Levridge

AgXml Integration Deployment

Table of Contents

1. Deployment Overview

The AgXml Integration consists of the following parts.

Integration Host

This is the heart of the integration. The Integration host runs as a Windows service and is responsible for receiving messages from the Azure Service Bus, transforming them, and sending them to the integration target.

Each instance of the Integration Host can receive messages from a single Azure Service Bus topic and send them to a single integration target (e.g., FinOps or file). If you have multiple integration partners, you must configure multiple instances of the Integration Host.

Translation Service

The Translation service is a sub-component of the Integration Host, it does not exist on its own. The translations are stored in an MS SQL Server database and are applied to the integration source document. Service configuration lives in the Integration Host's JSON file.

Translation Console

The Translation Console allows you to import and export translations.

Integration Console

The Integration Console allows you to process the messages in the Dead-letter Queue and setup translations. You must configure a new instance of the Integration Console for each Integration Host.

Translation Migrations

The Translation Migration (Levridge.Integration.IntegrationService.Translation.Migrations.exe) application is used exclusively to create or update the translation database schema.

File Watcher Service

The File Watcher Service (Levridge.FileWatcherService.exe) runs as a Windows Service and is responsible for sending new AgXml files to the Azure Service Bus.

The service watches a configured folder; and when a new or modified file is detected, it sends it to the ASB and moves the file to another folder.

2. Configuration settings

Configuration settings for all applications are stored in a file name "Appsettings.json". This file is co-located next to the application executable.

When setting configuration information, you will likely need to update all Appsettings.json files. The Integration Host configuration file is the only file to contain the integration source, integration target, and OData sections.

The Appsettings.json file is responsible for configuring:

- The Translation database connection information
- The integration source and target systems

- The Source Azure Service Bus
- The Target OData information
- Logging (Logging in .NET | Microsoft Docs)
- Application Insights (Application Insights logging with .NET - Azure Monitor | Microsoft Docs)

3. Authentication

FinOps OData services uses OAuth 2.0 shared secrets for service-to-service authentication to authenticate the Integration Service. Shared secrets involve registering an application with Azure Active Directory (AAD) to create a shared secret. This secret is given to AAD to by the client application (e.g., Integration Host) when the server application (e.g., FinOps OData) requests authentication. AAD decides if the secret is correct and redirects the client to the server.

None of the AgXml Integration components currently offer their own authentication. This means you may only configure them on a secure server.

4. IIS Hosting Prerequisites

All applications require the DotNet 6 runtime.

Before hosting the integration, you must install the ASP.NET Core Module/Hosting Bundle. You can download the installer using the following link:

Current .NET Core Hosting Bundle installer (direct download)

For more detailed instructions on how to install the ASP.NET Core Module, or installing different versions, see Install the .NET Core Hosting Bundle and Host ASP.NET Core on Windows with IIS | Microsoft Docs.

5. Application Registration

Authentication requires an application registration and an Azure Application Id. To register your application and manually add the app's registration information to your solution, follow these steps:

1. Sign into the Azure portal using either a work or school account, or a personal Microsoft account.
2. If your account gives you access to more than one tenant, select your account in the top right corner, and set your portal session to the desired Azure AD tenant.
3. Navigate to the Microsoft identity platform for developers App registrations page.
4. Select New registration.
5. When the Register an application page appears, enter your application's registration information:
    a. In the Name section, enter a meaningful application name that will be displayed to users of the app, for example AspNetCore-Quickstart.
    b. In Redirect URI, add https://localhost:44321/, and select Register.
6. Select the Authentication menu, and then add the following information:
    a. In Redirect URIs, add https://localhost:44321/signin-oidc, and select Save.
    b. In the Advanced settings section, set Logout URL to https://localhost:44321/signout-oidc.
    c. Under Implicit grant, check ID tokens.
    d. Select Save.
7. Create a section in Appsettings.json named **AzureAd**.

```
"AzureAd": {
  "Instance": "https://login.microsoftonline.com/",
  "Domain": "yourdomain.com",
  "TenantId": "00000000-0000-0000-0000-000000000000",
  "ClientId": "00000000-0000-0000-0000-000000000000",
  "CallbackPath": "/signin-oidc"
}
```

Taken from Quickstart: Add sign-in with Microsoft to an ASP.NET Core web app.

