Neuron ESB Training: Fundamentals

Overview

This training will provide you with the knowledge necessary to begin using Neuron ESB and understand the foundation upon which it is built. After this training you will be able to:

- Describe the core concepts around which Neuron ESB was built
- Install Neuron ESB
- Create a new Neuron ESB configuration, setup a Topic and Publishers/Subscribers (Parties)
- Work with the Neuron ESB Test Client
- Understand the basics of the Neuron ESB Client API and be able to write code to send and receive asynchronous and synchronous messages

You should complete the exercises provided at the end of the training to confirm your understanding of the material presented.

Prerequisites

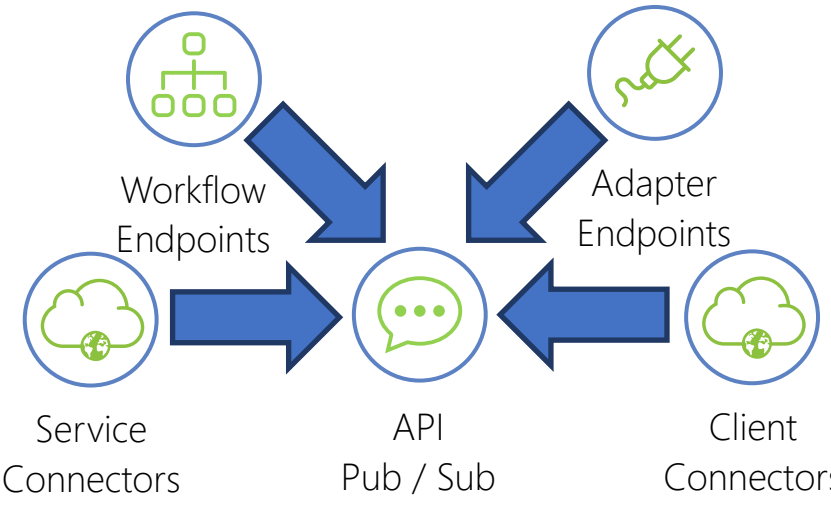
- .NET 4.7.2
    - (Neuron ESB Version 3.7 requires 4.7.2, Version 3.7.5 requires 4.8)
- RabbitMQ installed
- Erlang installed
- Visual Studio 2013

Concepts

Neuron ESB is an Enterprise Integration platform built on Microsoft .NET. Neuron's ESB capabilities allow it to be used as a distributed Messaging Oriented Middleware (MOM) with centralized management, as Web Services intermediary and router (Service Broker), an Enterprise Application Integration (EAI) platform, Workflow platform or any combination thereof.

Publish and Subscribe

As the diagram below depicts, Neuron ESB's messaging core and the publish/subscribe paradigm is the foundation which all other functionality can easily leverage:



By allowing all other integration components (i.e. Adapters/Connectors, Services and Workflows) to work seamlessly with the publish/subscribe core means Neuron ESB is inherently capable of functioning in asynchronous and 1-many or broadcast scenarios. Neuron ESB messages may also be sent using a Request Semantic or synchronous Message Exchange Pattern (MEP) which allows Neuron ESB to also be used for point-to-point communication. Neuron ESB's messaging core is predicated upon a hierarchical, Topic based pub/sub messaging system in which Parties send (publish) messages to Topics and Parties receive (subscribe) messages published to Topics.

Topics

Neuron ESB provides a powerful, hierarchical Topic based publish and subscribe messaging system where publishers can label each message with the name of a Topic, rather than addressing it to specific recipients. Topic based messaging systems tend to be more intuitive as they can be closely modeled after the specific business requirements and/or the existing organizational hierarchy. Neuron ESB manages sending the message to all eligible recipients that have expressed interest in receiving messages on that Topic. This form of asynchronous messaging is a far more scalable architecture than point-to-point alternatives, since message senders (publishers) need only concern themselves with creating the original message and can leave the task of servicing recipients (subscribers) to the messaging infrastructure.

Topic-based publish and subscribe messaging systems share several common attributes some of which are:

- Subscribers subscribe to one or more topics and only receive messages that are of interest to them
- The publishers have no knowledge about the subscribers, including how many there are or where they live
- The subscribers have no knowledge about the publishers, including how many there are or where they live
- New systems (publishers or subscribers) can easily be added or removed from the flow of information, without code changes.

This type of messaging architecture sends messages only to the applications that are interested in receiving the messages without knowing the identities of the receivers. There are many other reasons to adopt a hierarchical Topic taxonomy.

- It allows for more meaningful self-describing subscriptions which can model events, systems, categories or functions.
- Hierarchical Topics can be easier to manage, providing the hierarchy doesn't extend beyond 3 to 5 levels.
- They are more intuitive as they can be modeled to represent specific resources, people or systems to route messages to.
- Subtopics can be used to describe different versions of existing services that messages may be routed to.

A hierarchical Topic taxonomy should be an orderly, documented set of classifications that have contextual relationships to the messages on the bus and should be organized and related to one another in a meaningful way. Topic taxonomies can differ across message population; for example, one set of messages may be categorized as "infrastructure" types, whereas other messages may be categorized as "business" activities, events or requests.

However, care should be taken to ensure that sub topics are not named to represent a specific action as the message published should describe the action. This way if the action changes, the sub topic can remain consistent.

For example, a user may create a root topic called "Account", but then create several sub topics such as:

- Account.Loan
- Account.Loan.Debt
- Account.Loan.Debt.V2
- Account.Loan.Payment
- Account.Loan.Arrears
- Account.Loan.Cancellation

When hierarchical (sub topics) Topics are created, they can be logically mapped to recipients like Service and Adapter Endpoints or even to different versions of services that those endpoints represent.

Transports and QOS

Neuron ESB Topics control not only the path/route of the messages in Neuron ESB but also the Quality of Service (QOS) which is configured by choosing a specific Transport for the Topic. QOS attributes are always configured at the "root" Topic level and affect all of the respective sub topics, including whether or not durable messaging is enabled.

Neuron ships with 5 transports for Topics

- TCP – This transport is based on the WCF NetTcp binding and offers ordering, security and session reliability as options. It is not a durable transport so message loss can occur if no

subscribers are online when a message is published. TCP requires Port configuration and is commonly used in the following scenarios:

  - Request/Response style messaging where neither durability nor transactions are required. A good example would be web service traffic.
  - When the Neuron ESB Client API is hosted in custom applications existing on machines REMOTE to the Neuron ESB server
  - Multiple instances of the same Party must receive the same messages

- Named Pipe – This transport is based on the WCF NetNamedPipe binding. This transport is slightly faster than TCP, and easier to configure (does not require Port configuration). However, this transport cannot be configured for topics that will be utilized by remote applications hosting the Neuron Client API. It is not a durable transport so message loss can occur if no subscribers are online when a message is published. It is commonly used in the following scenarios:
  - Request/Response style messaging where neither durability nor transactions are required. A good example would be web service traffic.
  - When the Neuron ESB Client API is hosted in custom applications that coexist on the SAME machine as the Neuron ESB server
  - When all messaging traffic is generated by configured Service or Adapter Endpoints (i.e. all messaging is local to the Neuron ESB Server) and neither durability nor transactions are required.
  - Multiple instances of the same Party must receive the same messages
- MSMQ – This transport is based on the WCF NetMsmq binding. It is optionally a durable and transactional transport so parties that are not online when a message is sent can receive messages when they reconnect. The MSMQ transport provides a “pull” style subscription model so subscribers cannot be overloaded by messages. Because transactions are supported, the ambient transaction can be accessed within the Receive Handler of the Neuron ESB Client API or within a Neuron ESB Business Process. When configured as an in-memory transport (i.e. Durability and Transaction properties set to False) this transport can be faster and more efficient than TCP. This transport cannot be used to send messages greater than 4 MB in size but can provide guaranteed, once only delivery of messages across the bus. It is commonly used in the following scenarios:
  - Where message loss is not acceptable.
  - One way (multicast/datagram) message patterns
  - Where XA style transactional support is required
  - Subscribers must receive messages, yet may not be consistently online
  - Ordered messaging
- RabbitMQ – This transport is based on RabbitMQ which is an implementation of the AMQP open standard for messaging middleware. It is optionally a durable transport so parties that are not online when a message is sent will receive messages when they reconnect. This transport can be used as an alternative to MSMQ as a topic transport. Although this transport does not support atomic transactions, it does support ack/nack as well as batch style transactions. It also

does not have the 4MB message size limit that MSMQ has. It is commonly used in the following scenarios:

- Where message loss is not acceptable.
- One way (multicast/datagram) message patterns
- Subscribers must receive messages, yet may not be consistently online
- Ordered Messaging

In addition to Transports, Neuron ESB allows other QOS of elements to be configured at the Topic level:

- Auditing (message tracking)
- Compression
- Publisher-based throttling
- Encryption

Parties

A Party may subscribe to 1 or more Topics or sub Topics. Parties use Subscriptions to restrict which Topics they can publish messages to, as well as determine what messages they are interested in receiving from the bus. Parties that send messages are known as publishers, and parties that receive messages are known as subscribers. A party could be both a publisher and a subscriber. One or more subscriptions can be assigned to a Party.

Subscriptions

A Subscription is composed of a Topic (or sub Topic), the permission in which a message can be sent to or from that Topic (i.e. Send/Receive) and can be further optionally restricted using one or more Conditions.

A Condition is either a preexisting or ad hoc filter expression (using predicates) that can include message header properties as well as message content. Sometimes this is referred to as “Content Based Routing”.

Messages

A message is information that one Party sends to or receives from the bus. A message contains both data (the information that some other system, resource or person may be interested in) as well as metadata. Respectively, these are referred to as the payload and header (or context) properties of the message. Both are defined as parts of a Neuron ESB Message.

The payload of a Neuron ESB Message can be one of several formats:

- Serializable .NET Object - The ability to pass .NET objects provides flexibility for developers who prefer use of objects over XML
- Binary data - The ability to pass binary data provides flexibility for developers who have to share content that is neither serializable nor XML

- Text data – Any type of string data
- XML data – Any XML data. XSD schemas are not required to use XML as the payload
- JSON data – An alternative to XML for transmitting data between a server and web application

Parties are created within the Neuron ESB Explorer and are represented by a logical name, often referred to as the Party ID, Subscriber ID or Publisher ID.

Because Parties control the interaction of messages to Topics, all Adapter Endpoints, Workflow Endpoints, Client Connectors and Service Connectors that want to either publish or subscribe to Topics MUST be configured with a Party. The configured Party will determine what messages are received by Adapter Endpoints, Workflow Endpoints, or Service Connectors, and where (what Topic) messages will be published to by Adapter Endpoints and Client Connectors.

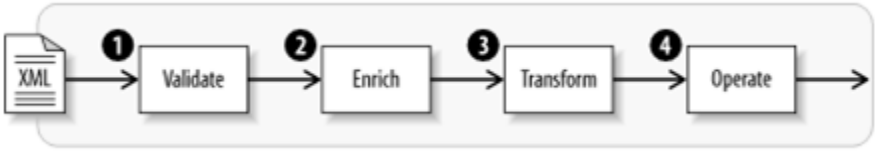
Although Workflow Endpoints MUST be configured with a Party, Pub/Sub Messaging is optional for Client Connectors, Service Connectors and Adapter Endpoints. Party configuration is not necessary for these because they can interact directly with each other as well as Business Processes, without the need to send and receive messages over Topics

---

A Party may also optionally manipulate the message it is sending or the message it receives. This manipulation occurs via a Neuron ESB Business Process or Workflow.

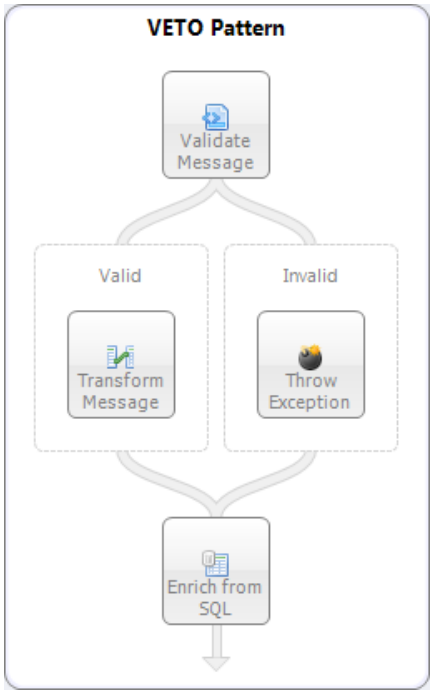
Business Processes

Processes are used within integration and messaging products to provide real-time workflow and business process capabilities for message processing and/or to implement specific integration patterns. For example, when a user or system submits a message to the bus, specific steps may first need to happen. These steps may include validating and transforming the message before it is published to other subscribers. VETO is a common integration pattern that stands for Validate, Enrich, Transform, Operate (see Figure below). The VETO pattern and its variations can ensure that consistent, validated data will be routed throughout the ESB.



Neuron ESB provides a unique process implementation that goes beyond the industry standard. Many patterns, like VETO, can be easily developed using the Neuron ESB process designer. The Neuron ESB Process Designer ships with 46 configurable process steps that do everything from validating a message to querying a web service or database. Users can also create “custom” process steps for reusability. For

example, **the figure below** displays a VETO pattern using the Neuron ESB Process Designer. When using Neuron ESB, this pattern can be implemented without any external custom code dependencies.



The Neuron ESB Process Designer significantly extends the ability to develop more complex patterns and processes without introducing additional workflow technologies into a project. For example, a complex business process may involve the need for custom code execution, a decision based on external criteria, calling to a web service or data store, rerouting a message, but overall, being able to maintain transactions and deal with the exceptions as they occur.

