex.InnerException.Message.ToString());
   }
   return null;
  }
 }
```

GetGatewayConnectionInfo

From a OneStream business rule, you can invoke this API to obtain connection 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= Gateway 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#:

```
// GetGatewayConnectionInfo
var GatewayName = "" //Name of the Gateway
GatewayDetails gatewayDetailInformation = BRApi.Utilities.GetGatewayConnectionInfo(si,
GatewayName);
int oneStreamPortNumber = gatewayDetailInformation.OneStreamPortNumber;
```

Here is the rule in VB:

```
' GetGatewayConnectionInfo
Dim GatewayName As String = "" ' Name of the Gateway
Dim objGatewayDetails As GatewayDetails = BRApi.Utilities.GetGatewayConnectionInfo(si,
GatewayName)
Dim oneStreamPortNumber As Integer = objGatewayDetails.OneStreamPortNumber
```

GetSmartIntegrationConfigValue

This BRAPI 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 BRAPI 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#:

```
// SIC Function demonstrating GetSmartIntegrationConfigValue
namespace TestProject.OneStream.BusinessRule.SmartIntegrationFunction.SecretTester
{
 public class MainClass
 {
 public static @bool RunOperation()
 {
 string result;
 // 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(""); //Enter config value name
 return true;
 }
 }
}
```

Here is another example in VB.NET:

```
' SIC Function demonstrating GetSmartIntegrationConfigValue

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("") ' Enter config value name
  Return True
  End Function
  End Class
End NameSpace
```

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 C#:

```
// SIC Function to get configured connection string from SIC Gateway
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
("",); // enter name of DB Connection
 SqlConnection conn = new SqlConnection(connectionString);
            // Insert custom code
 return dataTableResults;
 }
 }
}
```

Here is the rule in VB.NET:

```
' SIC Function to get configured connection string from SIC Gateway
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("") ' enter name of DB
Connection
 Dim conn As SqlConnection = New SqlConnection(connectionString)
 ' Insert custom code

 Return dataTableResults
 End Function
 End Class
End Namespace
```

BRApi.Utilities.IsGatewayOnline

The following business rule can check the status of Smart Integration Connector. You will need to replace "gateway-name" with the name of the gateway to be tested.

Here is the rule in C#:

```
// IsGatewayOnline

namespace OneStream.BusinessRule.Extender.TestHealthCheck
{
 public class MainClass
 {
 public const string GatewayName = "";

 public object Main(SessionInfo si, BRGlobals globals, object api, ExtenderArgs args)
 {
 try
 {
 TestGatewayConnection(si, GatewayName);
 return null;
 }
 catch (Exception ex)
 {
 throw ErrorHandler.LogWrite(si, new XFException(si, ex));
 }
```

```
 }

 public void TestGatewayConnection(SessionInfo si, string gwName)
 {
 bool response = BRApi.Utilities.IsGatewayOnline(gwName);

 if (response)
 {
 BRApi.ErrorLog.LogMessage(si, $"Health Check Successful for {gwName}");
 }
 else
 {
 BRApi.ErrorLog.LogMessage(si, $"Health Check Failed for {gwName}");
 }
 }
 }
}
```

Here is the rule in VB:

```
Namespace OneStream.BusinessRule.Extender.TestHealthCheck
 Public Class MainClass
 Public Const GatewayName As String = ""

 Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args As ExtenderArgs) As Object
 Try
 TestGatewayConnection(si, GatewayName)
 Return Nothing
 Catch ex As Exception
 Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
 End Try
 End Function

 Public Sub TestGatewayConnection(ByVal si As SessionInfo, ByVal gwName As String)
 Dim response As Boolean = BRApi.Utilities.IsGatewayOnline(gwName)

 If response Then
 BRApi.ErrorLog.LogMessage(si, $"Health Check Successful for {gwName}")
 Else
 BRApi.ErrorLog.LogMessage(si, $"Health Check Failed for {gwName}")
 End If
 End Sub
 End Class
End Namespace
```

Check OneStream Version

Remote business rules have the ability to provide logic based on the OneStream Version.

Here is the example rule in C#:

```
namespace OneStream.BusinessRule.SmartIntegrationFunction.version_test_csharp
{
    public class MainClass
    {
        public string RunOperation()
        {
            #if ONESTREAM8_4_0_OR_GREATER
                // Code if true
            #else
                // Code if false
            #endif
        }
    }
}
```

Here is the example rule in VB:

```
Namespace OneStream.BusinessRule.SmartIntegrationFunction.version_test_vb
    Public Class MainClass
        Public Shared Function RunOperation() As String
            #If ONESTREAM8_4_0_OR_GREATER
                ' Code if true
            #Else
                ' Code if false
            #End If
        End Function
    End Class
End Namespace
```

Business Rules Compatibility

There are some business rules that are not compatible with Smart Integration Connector. If you attempt certain rules, you will run into the following error: This BRAPI is not compatible with Smart Integration Connector. Refer to Smart Integration Connector Remote BRs.

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

BRApi.Database.SaveCustomDataTable

Although, this business rule is not supported, the functionality can be achieved through a remote business rule. You can call this business rule using BRApi.Utilities.ExecRemoteGatewayBusinessRule.

Here is the rule in C#:

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

namespace OneStream.BusinessRule.SmartIntegrationFunction.SaveCustomDataTable
{
    public class MainClass
    {
        public void RunOperation()
        {
            var tableName = "";          // Enter the name of the table to update
            var connectionName = "";     // Enter the name of the configured database
connection
            var connString =
OneStreamGatewayService.APILibrary.GetRemoteDataSourceConnection(connectionName);
            var dataTable = new DataTable();
            using (var connection = new SqlConnection(connString))
            {
                connection.Open();
                var sql = $"SELECT * FROM {tableName}";
                var cmd = new SqlCommand(sql, connection);
                var adapter = new SqlDataAdapter();
```

```
                adapter.SelectCommand = cmd;
                var commandBuilder = new SqlCommandBuilder(adapter);
                adapter.Fill(dataTable);
                // Add logic here to update values in DataTable
                // Update database with changes to the DataTable
                adapter.UpdateCommand = commandBuilder.GetUpdateCommand();
                adapter.Update(dataTable);
            }
        }
    }
}
```