6. Application Insights

Configure application insights here.

7. Solution Configuration

These sections refer to an *InstanceName* value, e.g., "C:\Levridge\LevridgeIntegrationHost_{InstanceName}". This is necessary to support multiple integration partners.

This means to configuration two partners, Lone Commodity and Granite Milling, you would create the following Integration Host folders.

- C:\Levridge\LevridgeIntegrationHost_LoneCommodity
- C:\Levridge\LevridgeIntegrationHost_GraniteMilling

7.1 Create the Translation Database

The Translation data is stored in a Microsoft SQL Server database, either locally or in Azure.

Create the Azure SQL Database

1. Create a single database - Azure SQL Database | Microsoft Docs
2. Add the database connection information to the following appsettings.json files.
   a. IntegrationConsole_{InstanceName}\appsettings.json
   b. Levridge.Integration.Host_{InstanceName}\appsettings.json
   c. TranslationConsole_{InstanceName}\appsettings.json
   d. TranslationMigration_{InstanceName}\appsettings.json

Configuration Examples

To connect to a database named "TranslationService" on the local SQL Server, using a trusted connection:

```
"ConnectionStrings": {
  "TranslationService": "Server=(local);Database=TranslationService;Trusted_Connection=True;MultipleActiveResultSets=true;"
}
```

To connect to an Azure SQL Database named "translationserver" using SQL credentials:

```
    "ConnectionStrings": {
      "TranslationService": " Server=tcp:translationserver.database.windows.net,1433;Initial Catalog=TranslationService;Persist Security Info=False;User ID={your_userid};Password={your_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
    }
```

Create or update the initial database schema (Preferred)

1. Extract the contents of Levridge.Integration.IntegrationService.Translation.Migrations.zip to “C:\Levridge\TranslationMigration_{InstanceName}”
2. From an Administrative command prompt, execute
   Levridge.Integration.IntegrationService.Translation.Migrations.exe migrate

Create the initial database schema using EF Core Migrations (Alternate)

1. Install the tools
   dotnet.exe tool install --global dotnet-ef
2. Create a migration named “InitialCreate”
   dotnet.exe ef migrations add InitialCreate

Update the database schema after upgrade

1. dotnet.exe ef database update

7.2 Configuring an instance of the Integration Host

The Integration host must be configured to run as a Windows Service. There are 3 PowerShell scripts in the application folder to support this.

Configuration PowerShell scripts

InstallService.ps1

This script installs a service and sets it to automatically run on startup. It accepts the following command line parameters:

- PublishPath - the path to the folder from which you want the service to run
- ServiceUser - user account under which the service will run (default = $env:computername+"\IntegrationHost")
- ServiceName - the name of the service (default = "Levridge.Integration.Host")
- ServiceDescription - the description for the service (default = "Levridge Integration Host Service")
- ServiceDisplayName - the display name for the service (default = "Levridge Integration Host")
- SourceName - the source name under which EventLog entries should be logged (default = $ServiceName)

This script will also execute the InstallEventSource.ps1

InstallEventSource.ps1

This script will install the specified EventLog source. It accepts the following command line parameter:

- SourceName - the source name under which EventLog entries should be logged (default = "Levridge.Integration.Host")

RemoveService.ps1

This script will remove the specified service. It accepts the following command line parameter:

- ServiceName

Levridge.Integration.Host.exe

This application can be run with the following command line parameters:

- debug (-d) - This will cause the application to wait for a debugger to be attached before it continues. This can be helpful to debug startup issues for services or web applications.
- service (-s) - This will cause the application to run as a service (if the debugger is not attached).
- SourceName (sn) - This will cause the application to use the specified source name for the application EventLog Source Name. If no source name is specified, the default application source name is use.