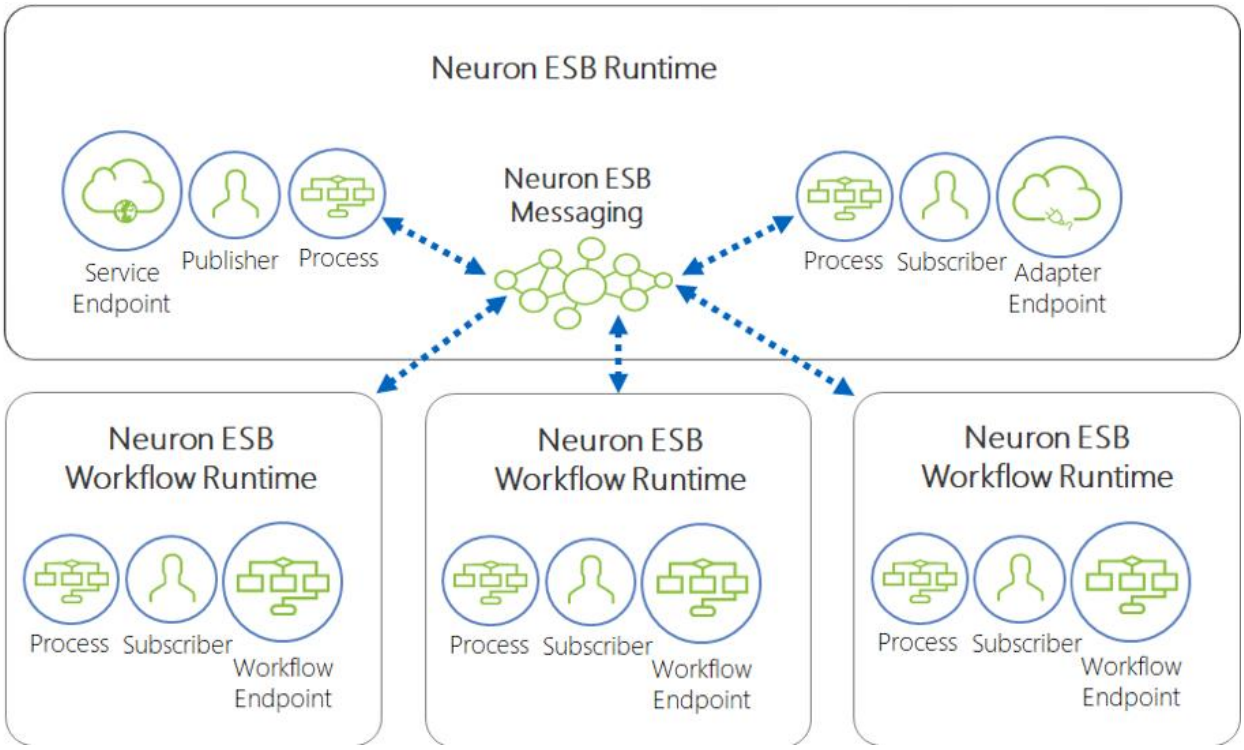
Within Neuron ESB more than one Business Process may be configured by a Party and conditions may be set inside or outside of the process to control process execution.

A Business Process may be attached directly to client connectors and adapter endpoints, running in publish mode, allowing these endpoints to directly execute the Business Process without needing to participate in the pub/sub model.

It is important to understand that within Neuron ESB's pub/sub architecture, Business Processes are connected to Parties, Client Connectors or Adapter Endpoints, not Topics. In fact, a Business Process may be used to alter the Topic of a message and thus alter the message flow. A Party may also be hosted in your own executable on remote machines. This means that Business Process logic attached to those Parties is inherently distributable while being centrally maintained. If a Business Process is either added or modified on the server, the Party, regardless of hosting environment and location, will automatically get the updated Business Process and execute it within the environment and location it is hosted in. This allows Neuron ESB to be used in ways not available to server-only products and thus allows Neuron ESB to function as a complete distributed, Service Oriented Architecture (SOA) platform.

Endpoint Hosts

In previous versions of Neuron ESB, the Neuron ESB Runtime Windows Service (esbservice.exe) was used to host all Adapter and Service Endpoints, as well as all Messaging Publishing Services and all internal Services. These services were isolated in their own .NET AppDomains within the Neuron ESB Runtime service. However, a fault tolerant hosting environment was provided for the Neuron ESB Workflow Engine. The hosting was embodied by Neuron ESB Availability Groups. Availability Groups were used to load balance the execution of Workflow Instances across multiple servers in dedicated/isolated host processes.



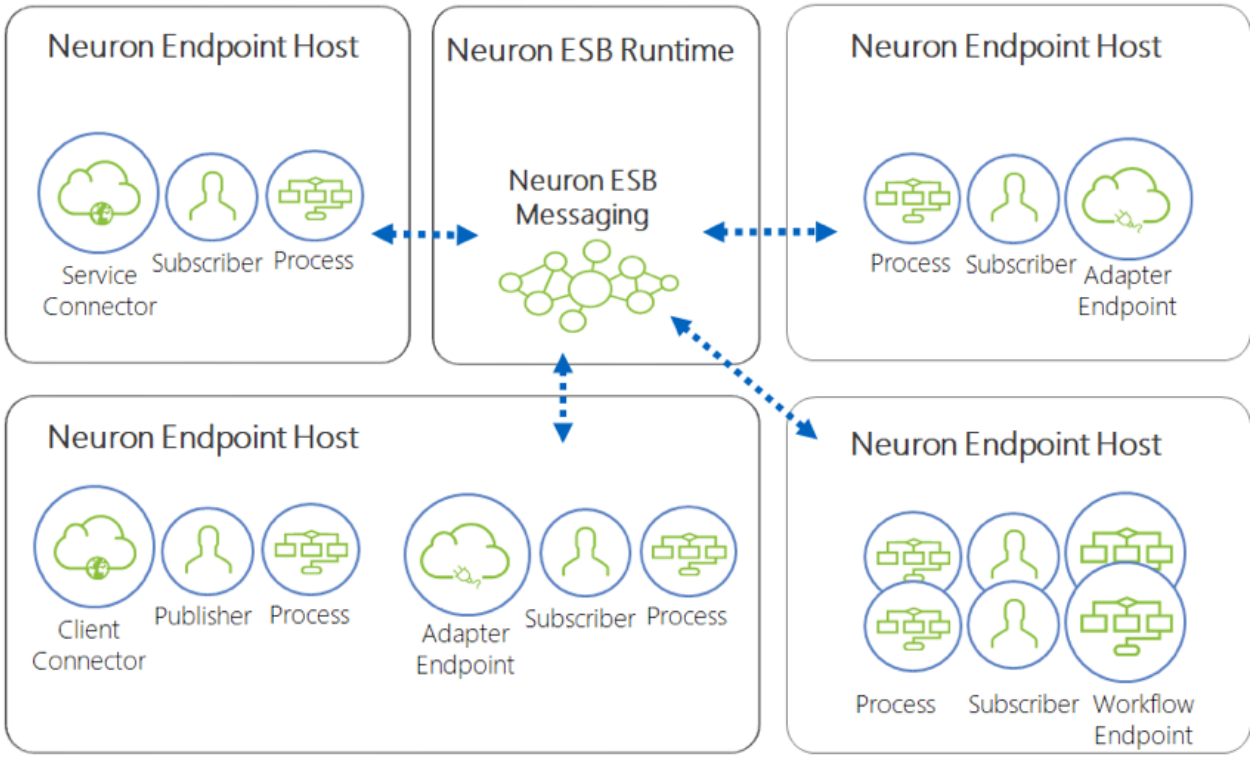
Availability Groups were designed with built-in clustering to achieve high availability and fault tolerance without the need to deploy Microsoft Windows Failover Cluster Services. Servers within a Neuron ESB Deployment Group could be assigned either Primary or Failover roles which allowed failed workflows to automatically rollover onto available servers and start where they left off, providing both resiliency and reliability for mission critical functions.

In Neuron ESB 3.7, we've refactored and extended Availability Groups to allow them to host Adapter and Service Endpoints. As a result, Availability Groups were renamed to Endpoint Hosts and have taken on a new and expanded role within Neuron ESB.

Prior to the introduction of Endpoint Hosts every Adapter and Service endpoint competed for the CPU processing of a single executable (i.e. esbservice.exe). Other competitors for those processing cycles and threads included Neuron ESB's own internal services, which would sometimes take precedent. In some cases, either an internal service or a specific endpoint could consume the majority of processing cycles, leaving other endpoints waiting for available threads to be returned to the pool before being allowed to

service requests. Additionally, even though all endpoints and internal services were encapsulated in their own .NET AppDomain, it did not preclude the possibility that one misbehaving Business Process, Adapter Endpoint or, custom code written by a customer could crash the Neuron ESB Runtime process (i.e. esbsevice.exe).

Endpoint Hosts alleviates these issues as each runs as an isolated process (i.e. NeuronEndpointHost.exe), with its own allocation of resources and threads. Using Endpoint Hosts, endpoints can be isolated into discreet, lightweight processes and deployed across multiple machines.



Workflows

Using workflow, it is possible to build business processes that can span days, weeks, or even months coordinating business activities, responding to business inputs, and integrating business systems. Neuron ESB 3.7 provides a complete Workflow hosting environment for running workflows as part of, or independent of, your ESB messaging solution.

Neuron ESB's Workflow is built on WF that was originally introduced in .NET 4.0 and improved upon in .NET 4.7. Although Neuron ESB uses WF to manage workflow execution and persistence, significant work was undertaken to make WF manageable, fault tolerant and truly enterprise-ready, including the development of the following:

- Workflow Designer
    - 84 Built-In Workflow Activities
    - Support for Custom Activities

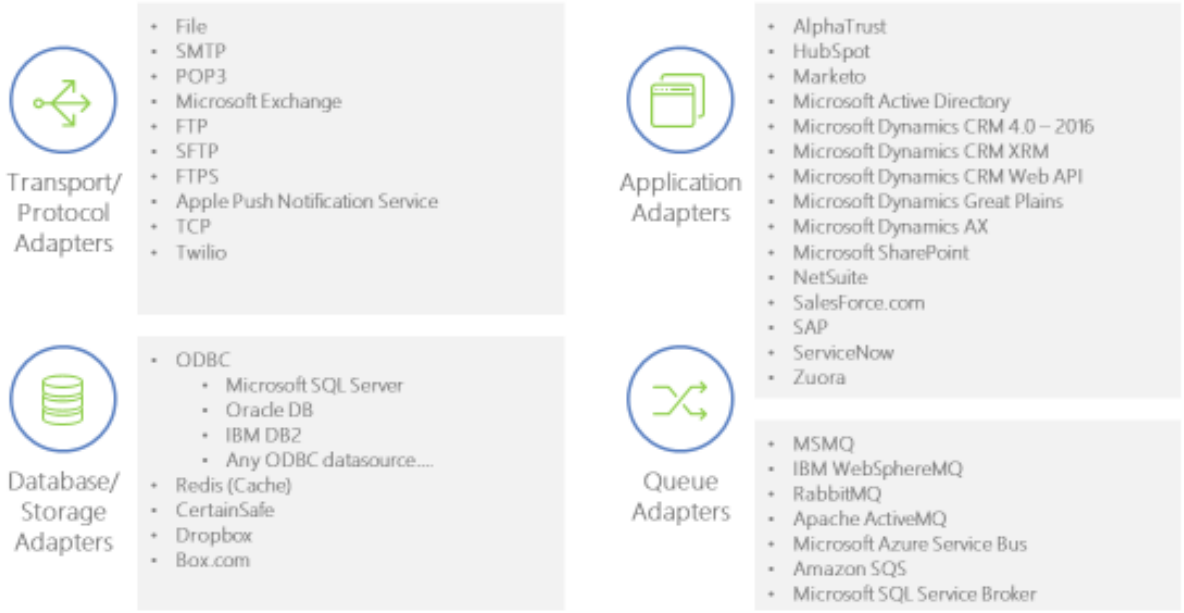
    - Workflow Simulation/Testing
- Workflow Types
    - Normal Workflow – Used when a workflow instance is executed for each message received and a response is not expected
    - Request-Reply Workflow – Used when a workflow instance is executed for each message received and a reply message is expected
    - Correlated Workflow – Used when a set of messages is to be processed by the same instance of a workflow
- Workflow Execution Environment
    - Workflow Endpoints – Associated with a specific workflow definition and acts as a subscriber to messages (also associated with a subscribing Party)
- ESB Message Integration
    - Message Auditing – Messages can be audited to the Message History table
    - Failed Message Reporting – Failed messages are tracked to the same failed messages table
    - Publish to Topics – Messages can be published back to the bus from within a workflow – either as a multicast or request/reply message.
    - Service and Adapter endpoints – Workflow activities allow the direct integration of existing service and adapter endpoints within a workflow. This allows the user to avoid publishing messages to the bus and instead sends them directly to the endpoint.
- Workflow Tracking and Playback
    - Users can observe the execution history and state transitions of the workflow
    - Users can step through the execution of the activities for the workflow, viewing both the input arguments and the values that are output by each workflow activity
    - Users are able to explore error conditions and view exception messages for errors that occurred during the workflow
- Workflow Control and Monitoring
    - Workflow Tracking provides users with control commands such as Start, Suspend, Cancel, Abort or Delete that can be used against any selected Workflow instance or group of Workflow Instances
    - A WMI Performance Counter group is available with Workflow; “Neuron Workflow Endpoints”, which exposes a number of counters, including:
        - Aborted
        - Active
        - Cancelled
        - Completed
        - CompletionRate
        - Errors
        - Idle
        - PendingEvents
        - PendingTime

        - Persisted
        - Terminated
        - WaitTime
        - Warnings
    - These WMI counters can used by third party tools for remote monitoring solutions as well as used within Microsoft Performance Monitor

Adapter Endpoints

Neuron ESB Adapters (used in adapter endpoints) are versatile and flexible bridges that connect external applications, protocols, transports or databases. Adapters can optionally be used as part of the pub/sub model, allowing them to publish messages to the bus, or receive messages from the bus. They can be configured to also work outside of the pub/sub model, either directly calling business processes when in publish mode, or directly used from within business process when in subscribe mode.