Here is the same rule VB:

```
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.SaveCustomDataTableVB
    Public Class MainClass
        Public Sub RunOperation()
         Dim tableName = ""            ' Enter the name of the table to update
         Dim connectionName = ""        ' Enter the name of the configured database
connection
         Dim connString = OneStreamGatewayService.APILibrary.GetRemoteDataSourceConnection
(connectionName)
         Dim dataTable = New DataTable()
         Using connection = New SqlConnection(connString)
         connection.Open()
         Dim sql = $"SELECT * FROM {tableName}"
         Dim cmd = New SqlCommand(sql, connection)
         Dim adapter = New SqlDataAdapter()
         adapter.SelectCommand = cmd
         Dim commandBuilder = New SqlCommandBuilder(adapter)
         adapter.Fill(dataTable)
                ' Add logic here to update values in DataTable
                ' Update database with changes to the DataTable
         adapter.UpdateCommand = commandBuilder.GetUpdateCommand()
         adapter.Update(dataTable)
         End Using
        End Sub
    End Class
End Namespace
```

BRApi.Database.InsertOrUpdateRow

BRApi.Database.DeleteRows

Although, these business rules are not supported, inserting, deleting and updating rows can be accomplished through the same remote business rule referenced in BRApi.Database.SaveCustomDataTable. You can call this business rule using BRApi.Utilities.ExecRemoteGatewayBusinessRule. You will insert your logic at the specific comment in the remote business rule.

BRApi.Database.ExecuteActionQuery

ExecuteActionQuery does not support returning a scalar when used as a remote business rule. A sample workaround is provided below:

```
Else If args.DrillBackType.NameAndDescription.Name.Equals("ScalarQuery",
StringComparison.InvariantCultureIgnoreCase) Then

                        Dim intResult As Int32 = 0
                        brapi.ErrorLog.Logmessage(si, "Entered GetData")
                        Dim sqlExecStatement As String = "Select count(*) FROM
InvoiceMaterialDetail"
                        brapi.ErrorLog.LogMessage(si, "statement = " & sqlExecStatement)
                            Using dbAXConn As DbConnInfo =
BRApi.Database.CreateRelayDbConnInfo(si, "jl_SQL_Gateway_Connection")
                                Using sicConnection As DbConnection = dbAXConn.GetConn()
                                    Dim sproccmd = sicConnection.CreateCommand()
                                    sproccmd.commandtext = sqlExecStatement
                                    intResult = sprocCmd.ExecuteScalar  ' Should be non-zero
                                    sicConnection.Close()
                                End Using
                                dbAXConn.Close()
                            End Using

                        Dim drillBackInfo As New DrillBackResultInfo
                        drillbackinfo.TextMessage = intResult
                        drillBackInfo.DisplayType =
ConnectorDrillBackDisplayTypes.TextMessage
                        Return drillBackInfo
```

SQL Bulk Copy

Use of the SQL Bulk Copy class is not supported to copy to and from databases accessed over Smart Integration Connector. Currently, there is not a workaround available.

SQL Transactions

Use of the SqlTransaction class is currently only supported in Smart Integration Connector Functions / remote business rules.

Limitations

This section details a list of known limitations in Smart Integration Connector:

Supported DLLs

If referencing custom DLLs with Smart Integration Connector in your application, .NET Framework and .NET Standard DLLs are supported. These DLLs will need to be copied to the Referenced Assemblies Folder that is defined within the Smart Integration Connector Utility.

Business Rule Compatibility

The Smart Integration Connector supports business rule APIs (BRAPIs) to allow for execution and management of remote business rules inside the context of the Smart Integration Connector Local Gateway Server. SomeOneStream Business Rules are not supported. For compatible Business Rules, see Business Rules.

.NET TypeCodes

The following .NET TypeCodes are supported in Smart Integration Connector:

- Int32
- Int16
- Decimal
- String
- Boolean
- DateTime
- Double
- Int64
- Single (float)
- Byte
- SByte
- UInt16
- UInt32

- UInt64
- Guid
- byte[]

Parquet Format Transfer

Smart Integration Connector transfers data in Apache Parquet format from the Local Gateway Service to your OneStream cloud instance. If you are transferring a data type that is not fully supported by parquet, the data returns as a string. If the data type can not be converted to parquet, you may have to cast the data type in your query.

Example for datetimeoffset:

- SELECT CAST(your_datetimeoffset_column AS VARCHAR(50)) AS formatted_datetime FROM your_table;

For additional information, see Troubleshooting.

Custom Email Connections

Email over Smart Integration Custom ("Notification Connection" in Data Management jobs) Connections is not supported. Remote BRs do support email in Smart Integration Connector.

FTP Transfers

SFTP is supported by the use of SSH.NET. FTP is currently not supported for SSH.NET. Use SFTP for all file transfers.

Internal Certificate Trust

Certificates issued by an internal domain controller cannot be trusted by OneStream.

SQL Table Editor

(missing snippet link)

Precision using Decimals

When transmitting data through Smart Integration Connector, numeric values exceeding certain lengths may be rounded. The following workarounds can help maintain data precision.

Receiving Data from Smart Integration Connector

Smart Integration Connector queries can only return numeric values with up to 38 total digits: 20 integer digits to left of the decimal point and 18 fractional digits to the right of the decimal point.