To install the Windows Service

1. Extract the contents of Levridge.Integration.Host.zip to "C:\Levridge\LevridgeIntegrationHost_{InstanceName}"
2. Install the Windows Service using the "InstallService.ps1" PowerShell script. You must execute the script from an Administrative Command Prompt. Use the following command line parameters, replacing the $() values with your specific value:
   a. -serviceName '$(IntegrationServiceName)'
   b. -fileName '$(TargetFolder)/$(IntegrationServiceName)/Levridge.Integration.Host.exe'
   c. -displayName 'Levridge Integration Host Service $(InstanceName)'
   d. -description 'Hosts the integration services.'

7.3 Configure an Instance of the Integration Console

The Integration console runs as a Web App inside IIS using the Kestrel web server and is configured by the Appsettings.json file located next to the IntegrationConsole.Server.exe executable.

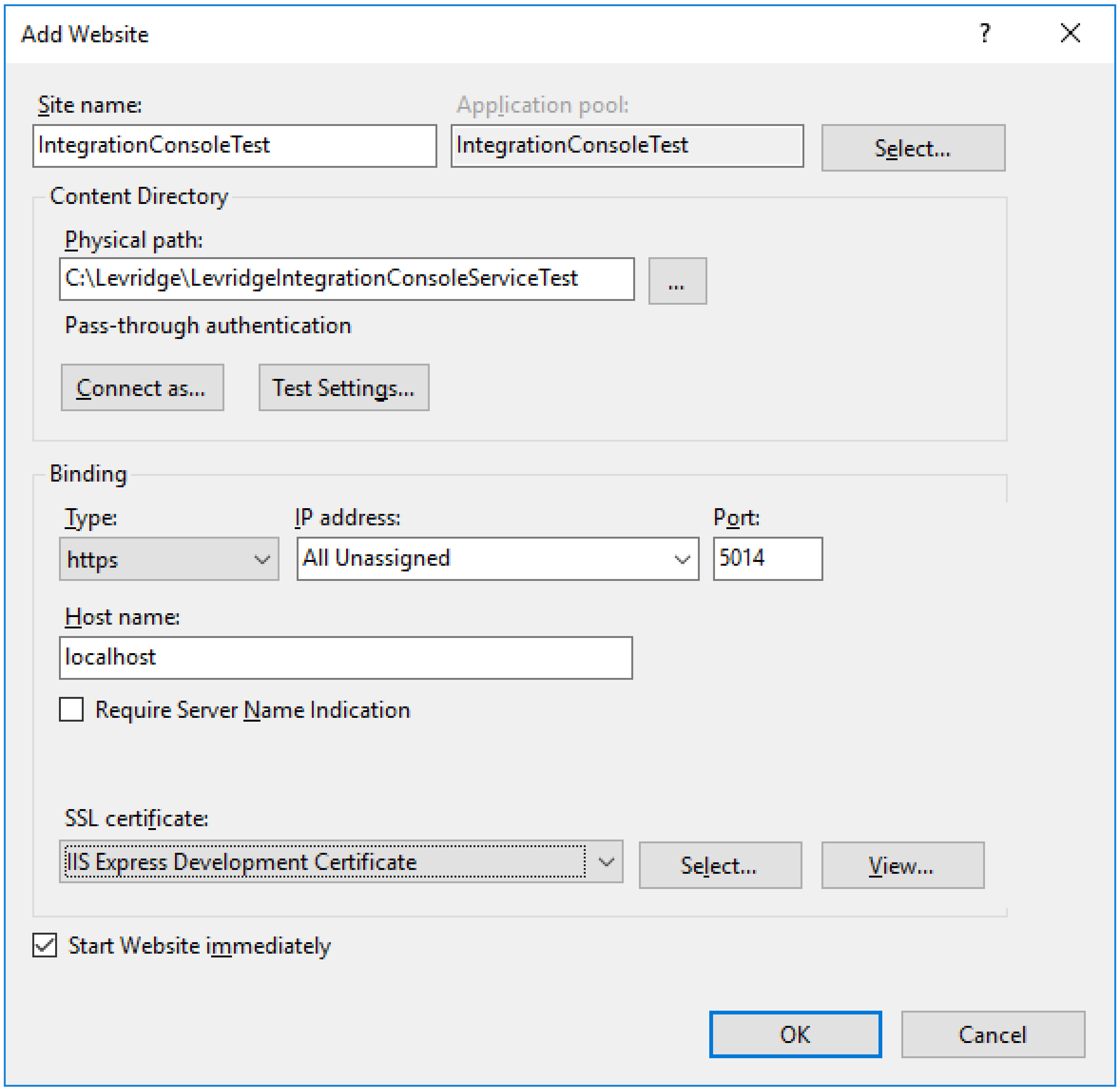
The service bus is identified by the "ServiceBus" configuration section.

```
"ServiceBus": {
  "ConnectionString": "Endpoint={ASBConnectionString},
  "TopicName": "{Name}",
  "SubscriptionName": "{Name}",
  "RequiresSession": {true|false}
}
```