Neuron ESB ships with many adapters, some of which are listed below. An extensibility interface in .NET C# exists that allows users to build "custom" adapters that can be directly managed and hosted by Neuron ESB as Adapter Endpoints. An Adapter Endpoint is an instance of a Neuron ESB or custom Adapter that is specifically configured to bridge a target source system.



* The list above includes most of the adapters included with Neuron ESB. Neuron ESB supports the use of the Microsoft WCF LOB Adapters which can be purchased separately.

Service Endpoints

Neuron ESB supports two types of Service Endpoints: Client Connectors and Service Connectors. Client Connectors are either REST or SOAP based Services hosted by the Neuron ESB runtime. Neither IIS or

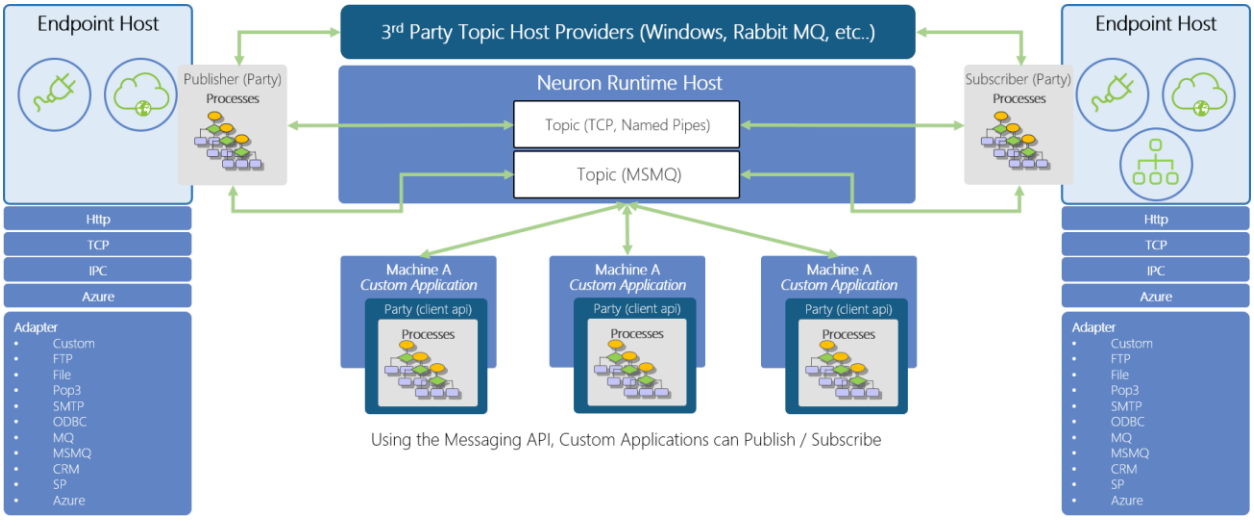
WAS is required. Client Connectors are created through the Neuron ESB Explorer User Interface and can support any Windows Communication Framework (WCF) Binding and Behavior.

In contrast to Neuron ESB Client Connectors, Service Connectors (also Service Endpoints) can optionally function as configurable subscribers to the bus and route messages to existing REST or SOAP services that preexist in the environment.

Service Endpoints function similarly to Neuron ESB Adapter Endpoints in so far as they serve as “bridges” between external SOAP/REST based services. Like adapters Neuron ESB Service Endpoints can be optionally be configured with Parties to facilitate communication to and from Topics to the external SOAP/REST services, or can be used outside of the pub/sub model.

Client connectors are publishers and therefore can call business processes directly. While service endpoints are subscribers and can be called directly from within a business process. In either case this bypasses the pub/sub model, in much the same way as can be accomplished with adapters.

The relationship between Parties, Topics, Adapter Endpoints, Service Endpoints, Processes and Workflows is depicted in the figure below.



Installation

Neuron ESB is a self-hosting runtime application and environment and installs as a Microsoft Windows Service. This means it does not require IIS or WAS (Windows Process Activation Services) to be installed to run.

Neuron ships with an installer that will install the necessary files for using RabbitMQ as a transport. We recommend using this installer even if you don’t need to install RabbitMQ. The associated MSI file should only be used for silent installations.

Neuron ESB is downloadable as a 64 bit application. When you extract the contents of the download, you will see two the necessary files for installation:

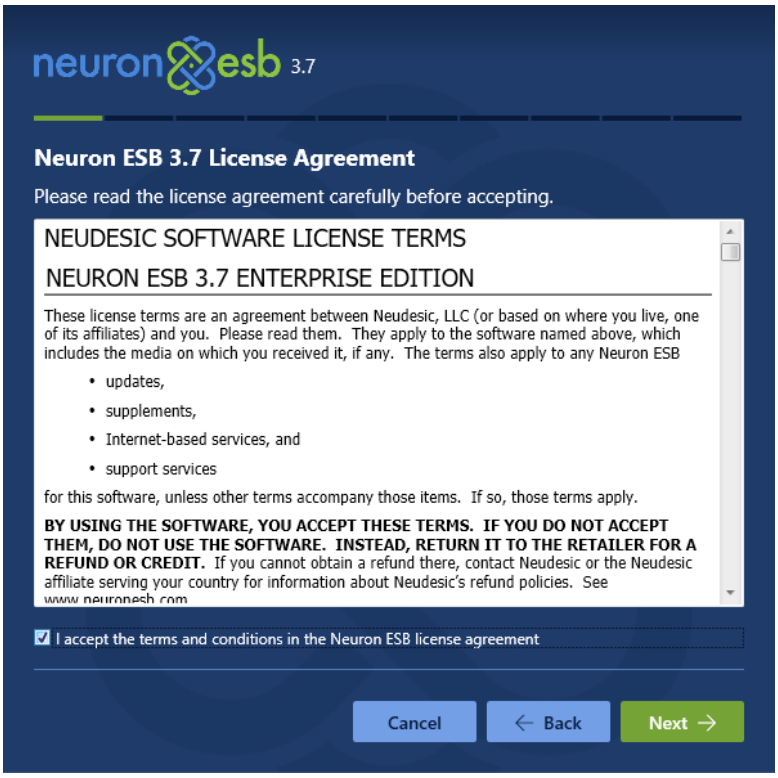
- InstallNeuronESB.exe
- Readme.html
- NeuronESB_v3_x64_Release.msi

To begin, double click InstallNeuronESB.exe and click Run.

The following screen should appear.

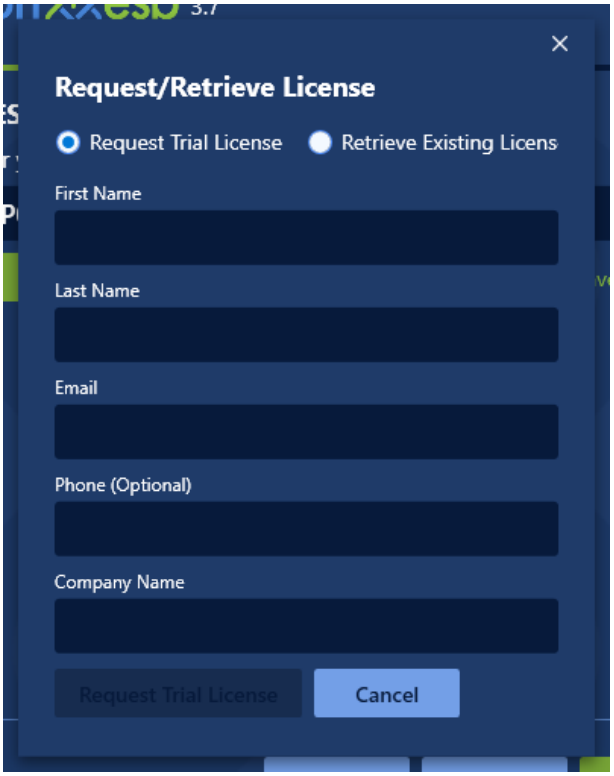
Click Next.

Accept the license agreement and click Next.



Enter the appropriate license key. Click Next.

Neuron ESB license keys are issued to you by Neuron ESB. If you do not have a license key, click the "Don't have a license?" link to request one.

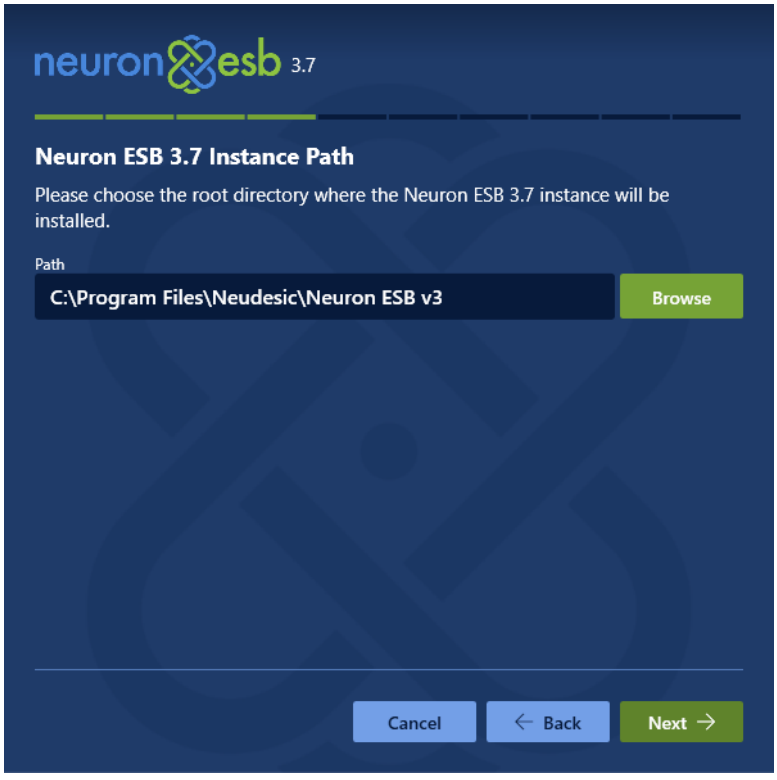


Select the software packages to install. If you want to try out RabbitMQ-based topics, select Erlang and RabbitMQ. Erlang is a prerequisite for RabbitMQ. Both of these packages are downloaded from the Neuron ESB Web server during installation. If you select them make sure your machine has an internet connection.

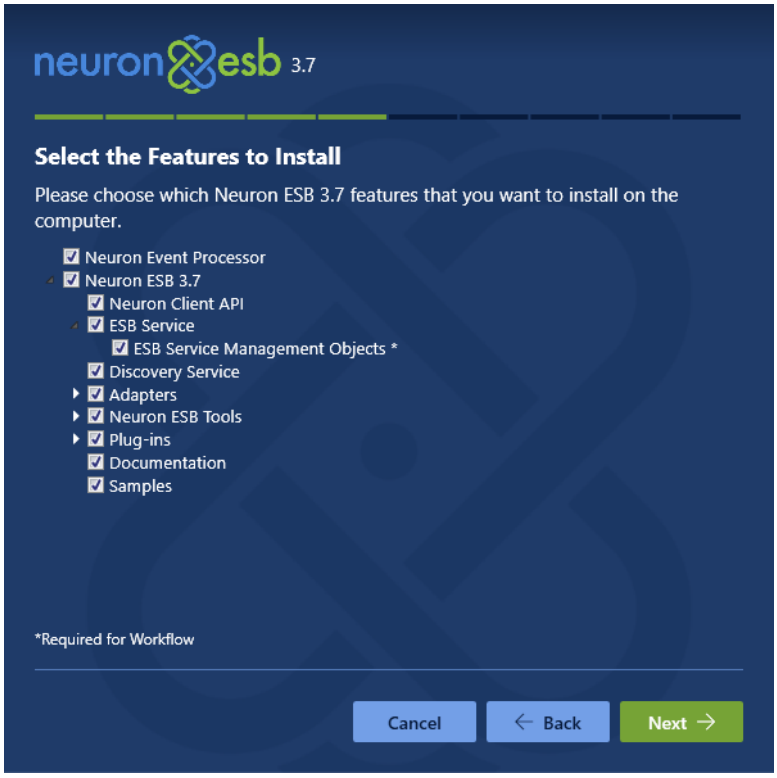
If your machine does not have an internet connection, the Erlang and RabbitMQ installation packages can be downloaded and installed manually. See the README.html that's included with the zip package for download instructions.

Click Next.

Enter the path to install Neuron ESB or leave the default value. Click Next.

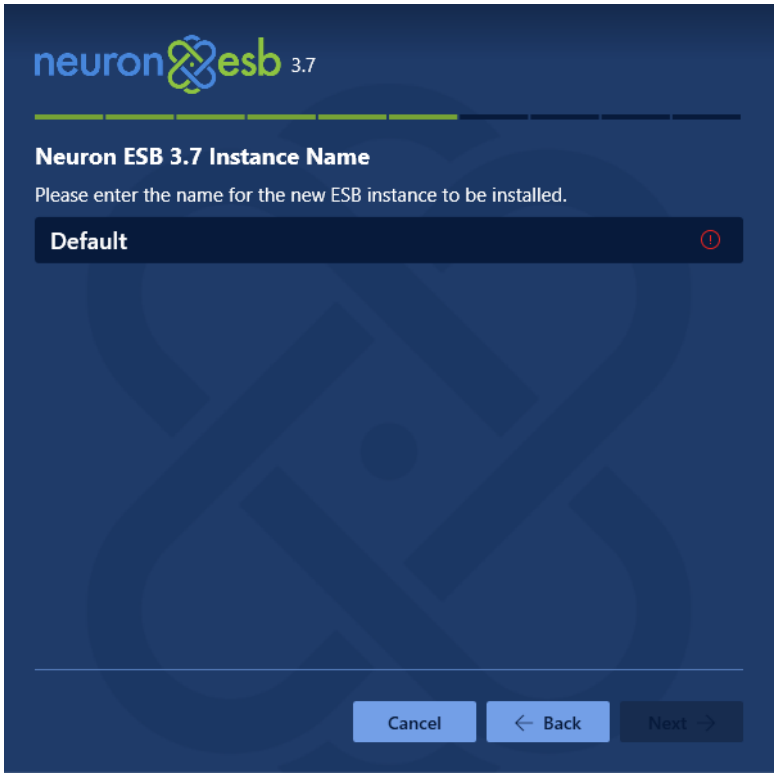


Select the options to install and click Next. Usually all options are selected.