For example, returning a column with a value of 123456789123456789123 (21 digits) is not supported. Even though there is no decimal point, it still exceeds 20 integer digits, which is the maximum amount.

Similarly, returning a column with a value of 0.1234567891234567891 (19 decimal digits) is not supported, as it contains more than 18 digits on the right side of the decimal point.

If your queries can return values that require more than 20 integer digits or 18 fractional digits, consider casting to a VARCHAR as the following:

- `"SELECT CAST(123456789123456789123 AS VARCHAR)" -- 21 integer digits`

If there is no risk of overflowing the opposite side of the decimal point, you can also divide by a factor of 10 to shift right or multiply by a factor of 10 to shift left. This approach is more efficient than casting to a VARCHAR

For example:

- `SELECT 123456789123456789123 / 100 -- 21 integer digits will shift by two digits to the right`
- `SELECT 0.1234567891234567890 * 100 -- 19 fractional digits will shift by two digits to the left`

Sending Data to Smart Integration Connector

For sending either a DataTable or a CompressionResult into a remote rule, Smart Integration Connector can only return numeric values with up to 18 characters of significance. Values with more than 18 significant digits will lose precision.

For example, when sending a DataTable into Smart Integration Connector, a value of 1234567890123456789.123456 (25 significant digits) will become 1234567890123456800 (17 significant digits).

Similarly, sending a CompressionResult with a value of 123456789012345.123456 (21 significant digits) will become 123456789012345.13 (17 significant digits).

> **NOTE:** This precision limit is a ceiling, not a typical round.

If you are confident that the data will not reach 18 significant digits, no action is needed. If you anticipate that the data sent into Smart Integration Connector will reach this limit, consider using this Business Rule Extender:

```
namespace OneStream.BusinessRule.Extender.SIC_PrecisionDemo_Extender
{
```

```
            public class MainClass
            {
                        public const string remoteGatewayName = "";          // Enter
gateway name
                        const decimal OriginalValue = 1234567890123456.12M; // 18 total
digits


                        public object Main(SessionInfo si, BRGlobals globals, object api,
ExtenderArgs args)
                        {
                                    // dataTable with a single row containing OriginalValue
above
                                    var dataTable = new System.Data.DataTable
("TestDataTable");
            dataTable.Columns.Add("DecimalColumn", typeof(decimal));
            // Add the original value to the datatable
            var row = dataTable.NewRow();
            row["DecimalColumn"] = OriginalValue;
            dataTable.Rows.Add(row);

                                    // base64-encoded string version of dataTable
                                    var base64 = GetBase64EncodedDataTable(dataTable);

                                    // pass both the raw dataTable and encoded version as
two separate arguments to the Remote Business Rule
                                    var arguments = new object[]{dataTable, base64};

                                    var resultObject =
BRApi.Utilities.ExecRemoteGatewayBusinessRule(si, "SIC_PrecisionDemo", arguments,
remoteGatewayName, "RunOperation");

                                    /*
                                    resultObject.ObjectResultValue: "
                                                WITHOUT encoding: values don't match --
Expected 1234567890123456.12 -- Actual 1.23456789012346E+15.
                                                WITH encoding: values should match --
Expected 1234567890123456.12 -- Actual 1234567890123456.12.
                                    "
                                    */
                                    return null;
                        }

                        public string GetBase64EncodedDataTable(DataTable dt)
                        {
                                    var serializer = new
System.Runtime.Serialization.DataContractSerializer(typeof(DataTable));
                                    var memoryStream = new System.IO.MemoryStream();
                                    serializer.WriteObject(memoryStream, dt);
                                    var bytes = memoryStream.ToArray();
                                    return Convert.ToBase64String(bytes);
                        }
            }
}
```

To test the Extender, use this Remote Business Rule:

```
namespace OneStream.BusinessRule.SmartIntegrationFunction.SIC_PrecisionDemo
{
            public class MainClass
            {
                        const decimal ExpectedValue = 1234567890123456.12M; // 18 total
digits

                        public string RunOperation(DataTable dataTable, string base64)
                        {
                                    // Retrieve and log raw dataTable
                                    var retrievedValue = dataTable.Rows[0]["DecimalColumn"];
                                    var message = $"WITHOUT encoding: values don't match --
Expected {ExpectedValue} -- Actual {retrievedValue}.";

                                    // Retrieve and log encoded data table
                                    var bytes = Convert.FromBase64String(base64);
                                    var memoryStream = new System.IO.MemoryStream(bytes);
                                    var serializer = new
System.Runtime.Serialization.DataContractSerializer(typeof(DataTable));
                                    var dt = (DataTable)serializer.ReadObject(memoryStream);
                                    var correctValue = dt.Rows[0]["DecimalColumn"];
                                    message += $"\nWITH encoding: values should match --
Expected {ExpectedValue} -- Actual {correctValue}.";

                                    return message;
                        }
            }
}
```

Troubleshooting

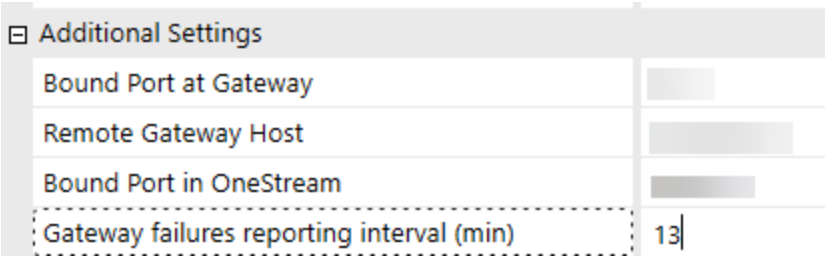
This section provides help on addressing errors in Smart Integration Connector:

Error Log

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

Every minute, 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 adds an error to the error log based on the **Gateway failures reporting interval (min)**. 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 log this gateway failure 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)**.