To install the service in IIS

1. Extract the contents of IntegrationConsole.zip to "C:\Levridge\IntegrationConsole_{InstanceName}"
2. Open Internet Information Services (IIS) Manager
3. Expand Sites, and right-click and choose "Add Website…"
4. Choose a site name
5. Select the path from step 1.

6. Bind to an unused https port (i.e., 5014)
7. Set your host name.
8. Choose your SSL certificate.



Add Website

Site name: IntegrationConsoleTest

Application pool: IntegrationConsoleTest

Content Directory

Physical path: C:\Levridge\LevridgeIntegrationConsoleServiceTest

Pass-through authentication

Binding

Type: https

IP address: All Unassigned

Port: 5014

Host name: localhost

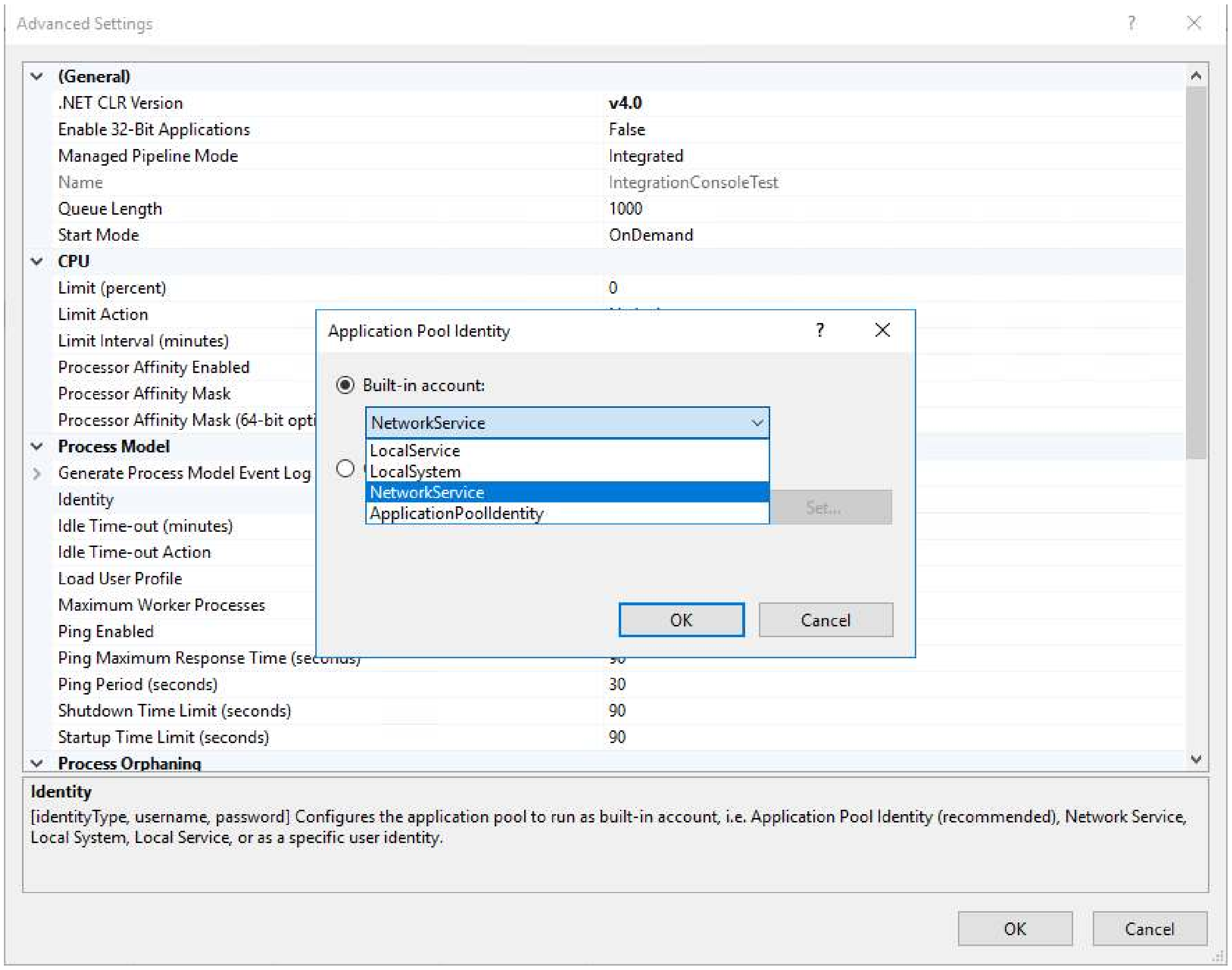
☐ Require Server Name Indication

SSL certificate: IIS Express Development Certificate

☑ Start Website immediately

9. Select the Application Pool for your site.
10. Open the Advanced Settings dialog

11. Change the Identity to “NetworkService”. **Note: This is only necessary to use a trusted connection to the Translation Database running on a local SQL Server.**


Advanced Settings

(General)	
.NET CLR Version	**v4.0**
Enable 32-Bit Applications	False
Managed Pipeline Mode	Integrated
Name	IntegrationConsoleTest
Queue Length	1000
Start Mode	OnDemand
CPU	
Limit (percent)	0
Limit Action	
Limit Interval (minutes)	
Processor Affinity Enabled	
Processor Affinity Mask	
Processor Affinity Mask (64-bit opti	
Process Model	
Generate Process Model Event Log	
Identity	
Idle Time-out (minutes)	
Idle Time-out Action	
Load User Profile	
Maximum Worker Processes	
Ping Enabled	
Ping Maximum Response Time (seconds)	90
Ping Period (seconds)	30
Shutdown Time Limit (seconds)	90
Startup Time Limit (seconds)	90
Process Orphaning	

Application Pool Identity

Built-in account: NetworkService

LocalService
LocalSystem
NetworkService
ApplicationPoolIdentity

Set... OK Cancel

Identity
[identityType, username, password] Configures the application pool to run as built-in account, i.e. Application Pool Identity (recommended), Network Service, Local System, Local Service, or as a specific user identity.

OK Cancel

12. Test your site by browsing to your selected port (e.g., https://localhost:5014)

7.4 Configure the File Watcher Service

The File Watcher Service runs as a Windows Service, sending new AgXml documents to the Azure Service Bus.