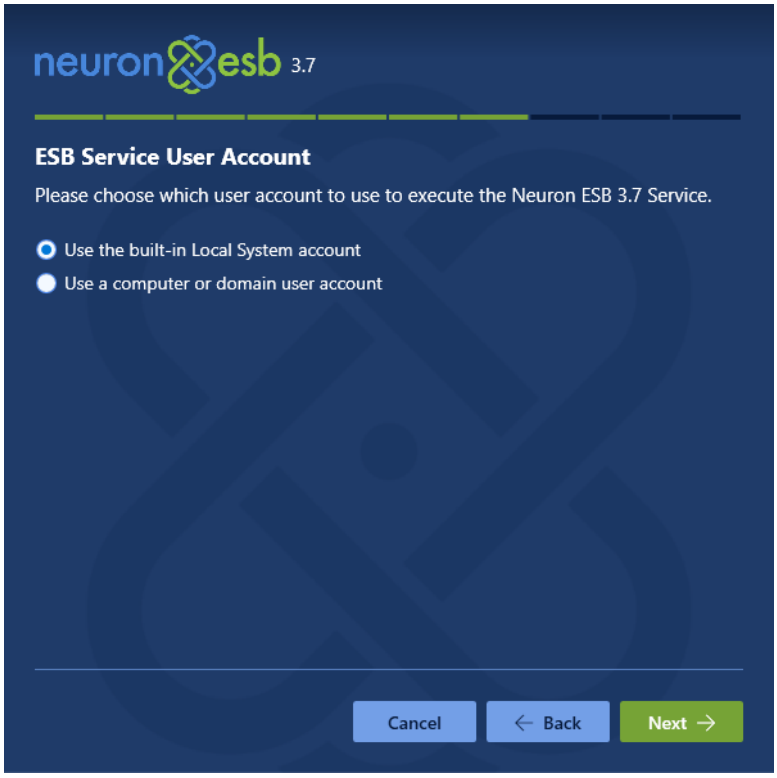


The setup program can be used to install multiple instances of Neuron ESB. This allows the hosting of multiple Neuron ESB Configurations on the same server. Since this is the first Neuron ESB installation, just leave the default value of “DEFAULT” for the instance name and click Next.

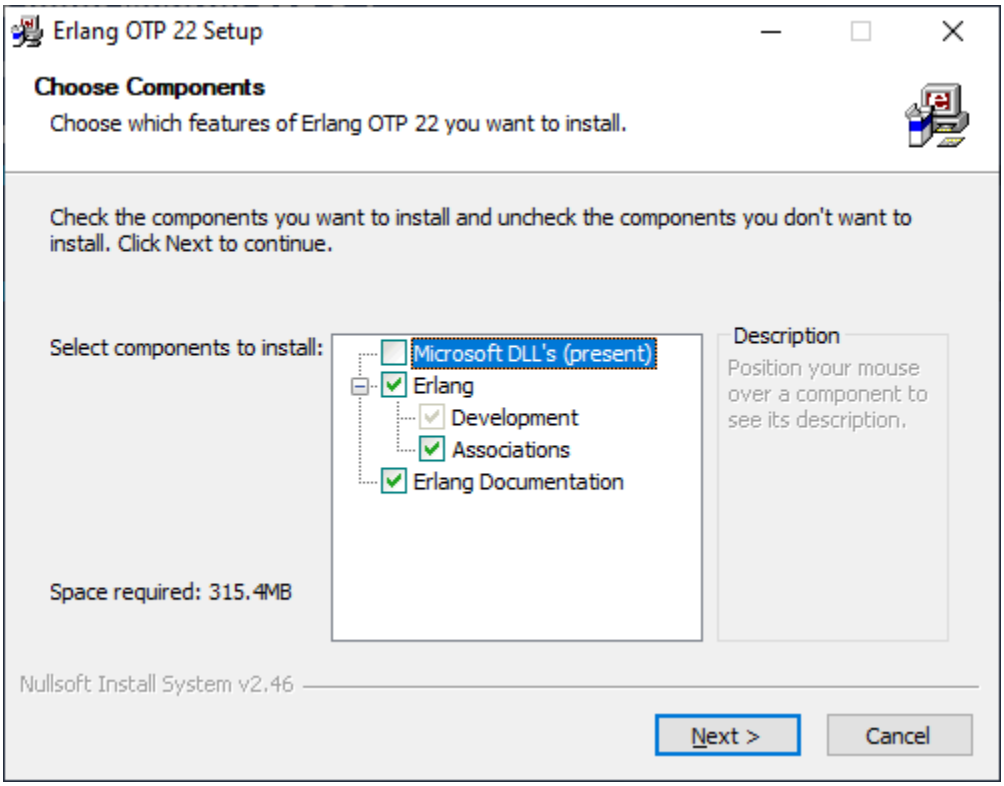
Each Neuron ESB “Instance” installed is capable of being configured to run one Configuration, commonly referred to as a “Solution” or “Application”. Solutions are created using the Neuron ESB Explorer. Users may install up to 10 “Instances” on a single machine, allowing up to 10 different Solutions to run concurrently.



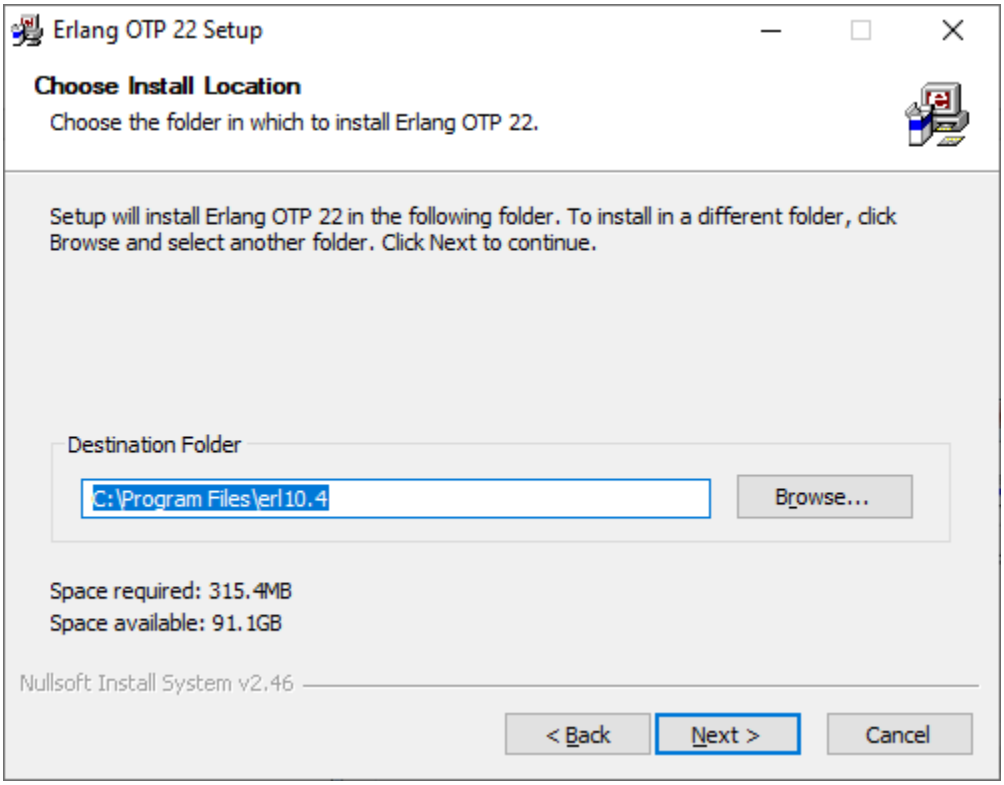
Leave the default choices in place so that Neuron ESB is running as the Local System account. Click Next to begin the installation. If you chose to install the Erlang software package, the installer will first download it and then start the installation. The Erlang installer often gets hidden behind the Neuron ESB installer. If you see this screen, look for the Erlang installer icon in the Taskbar and click on it:



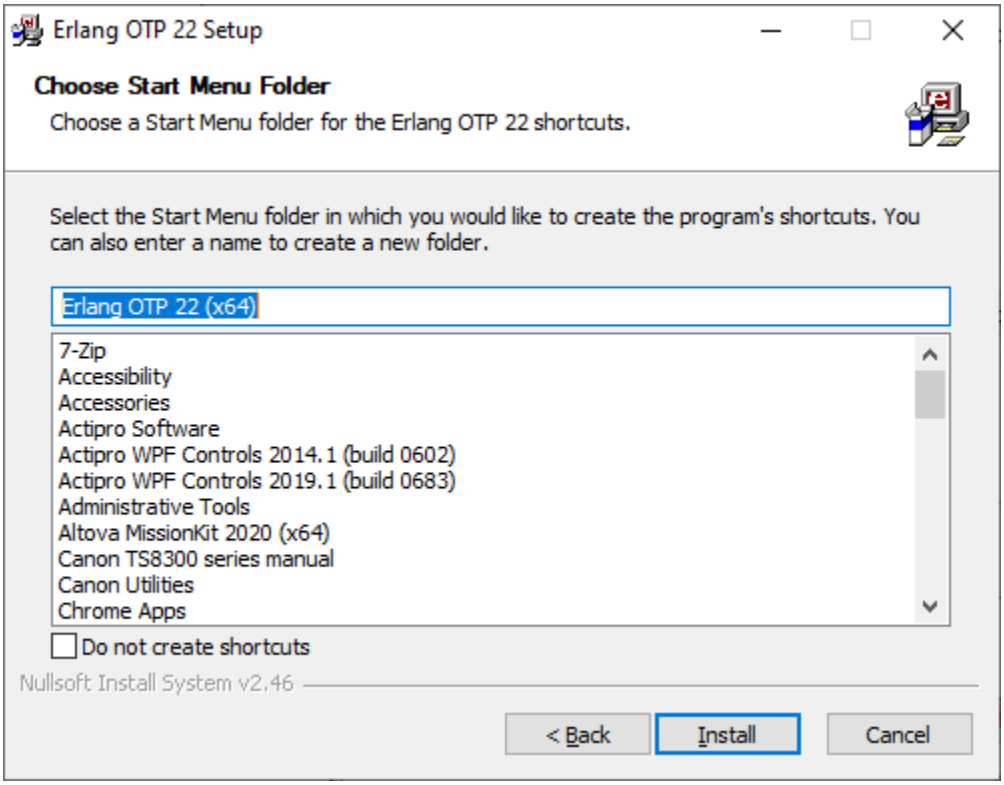
Clicking on the Erlang icon in the Taskbar brings up the Erlang installer. Select the components to install for Erlang. If the Microsoft DLL's are not present, make sure to check that box. Click Next.



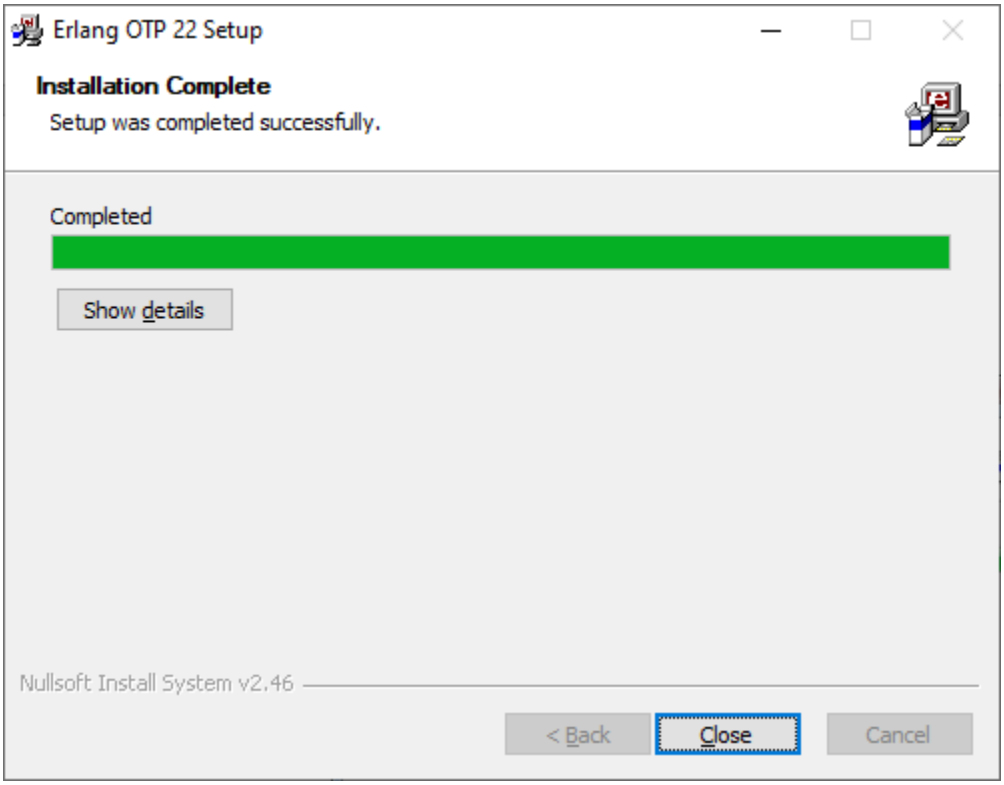
Set the installation folder. Click Next.



Choose the Start Menu Folder for the Erlang programs. Click Install.