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 connection is reporting online, is of type "Gateway Connection" and the Version is empty, verify with your IT Admin that port 443 is fully open outbound between the Smart Integration Connector Local Gateway Server and the Azure Relay and that Deep Packet Inspection or SSL Teardown is not being performed.

Refer to Knowledge Base article KB0013213 for additional information.

⊟ General Connection Settings	
Name	dynamicsql
Description	
Connection Type	Gateway Connection
Gateway Server Name	saybrookdynamic
Web API Key	
Gateway Key	dYsf3zCYPw2plYoLZCET9Qqg1Ws60ha4B+ARmGxcaAI=
Status	Online
Instance Count	1
Version	
Active Local Gateway Server Computer Name	
⊟ 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 connection. It is recommended to wait for five minutes from creating a new connection to when you create the custom connection.

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 Smart Integration Connector 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:

```
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.

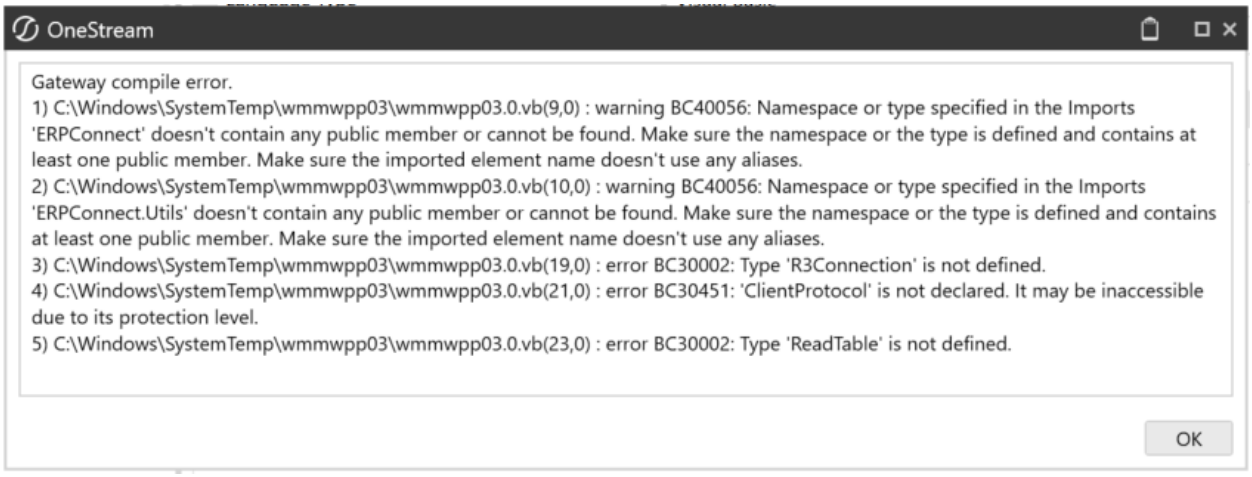
You must comment out the following line *<!--<add name="Oracle Data Provider for .NET" invariant="Oracle.DataAccess.Client" description=".Net Framework Data Provider for Oracle" type="Oracle.DataAccess.Client.OracleClientFactory, Oracle.DataAccess" />-->* when editing your OneStreamLocalGatewayConfiguration.exe.config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

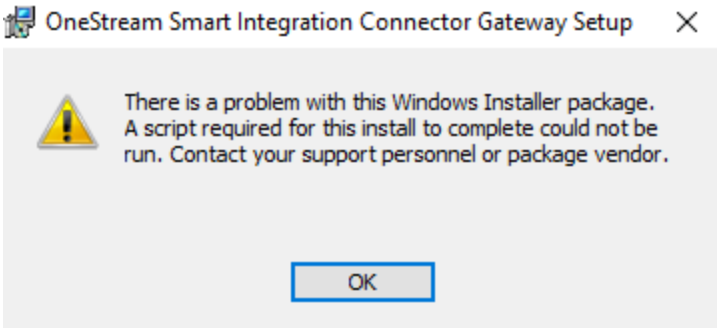
Incorrect or Missing Library References

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.



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.