The service bus and folder configuration looks like this:

```
{
  "ServiceBus": {
    "ConnectionString": "Endpoint={ASBConnectionString},
    "TopicName": "{Name}"
  }
  "FolderName": "C:\\{Folder}\\Watched\\Watch",
  "SuccessFolder": "C:\\{Folder}\\Watched\\Success",
  "FailureFolder": "C:\\{Folder}\\Watched\\Failed",
  "IncludeSubfolders": false,
}
```

To install the Window Service

13. Extract the contents of IntegrationConsole.zip to “C:\Levridge\LevridgeFileWatcher_{InstanceName}”
14. Install the Windows Service using sc.exe create. You must execute the script from an Administrative Command Prompt. Use the following command line parameters.
    a. create
    b. “Levridge File Watcher {InstanceName}”
    c. displayname=” Levridge File Watcher {InstanceName}”
    d. start=auto
    e. binpath={PathTo}\Levridge.FileWatcherService.exe

Example:

```
sc.exe create "Levridge File Watcher" displayname="LevridgeFileWatcher" start=auto
binpath="%cd%\Levridge.FileWatcherService.exe"
```

8. Application Documentation

8.1 Integration Translation Console

The TranslationConsole allows you to import and export translation definitions from the database.

To import:

```
TranslationConsole.exe import <file>
```

To Export:

```
TranslationConsole.exe export <file>
```

Each translation is defined by the following structure:

```
{
  "SourceSystem": "<NAME>",
  "XPath": "<XPATH>",
  "EntityTypeName": "<NAME",
  "AdditionalValuePaths": [
   {
    "ValueXPath": "<XPATH>"
   }
   {
    "ValueXPath": "<XPATH>"
   }
  ],
  "ValuePairs": [
   {
    "TargetSystem": "<NAME>",
    "From": "<VALUE>",
    "To": "<VALUE>"
   },
   {
    "TargetSystem": "<NAME>",
    "From": "<VALUE>",
    "To": "<VALUE>"
   }
  ]
 }
```

SourceSystem

The “SourceSystem” tag is an Enum with one of the following values:

1. DynamicsAx
2. AgXML

XPath

The “XPath” value is an XPath expression to the primary value. If the value appears in more than one node, provide additional “XPath” expressions in the “AdditionalValuePaths” array.

The XPath expression must use the "ns" namespace alias. Xml namespaces are roughly equivalent to C# namespaces. This means that you cannot uniquely identify a node without specifying its namespace. At runtime, the Xml document's default namespace will be assigned the alias "ns".

EntityTypeName

This is the name of the entity being translated.

Most of the time this will be "CommodityMovement".

ValuePair

The translation values are defined as an array of “ValuePair” objects. Each “ValuePair” defines the target system, the original value, and the replacement value.

Example

The translation file is a JSON file that looks like this:

```
[
 {
   "SourceSystem": "AgXML",
   "XPath": "/ns:CommodityMovement/ns:CommodityInformation/ns:CommodityNameCode",
   "EntityTypeName": "CommodityMovement",
   "AdditionalValuePaths": [
    {
      "ValueXPath":
"/ns:CommodityMovement/ns:CommodityMovementQualityInformation/ns:CommodityMovem
entActualQualityInformation/ns:QualityCertificate/ns:QualityCertificateHeader/ns:CommodityIn
formation/ns:CommodityNameCode"
    },
    {
      "ValueXPath":
"/ns:CommodityMovement/ns:CommodityMovementWeightInformation/ns:CommodityMovem
entActualWeightInformation/ns:WeightCertificate/ns:WeightCertificateDetail/ns:WeightCertific
ateLine/ns:CommodityInformation/ns:CommodityNameCode"
    }
   ],
   "ValuePairs": [
    {
      "TargetSystem": "DynamicsAx",
      "From": "C500001",
      "To": "500001"
    }
   ]
 },
 {
   "SourceSystem": "AgXML",
   "XPath":
"/ns:CommodityMovement/ns:ShipmentInformation/ns:Shipper/ns:EntityInformation/ns:Entity
Name",
   "EntityTypeName": "CommodityMovement",
   "AdditionalValuePaths": [],
   "ValuePairs": [
    {
     "TargetSystem": "DynamicsAx",
     "From": "Key_00",
     "To": "00_Key"
    },
    {
     "TargetSystem": "DynamicsAx",
     "From": "Key_01",
     "To": "01_Key"
    },
    {
     "TargetSystem": "DynamicsAx",
     "From": "Key_02",
     "To": "02_Key"
    }
   ]
 }
]
```