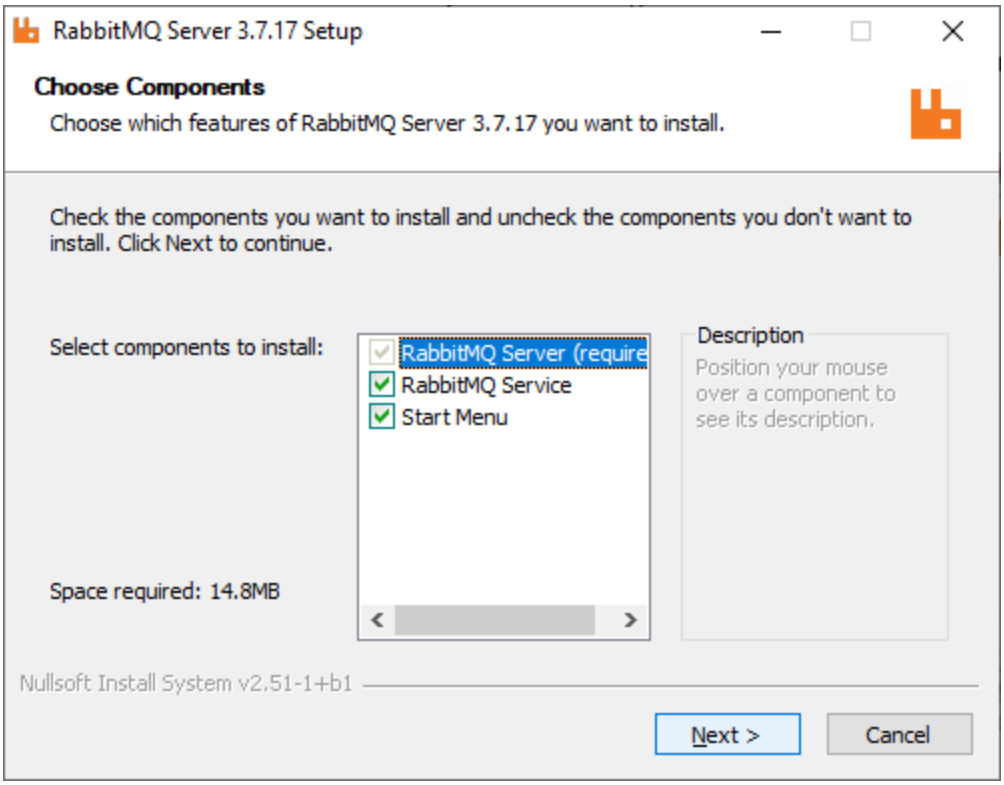


When the installation completes, click Close.

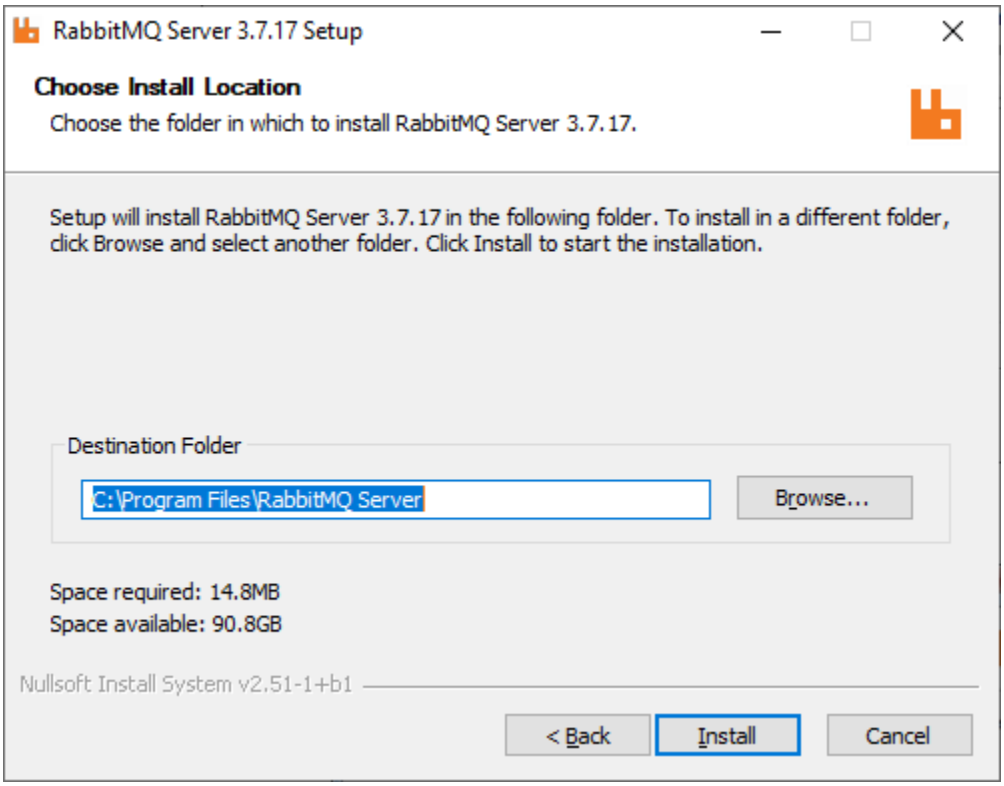


If you chose to install the RabbitMQ software package, the installer will next download it and then start the installation. The RabbitMQ installer often gets hidden behind the Neuron ESB installer.

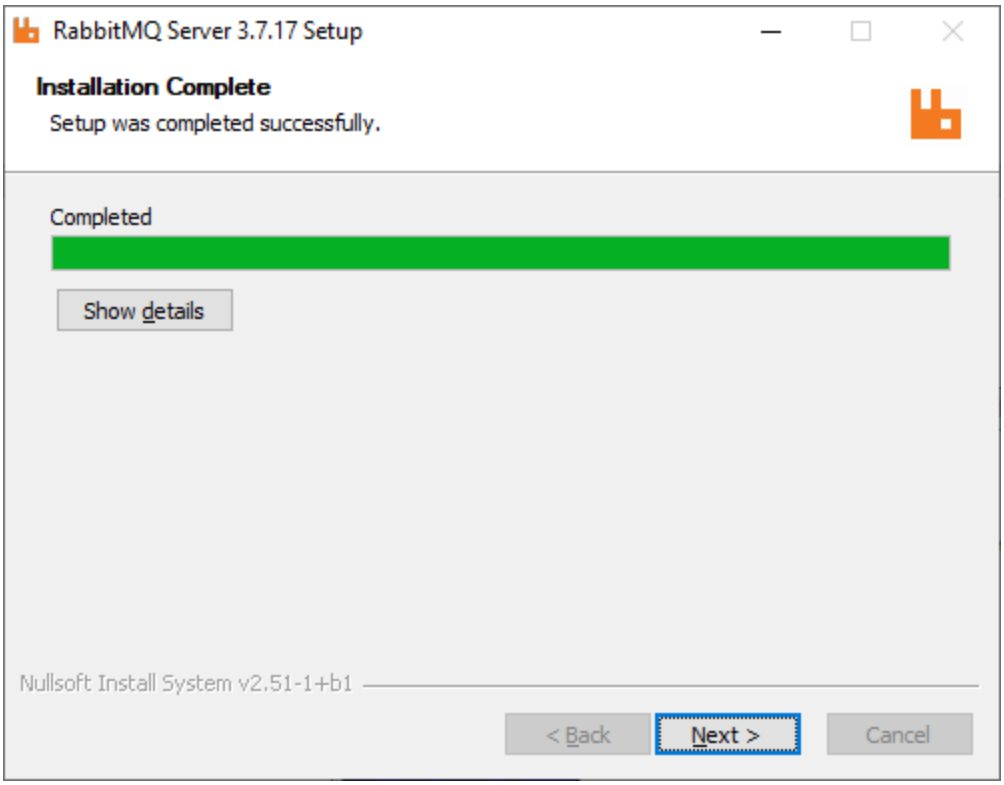
Clicking on the RabbitMQ icon in the Taskbar brings up the RabbitMQ installer. Select the components to install for RabbitMQ. Click Next.

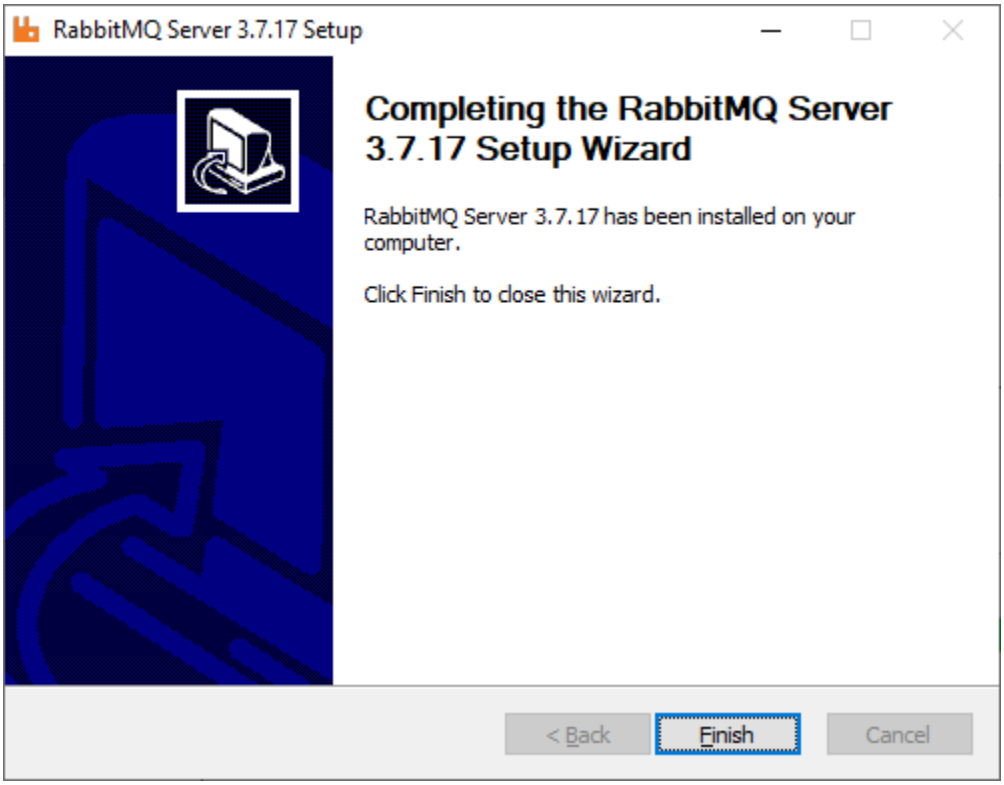


Set the RabbitMQ install location. Click Install.



When the RabbitMQ installation completes, click Next and then Finish.


RabbitMQ Server 3.7.17 Setup
Installation Complete
Setup was completed successfully.
Completed
Show details
Nullsoft Install System v2.51-1+b1
< Back
Next >
Cancel

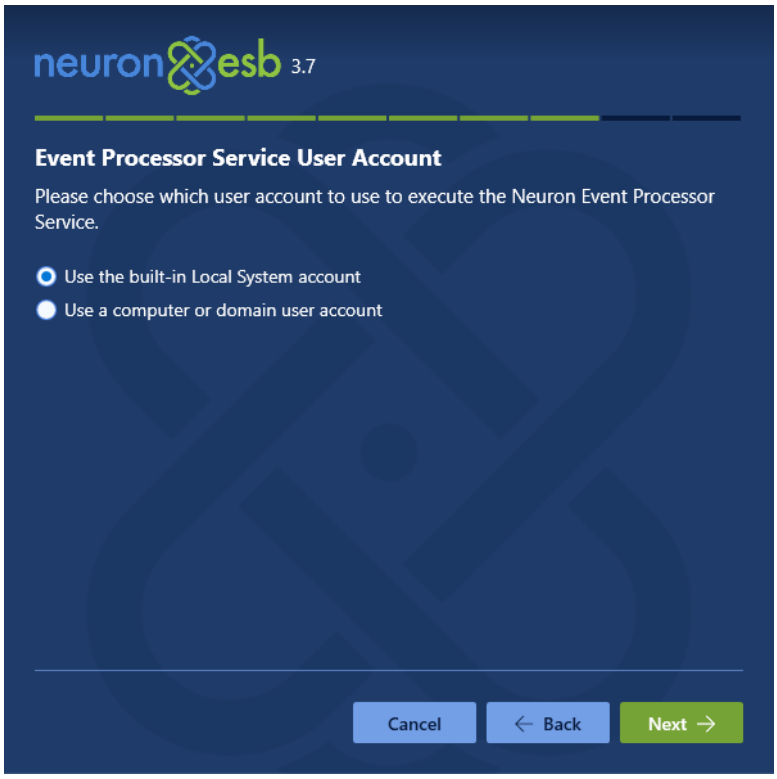

RabbitMQ Server 3.7.17 Setup
Completing the RabbitMQ Server 3.7.17 Setup Wizard
RabbitMQ Server 3.7.17 has been installed on your computer.
Click Finish to close this wizard.
< Back
Finish
Cancel

Installation of RabbitMQ Server is optional and is not required to use Neuron ESB 3.7. If RabbitMQ Server is not installed, the Neuron ESB 3.7 installer will give you the option of automatically downloading and installing Erlang and RabbitMQ Server. If you want to use the new RabbitMQ Topics, but your computer does not have Internet access, please download and install the following software packages before installing Neuron ESB 3.7. Neuron ESB 3.7 has been upgraded to support Erlang version 23 and Rabbit MQ version 3.8.5.