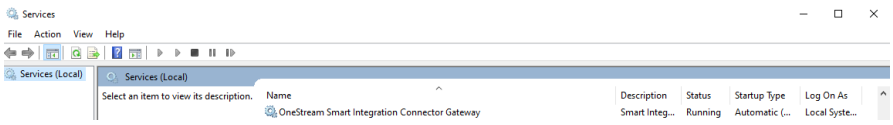
In certain cases, if you receive the error "The method or operation is not implemented" then you can use a remote business rule to transfer data. This occurs when returning the varbinary(max) datatype.

Manual Start and Stop

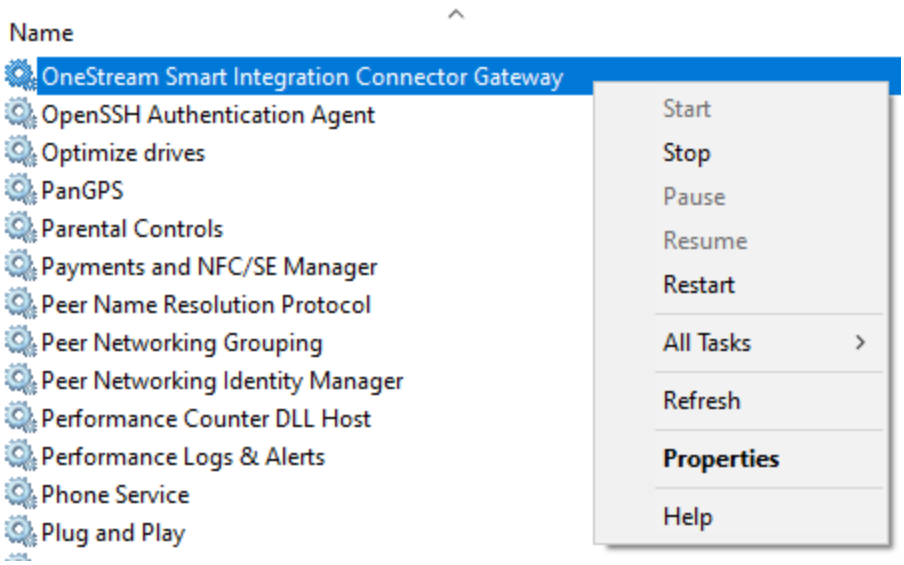
If you run into errors with the service, you may need to manually stop and restart the service. This can be accomplished in the GUI-based Services control manager as shown below or by using the command-prompt/PowerShell. The name of the service when using command-line tools is "OneStreamSmartIntegration".

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

Remote Endpoint Not Found/Could Not Decrypt

To troubleshoot the errors "Remote Endpoint Not Found" or "Could not decrypt connection string on Smart Integration Connector Gateway Connection: [Connection Name]", check your service account permissions. The service account used will require local administrative rights to access resources on the Windows server, such as the machine certificate store and private keys used for encryption.

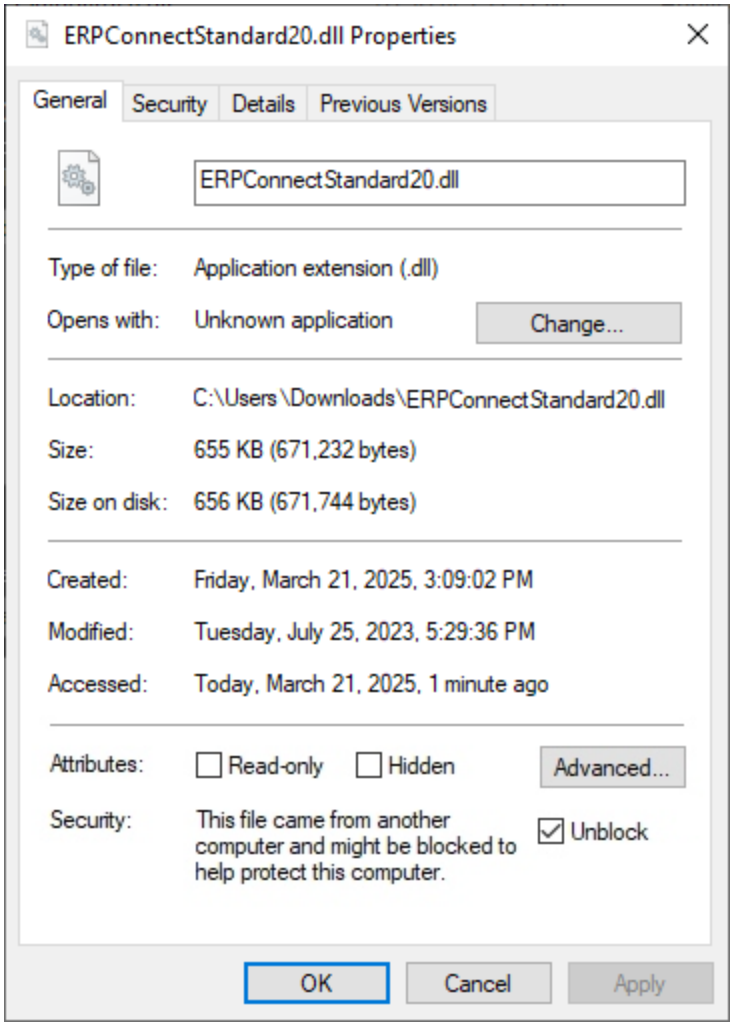
ERPConnect Module or Dependency Not Found

When executing a Smart Integration Function that uses ERPConnect, an error may occur that indicates that the ERPConnect DLL or one of its dependencies was not found or did not load. For additional information, refer to Knowledge Base article KB0013553.

If the DLL is located in the correct folder, Windows security policies are preventing the DLL from loading and the file must be unblocked.

To unblock the DLL:

1. Right click the DLL file and then select **Properties**.
2. In the **General** tab, select **Unblock** and then select **OK**.



3. Restart the Smart Integration Connector gateway. For details, refer to Restart the Gateway.

Connections Requiring a Signed Certificate

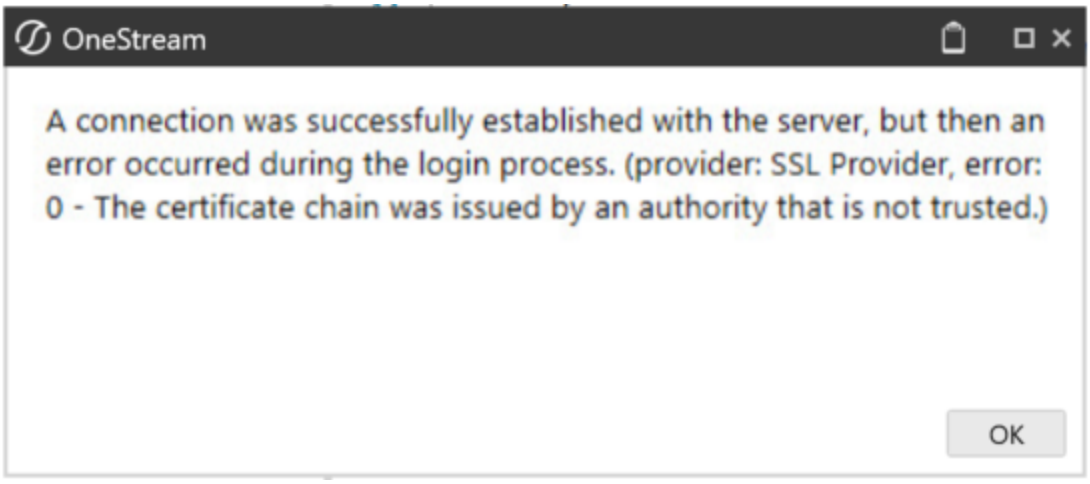
For connections that require a signed certificate in order to establish a connection, then a Certificate Authority (CA) needs to be accessible from the Smart Integration Connector Local Gateway Server in order to function.

- **Gateway Connections:** CA needs to be accessible from the Smart Integration Connector Local Gateway Server.
- **Direct Connections:** CA needs to be publicly accessible from OneStream.

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.)



If you do not have a trusted certificate installed on your DB server, you can work around this with TrustServerCertificate. However, this workaround is less secure and discouraged in production environments. To resolve this error, include **TrustServerCertificate=True**; to your connection string within the function.

Unable to Connect

If your connection fails, 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 are set up properly. See Networking and Whitelisting.

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.

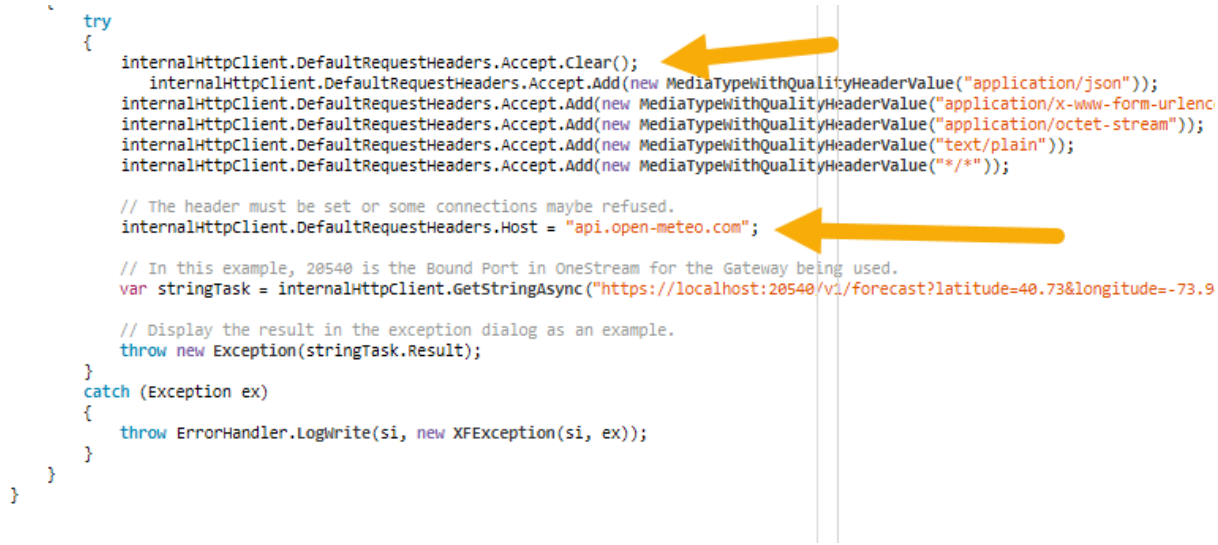
[14:13:36 INF] HTTP Request with invalid API key

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

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

Host Header Communication Error

If you copy the business rule below and are having trouble communicating with your WebAPI after compiling, ensure that you have set your host header correctly. Refer to highlights in the screenshot below.