- *Erlang*
- *RabbitMQ Server*

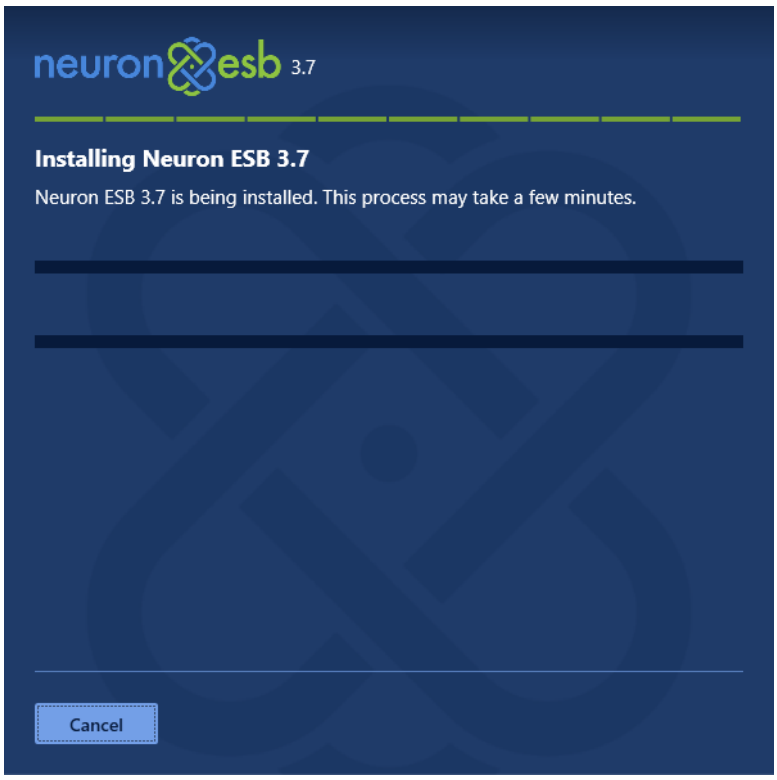
The Neuron ESB installation will continue.

Keep the default settings for the Event Processor Service User Account.

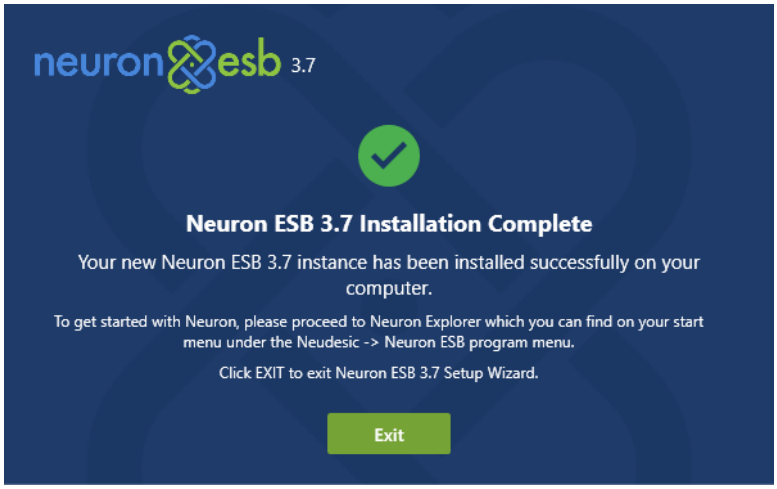


Keep the default settings for the Neuron Event Processor Port.

Neuron will continue the installation process.

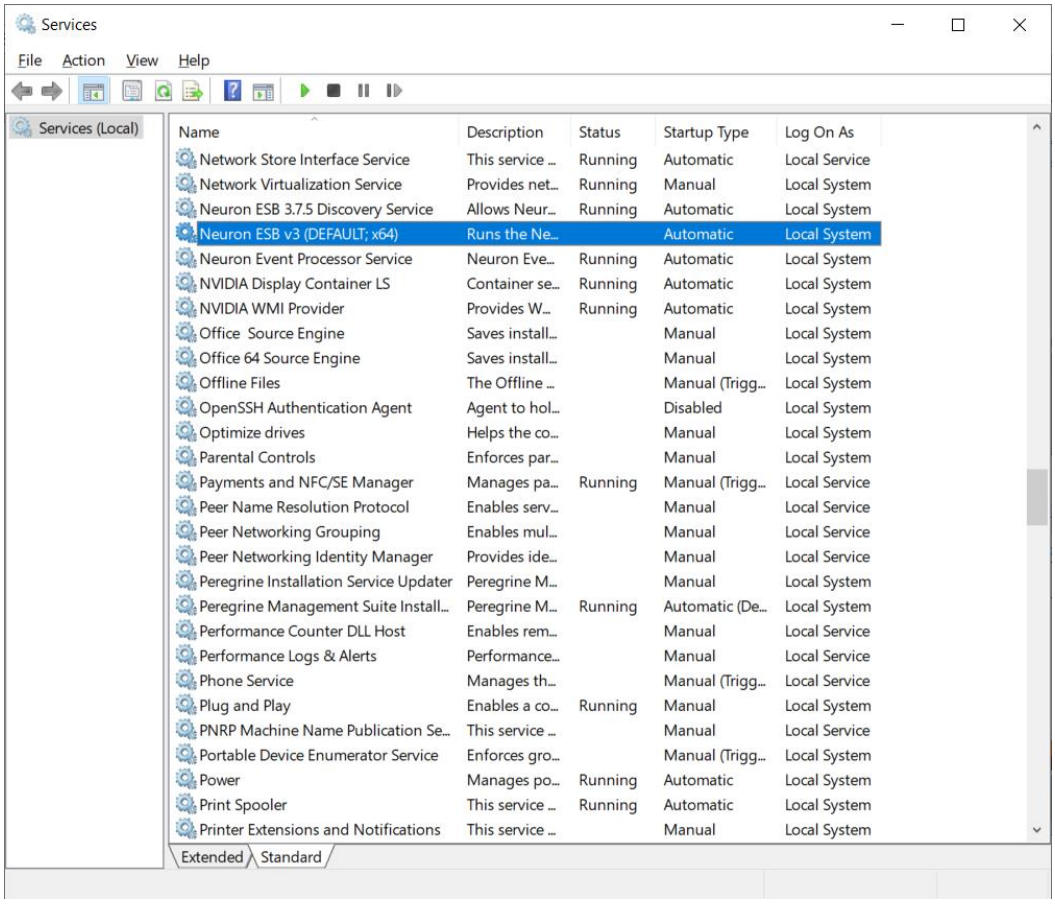


When the installation completes, click the Exit button.



Neuron ESB should now be installed.

Type services.msc into your run bar or navigate to the Services console through the admin tools. Scroll through the list of services until you see the Neuron ESB v3 (DEFAULT; {Platform}) Service

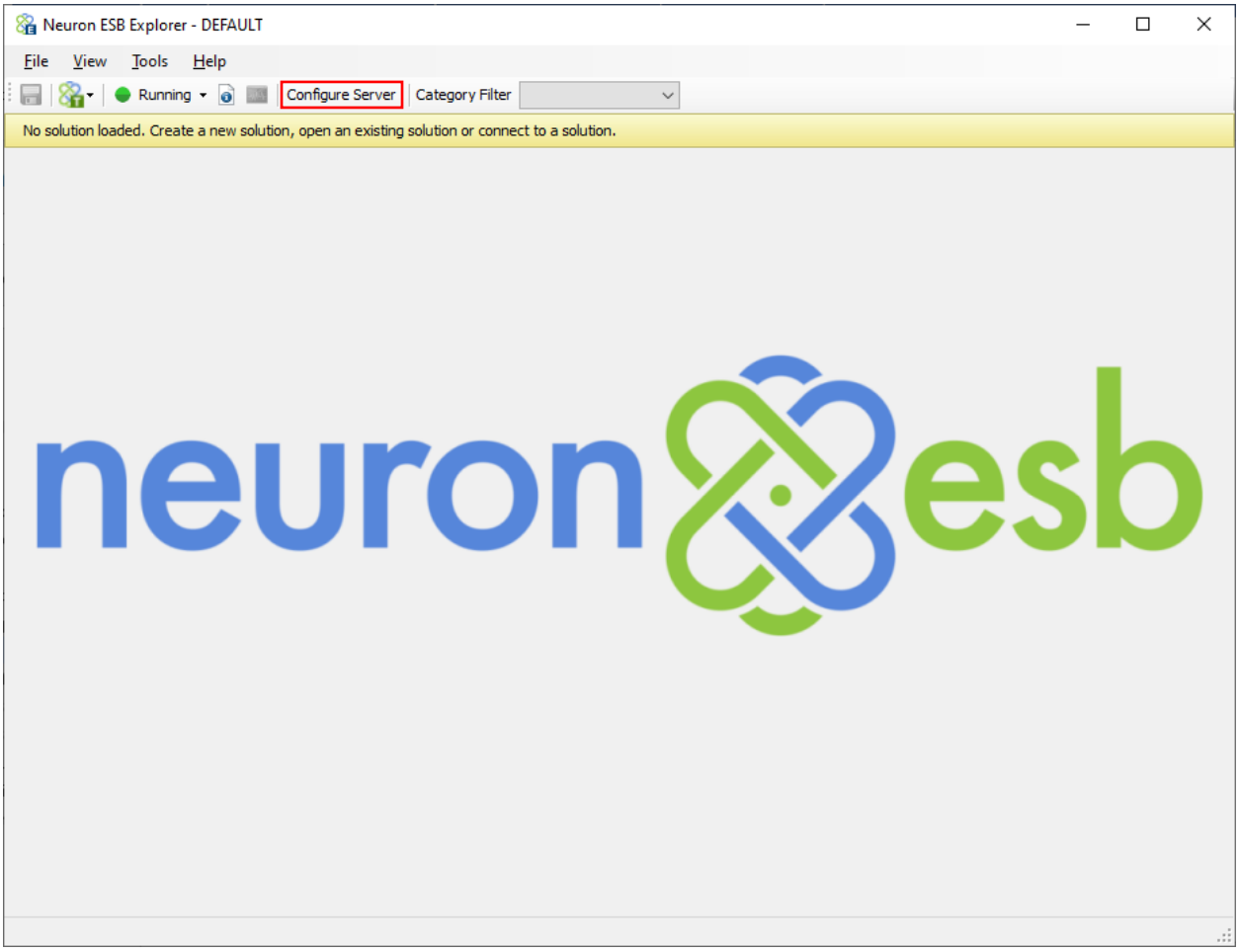


You should see that Neuron ESB is installed to begin automatically but is not yet in a started state. Do not start the service here. We will do that with Neuron ESB Explorer.

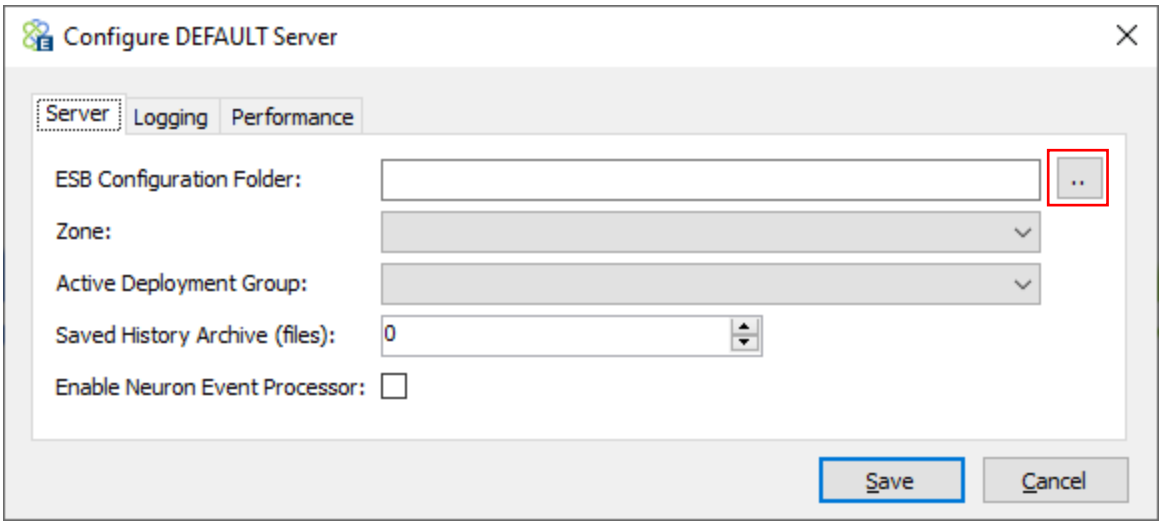
An Error will be reported if an attempt is made to start the ESB Service before it has been configured to run a solution using the Neuron ESB Explorer's Configure Server dialog.

Navigate using your Start Menu to the Neudesic folder and choose the shortcut for Neuron ESB Explorer.