```
    try
    {
        internalHttpClient.DefaultRequestHeaders.Accept.Clear();
           internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
        internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/x-www-form-urlenc
        internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/octet-stream"));
        internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("text/plain"));
        internalHttpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("*/*"));

        // The header must be set or some connections maybe refused.
        internalHttpClient.DefaultRequestHeaders.Host = "api.open-meteo.com";

        // In this example, 20540 is the Bound Port in OneStream for the Gateway being used.
        var stringTask = internalHttpClient.GetStringAsync("https://localhost:20540/v1/forecast?latitude=40.73&longitude=-73.9

        // Display the result in the exception dialog as an example.
        throw new Exception(stringTask.Result);
    }
    catch (Exception ex)
    {
        throw ErrorHandler.LogWrite(si, new XFException(si, ex));
    }
  }
}
```

```
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
```

```
    <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" />-->
```

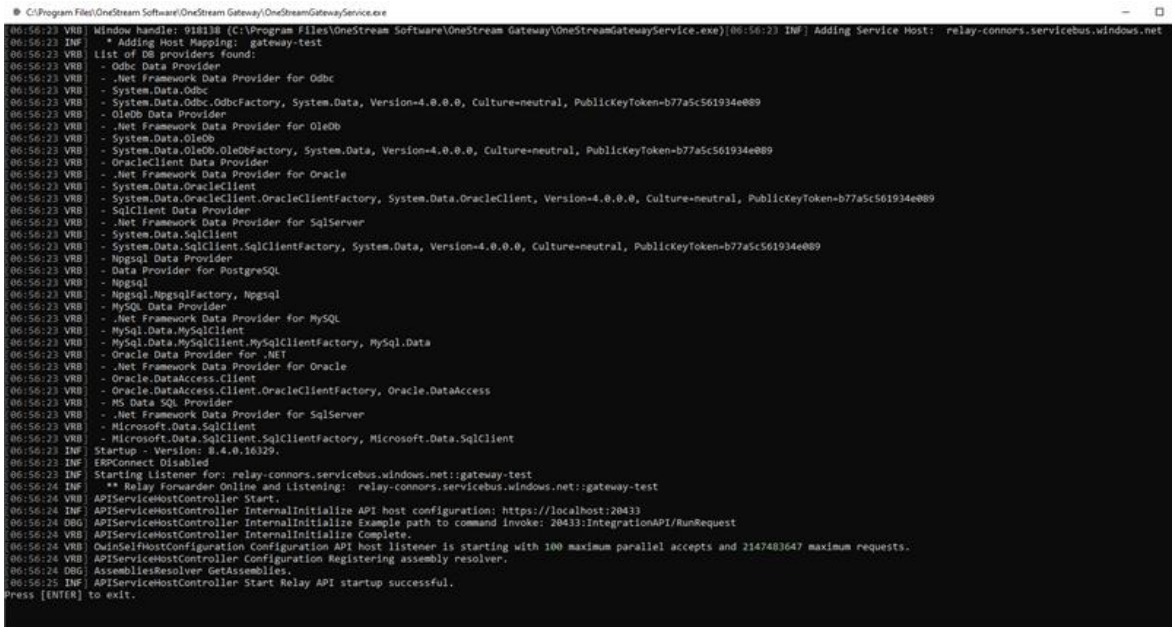
Gateway Testing Issue Resolution

If your connection testing is failing, refer to the steps below to fully test the connection.

1. You can test the connection 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.

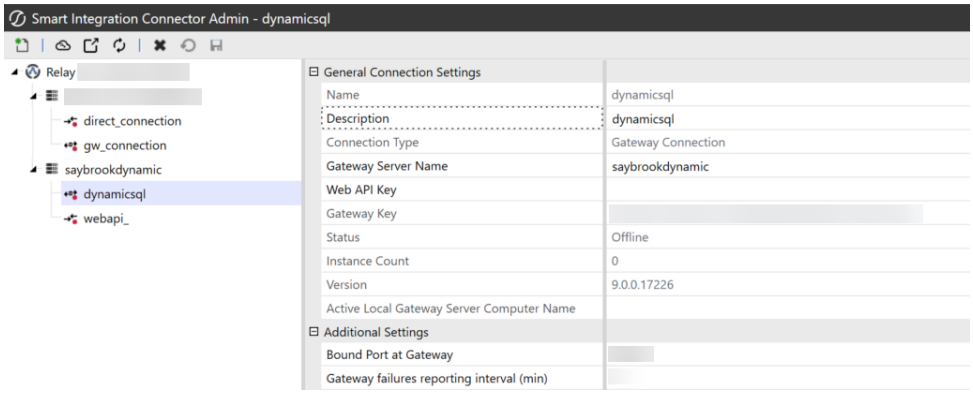
   The following command window is displayed:

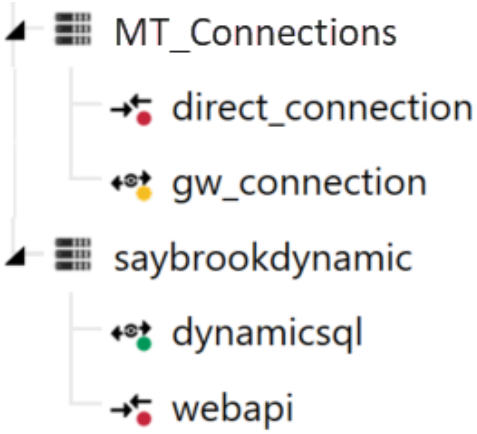


2. Correct any errors that are displayed in the command window.

> **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 Connection Details from **System** > **Administration** > **SmartIntegration Connector** > *Your connection*.
    - The **Instance Count** changes from 0 to 1.
    - The **Status** changes from **Offline** to **Online**. Additionally, status indicators turn green on the side menu if the **Connection** is **Online**, red if the **Connection** is **Offline**, and yellow if the **Connection** 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 shows the version of the running Smart Integration Connector Gateway.





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

Automatic Business Rule Decompression (Prior to Version 8.0)

Prior to version 8.0, 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
```

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

```
bytesFromFile = objRemoteRequestResultDto.ObjectResultValue
```

Frequently Asked Questions

This section contains frequently asked questions related to Smart Integration Connector:

Security and Network Configuration

Is Smart Integration Connector secure in comparison to using a VPN?

Yes, see below:

- Smart Integration Connector is encrypted end to end using TLS.
- Smart Integration Connector is 100% customer managed. IT is able to configure all data sources to OneStream.

- Database connection strings are encrypted upon saving.
- Smart Integration Connector is less invasive than VPN and is network friendly.

Are there any ports or IPs that need to be whitelisted in our firewall to set up this connection? How can we whitelist the Relay?

Smart Integration Connector Local Gateway Server requires port 443 outbound open to communicate with the Azure Relay. If you need to further lock down the firewall, you can limit the traffic outbound to go to *.servicebus.windows.net.

Is IP Whitelisting supported?

In OneStream v8.1 and higher, specific IPs or CIDRs, a range of IPs, can be whitelisted from the OneStream Windows Client Application. For details, see Networking and Whitelisting.

Configuration and Connectivity

Is there an easy way to see if my connection is online?

Yes. You can check the status within OneStream from the System/Smart Integration Connector page. Look to see if the status of the connection you selected is online.

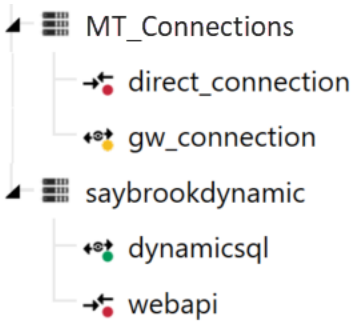
General Connection Settings	
Name	dynamicsql
Description	dynamicsql
Connection Type	Gateway Connection
Gateway Server Name	
Web API Key	
Gateway Key	dYsf3zCYPw2plYoLZCET9Qqg1Ws60ha4B+ARmGxcaAI=
Status	Online
Instance Count	2
Version	9.0.0.17226
Active Local Gateway Server Computer Name	
Additional Settings	
Bound Port at Gateway	20433
Gateway failures reporting interval (min)	5

For gateway connections, if the version is empty and the status is online, you may have some firewall rules that are blocking full connectivity over port 443.