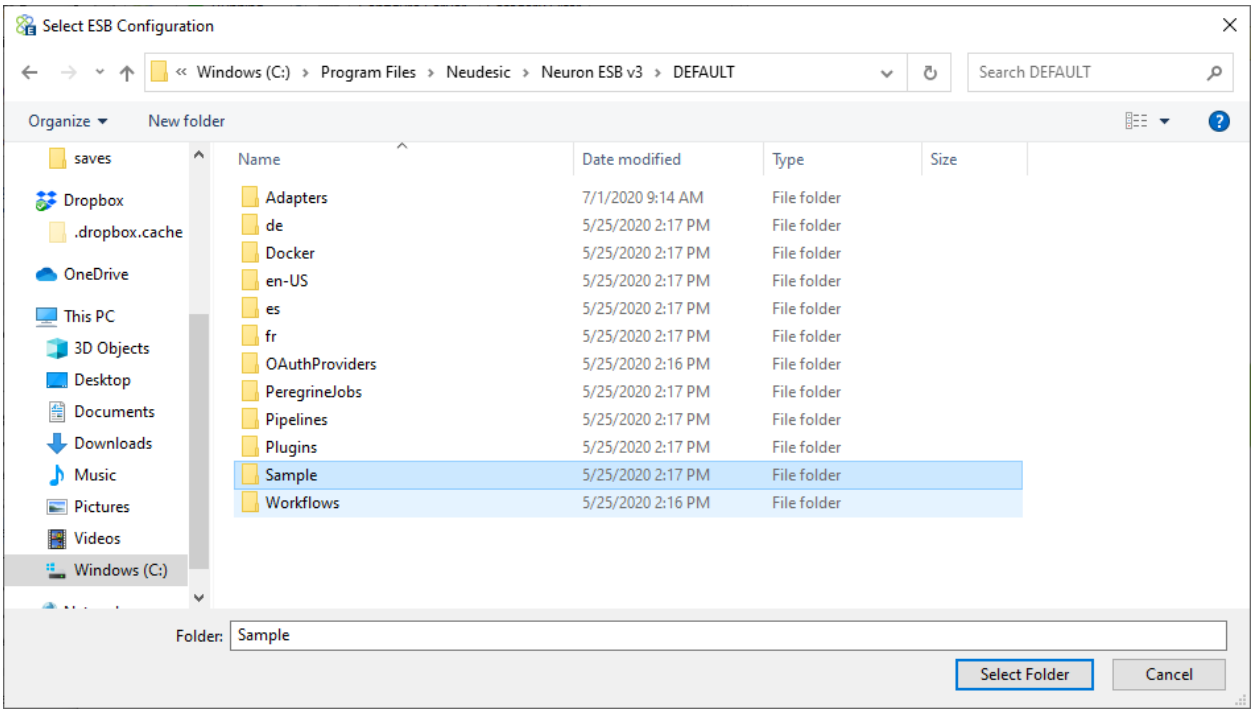
When the following screen appears, click the Configure Server button:



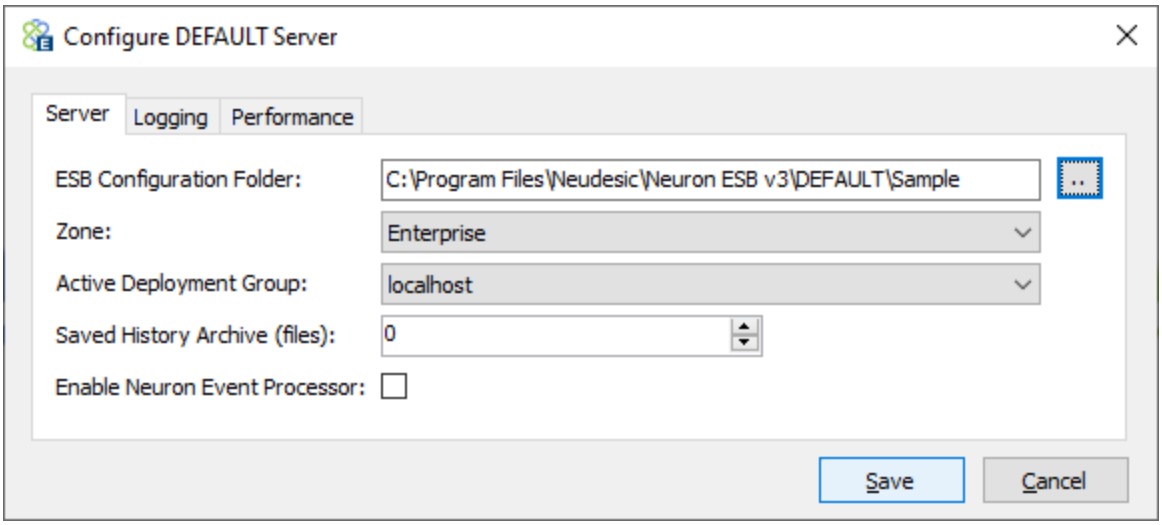
The Configure DEFAULT Server prompt will appear, which allows you to configure the Neuron ESB runtime. Neuron ESB ships with a sample configuration that you initially configure the runtime to host.



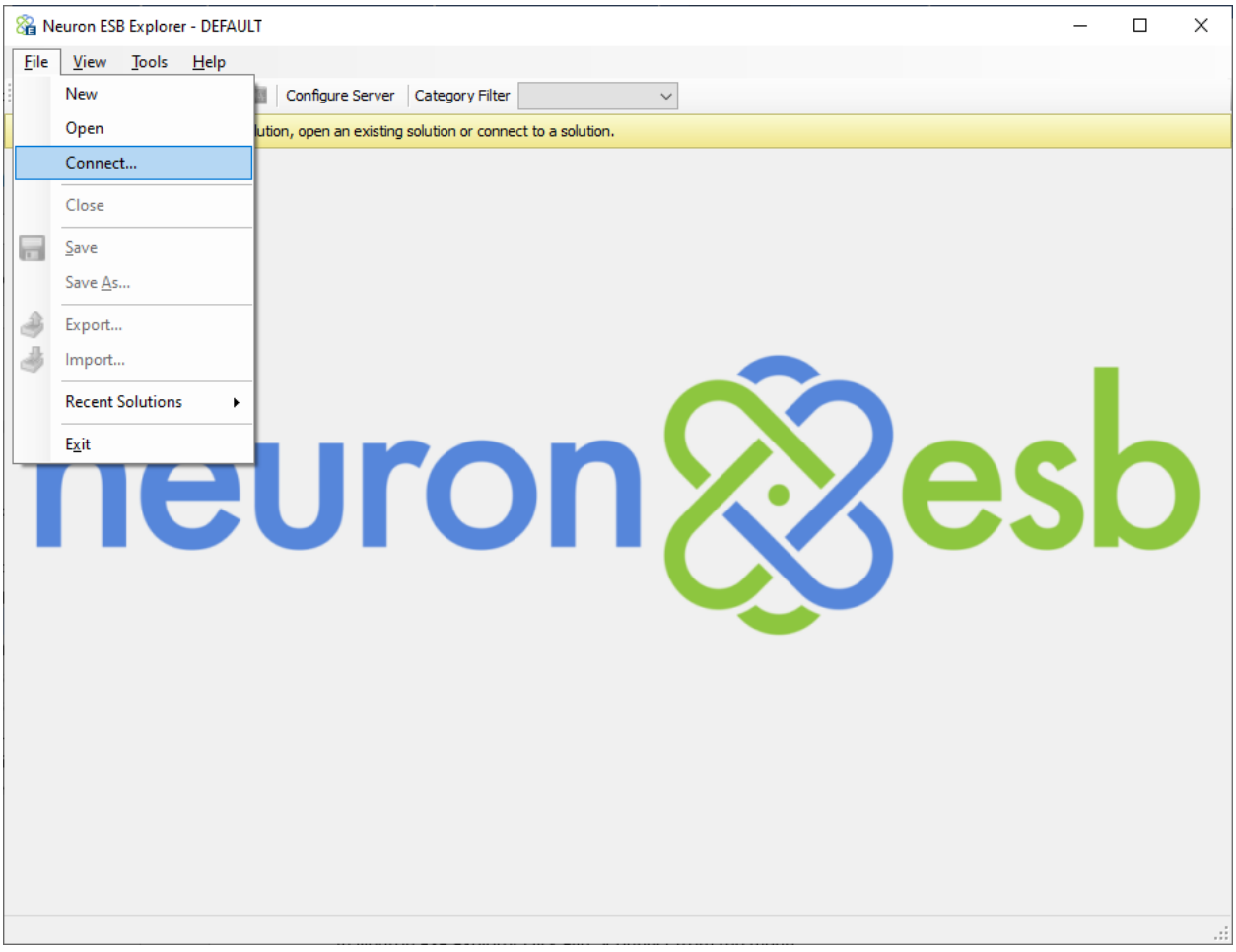
Click the ellipsis to open the folder browser, navigate to <Program Files>\Neudesic\Neuron ESB v3\DEFAULT, and select the folder “Sample”:



Click the Select Folder button.



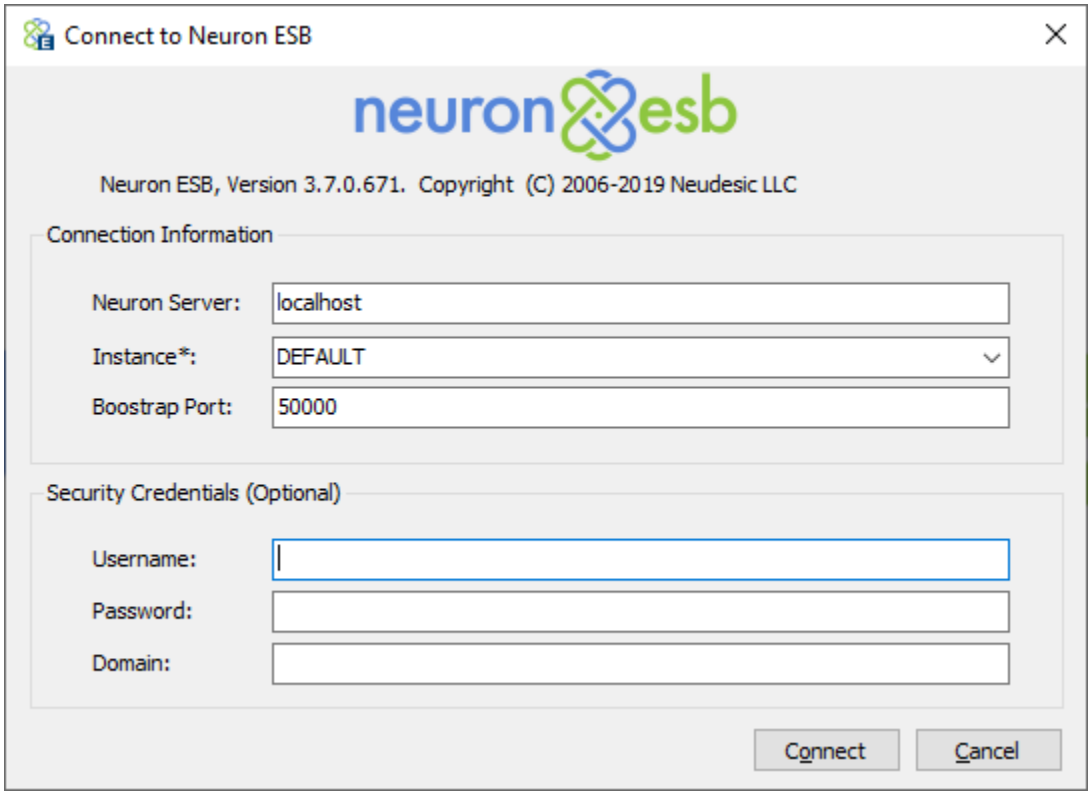
Select Enterprise from the Zone dropdown and localhost from the Active Deployment Group dropdown. Click Save.



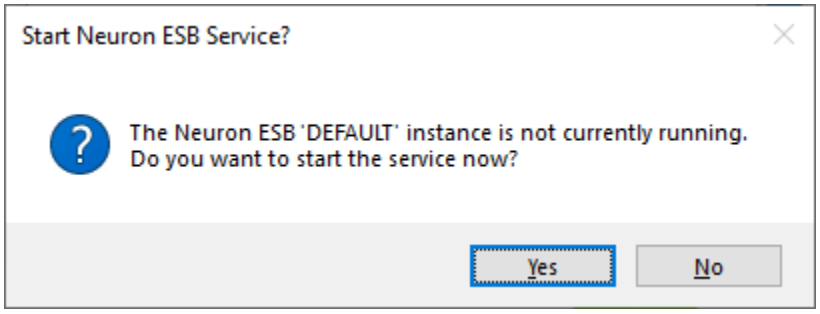
In Neuron ESB Explorer click File->Connect from the menu.

The "Connect" option should only be used when remotely monitoring Neuron ESB running solutions. In all other cases File->Open should be used to directly load the solution configuration within the Neuron ESB Explorer for editing. The Connect option is NOT supported for editing solutions.

The Connect to Neuron ESB prompt will appear:

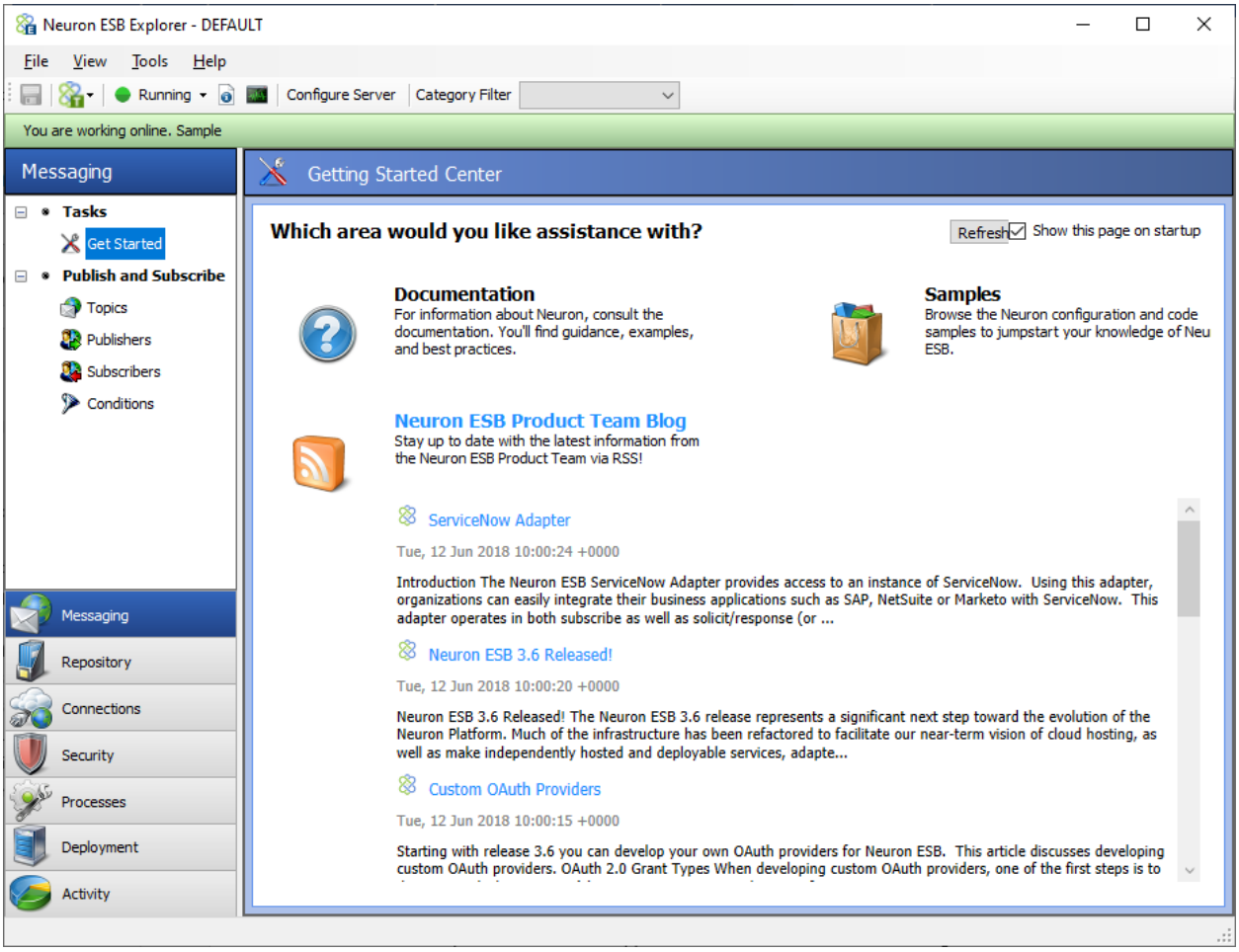


Leave the default setting set to "Override". Click Connect. A prompt should appear like the following provided you did not use Services Manager to start the Neuron ESB service directly:



Choose Yes.