Status indicators in the list of connections provide a visual indication of the **Connection** status.

- **Green:** The Connection is **Online**.
- **Red:** The Connection is **Offline**.
- **Yellow:** (Gateway Connections only) The Connection is **Online** and an update to the Local Gateway Server is available.

> **NOTE:** For **Direct Connections**, the yellow status is not displayed as these connections do not report a version number back to OneStream.

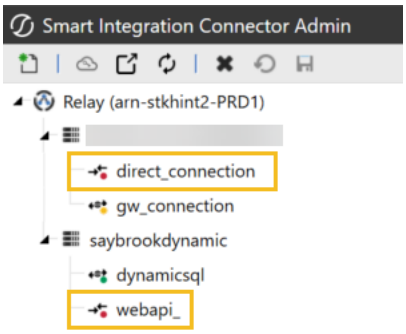


Can I connect the Smart Integration Connector Gateway Service to both DEV and PROD?

Yes, but this is not best practice. Customers in the past have tested large jobs in DEV that have caused performance issues within PROD.

Can I connect to multiple SFTP servers?

Yes. You can set up direct connections to multiple SFTP servers.



Data Handling and Query Behavior

Are there data limitations we need to be concerned about?

Smart Integration Connector has a threshold limit of five million rows or five GB. Additional CPU/RAM resources are required for large quantities of data. If this limit is exceeded, you will receive a Smart Integration Connector Remote Query Error.

What if I have a query that returns null values?

Beginning in Version 8.2, queries that contain null values are now being returned. Prior to this, null values would have to be replaced with something, like a zero.

Are there any restrictions on the time queries are allowed to run?

Beginning in Version 8.2, queries that run longer than ten minutes will now return data.

Integration and API Usage

Can anything other than JSON format be returned when using a WebAPI?

Any object type that can be wrapped in JSON can be returned.

Why would we use BRAPI verses Remote Code Execution of Business Rules?

Remote code execution takes a simple OneStream BR and ships it to the Smart Integration Connector Gateway to compile/run. The benefits of this are twofold:

- Customers can have a dependency on any third party .NET library they wish.
- Remote BRs support more complex data-ingestion scenarios as well. Picture data being pulled from multiple file shares on a customer's network, then being assembled/parsed and shipped back to OneStream. Some of this work could be offloaded into the customer's environment where direct access to the data is available.

What are some of the use cases for when we use the three Remote Code Execution Options? Is there a Use Case for BRAPI method?

There are four options for invoking things remotely on the Smart Integration Connector Gateway:

- **ExecRemoteGatewayRequest**: This is a general-purpose API that is used internally inside OneStream to do everything on a remote endpoint. It is exposed to provide granular control on timeouts or other custom scenarios.
- **ExecRemoteGatewayBusinessRule**: This takes a Smart Integration Function BR built in OneStream and sends it to a specific Smart Integration Connector Gateway to compile/run. It provides options to control caching to make it run faster on subsequent calls since BR will already be compiled.
- **ExecRemoteGatewayJob**: This is similar to the second option listed above, but instead of running synchronously and blocking things on OneStream, it is for long-running BRs. Think of this like running a DM job on the Smart Integration Connector Gateway that can run for up to thirty minutes. The status of the job is polled from OneStream to obtain the status and gather the results.
- **ExecRemoteGatewayCachedBusinessRule**: This is a BRAPI to run a cached, previously compiled BR on the Smart Integration Connector Gateway. This is seldom used.

Are there any specific Business Rule functions that are not compatible with Smart Integration Connector?

For business rule compatibility, see Business Rules.

Maintenance and Reliability

How do I adjust the reporting interval for Smart Integration Connector failures?

By default, failures will be reported every five minutes. It is recommended that you adjust the reporting time intervals for queries that run for longer than five minutes.

Will OneStream upgrades stop the connections from running?

> **NOTE:** For OneStream v9.0, it is required to use Smart Integration Connector v9.0.

Although Smart Integration Connector is designed to be backwards compatible within major versions, it is highly recommended and a best practice to always keep the two versions synced.

Host Headers

Cannot Communicate using a Direct Connection.

Connection issues:

- Check that the server is online.
- Make sure that the correct domain name is used.
- Verify that host header is included in the BR and configured correctly.

Is the Host Header required for all integrations?

No. It's only required for services that explicitly check for it. However, including it can help avoid connectivity issues with certain APIs or load-balanced environments.

What value should I use for the Host Header?

Use the domain name of the target server (for example, `api.example.com`). This should match what the server expects in incoming requests.

For a code example, refer to Obtain Data through a WebAPI.

Migration from VPN

I am migrating from a VPN solution to Smart Integration Connector. is there anything I need to take into consideration during migration?

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
Check if your VPN connection is used for securing authentication paths to OneStream. Smart Integration Connector is not providing this capability, however other considerations such as whitelisting IP access are options. SeeModify Inbound Client Access Rules.	☐
Determine how many VPN connections exist. If OneStream is integrating with data sources from multiple subnetworks, you may have multiple VPN connections. This configuration can be managed with multiple Local Gateway Servers.	☐
Smart Integration Connector requires the installation and operation of a Local Gateway Service. Make sure you have identified a Virtual Machine or physical server to operate the Local Gateway Server. See Requirements.	☐
Take inventory of what you currently use for example, business rules, workspaces, queries, grid views, drill-backs, and whitelisted endpoints for each plan for any updates needed when using Smart Integration Connector.	☐
Set up a time with your OneStream Cloud Support Representative to plan when the VPN can be disconnected.	☐

Troubleshooting

See Troubleshooting.