You should see the following main screen of Neuron ESB Explorer after start completes



Congratulations! You have installed Neuron ESB and it is running.

Before we continue to the next section of this training, we will describe what Neuron ESB Explorer is and what it can be used for.

Neuron ESB Explorer is a Windows application that interacts with the running service and the configuration folder which drives it. The configuration folder contains a collection of XML files which represent the Neuron ESB artifacts such as Topics, Endpoints and Parties that are part of the Neuron ESB solution.

On Windows Server 2019, 2016, 2012, Windows 2008 R2, Windows 10 or any OS with UAC enabled you must explicitly run Neuron ESB Explorer as administrator to change settings.

The Neuron ESB Explorer may connect to a running instance using “Connect” mode or it may work using the “Open” mode. The first method affects and communicates with the running instance of the Neuron ESB runtime service and the ESB configuration that is has loaded into memory on startup. This is used

solely for remote monitoring and should not be used when editing the solution. The “Open” mode allows opening of and manipulation of any Neuron ESB configuration including the currently loaded configuration. There may be a few seconds delay between saving the opened configuration and seeing the changes affect the ESB Service.

You can also create a “New” configuration, which is identical to “Open” mode except it provides a completely new Neuron ESB configuration without any configured Topics or Parties.

The Neuron ESB Explorer navigation is accomplished through a combination of using the Navigation Tabs to the left of the main panel and by using the Menu items along the top.

A brief synopsis of the Navigation Tabs follows (detailed explanations of each section can be found in the Neuron ESB documentation or other training materials):

- Messaging – This is where you set up Parties, Topics and Conditions. Attaching and removing Processes to and from Parties and creating Subscriptions is done here as well as choosing Transports for Topics. Every Neuron ESB solution begins here.
- Repository – This is an inert data store of XML including imported Schemas, XSLT, XML and WSDL Documents. The documents in this store can be accessed at runtime by Process steps. For example, the Validate-Schema and Transform-Xslt steps can be configured to dynamically reference documents from the repository at runtime for validation and transformation. Client connectors can be configured to return WSDLs that are stored in the repository.
- Connections – This is where Service Endpoints such as Client Connectors and Service Connectors as well as Adapters, Adapter Endpoints, and Workflow Endpoints are configured. This is probably the most used tab in Neuron by developers after they have setup their Topics and Parties.
- Security – This is where encryption keys, certificate configurations and user names and passwords are configured. Impersonation credentials for adapters are also configured here.
- Processes – This is where Processes and Workflows are built. They can also be tested and debugged using the Process tester or Workflow Designer. This is also where you can import an existing workflow.
- Deployment – This is where basic and advanced deployment options are chosen including Zones, Deployment Groups, Endpoint Hosts and Environment Variables. By default, most of these choices and settings are configured for you when you create a Neuron ESB configuration. However, certain advanced scenarios such as server farms or housing multiple deployment groups will require changing these settings. Additionally, resources that may be used within the solution such as Neuron ESB’s underlying queues and the Neuron ESB Audit Database as well as all the Neuron ESB service runtime instances, both local and on remote machines, can be managed here. Note: To use Workflows a Database must be configured.
- Activity – This tab can be used to access real time message functionality, endpoint health of Client Connectors, Service Connectors, Topics, Adapter Endpoints and Workflow Endpoints. For Audited topics you can see message history and failed message history. Workflows instances and their current state can be seen from Workflow Tracking.

The Menu items along the top of immediate interest are:

- File – This allows users to open (offline) existing configurations, close, save and create new configurations. Users can also connect to running instances (online) and import/export Neuron ESB configurations. There is also a Most Recently Used list implemented that will saves the most recently opened solutions.
- View – This allows you to see a running manifest of the changes made to the open configuration since the last save. It also allows you to navigate just to the Getting Started screen and launches the Samples explorer
- Tools – This provides a shortcut to launching Neuron ESB Test Clients, various Windows management utilities and modify the Neuron license key
- Launch Test Client – Provides a shortcut to quickly launch up to 4 Neuron ESB Test Clients
- Running dropdown – This allows recycling of the Neuron ESB service from Neuron Explorer instead of using the Service Control Manager
- Event Log – Opens the Neuron ESB v3 Event Log on the local server
- Endpoint Health – Provide a shortcut to quickly navigate to the Endpoint Health monitor in the Activity tab
- Configure Server – This brings up a dialog which allows you to manipulate settings in the .config file for the local Neuron ESB Windows Service. Any change here requires a full service restart. This is where the Neuron ESB Configuration Folder, deployment group, logging options and performance options can be set and modified. Configure Server is also accessible by navigating to the Deployment ->Manage ->Servers screen.
- Category Filter – Allows you to filter which entities are displayed by their category

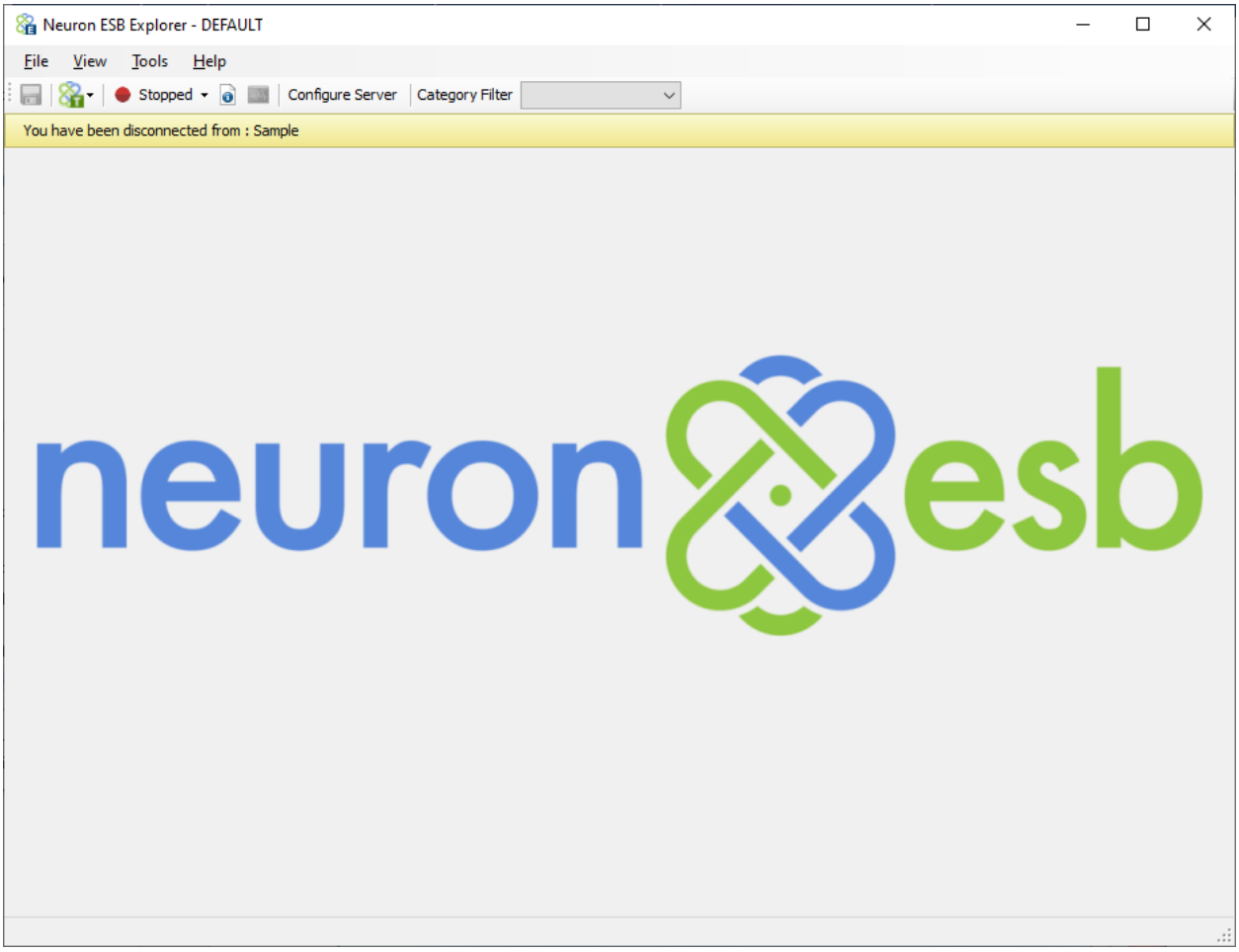
Exercise – Configuring Publish/Subscribe

Neuron ESB Configuration

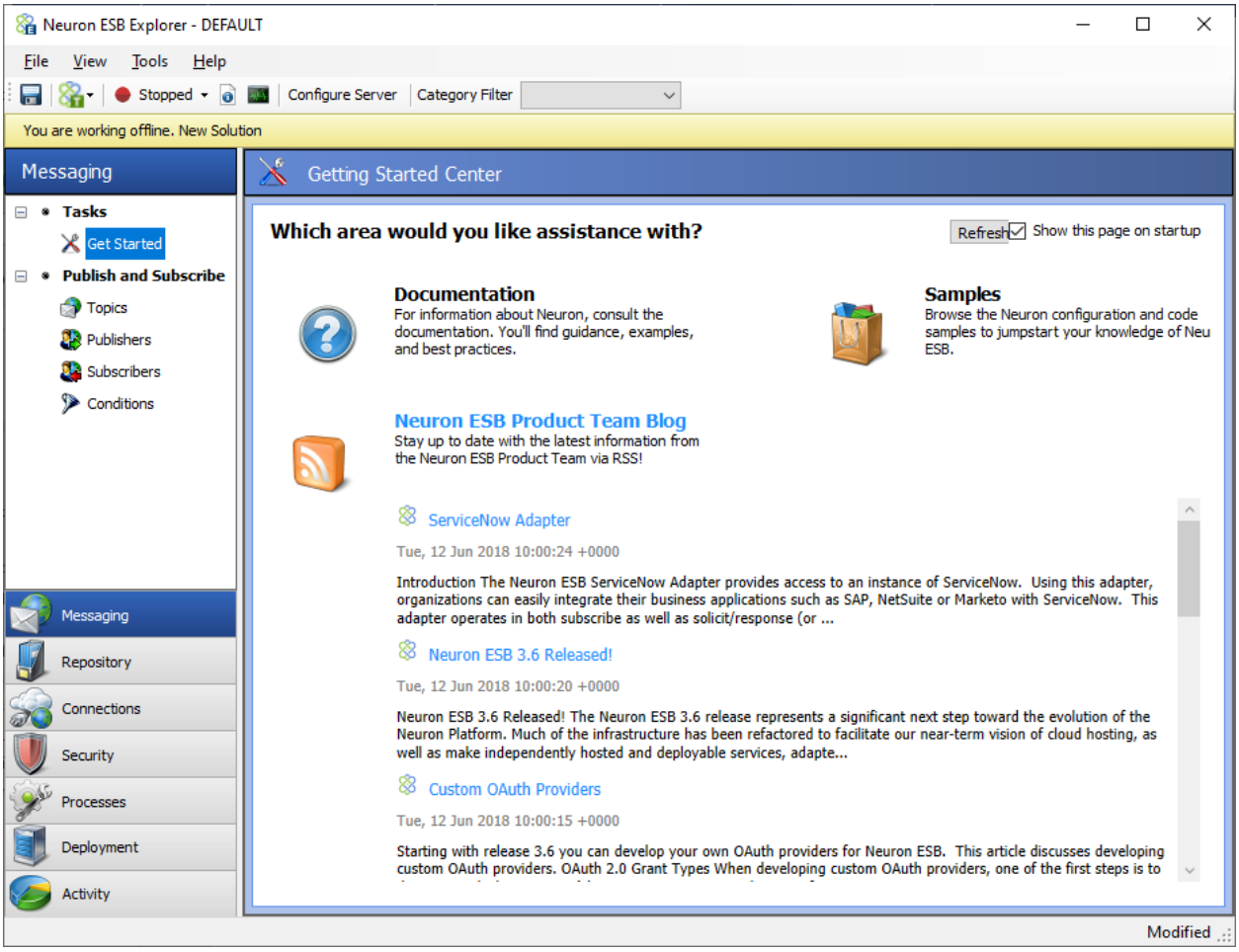
We will now create a Neuron ESB configuration (Solution/Application) from scratch and set up basic communications. If Neuron ESB Explorer is still running, choose the File option on the menu and select Close. If you have closed the Neuron ESB Explorer, then navigate to the shortcut using your Windows Start Menu.

Using either path will bring you to the default view in Neuron ESB Explorer. Choose the “New” option from the File menu.

You should see something like below:



Select File->New and you should see this:

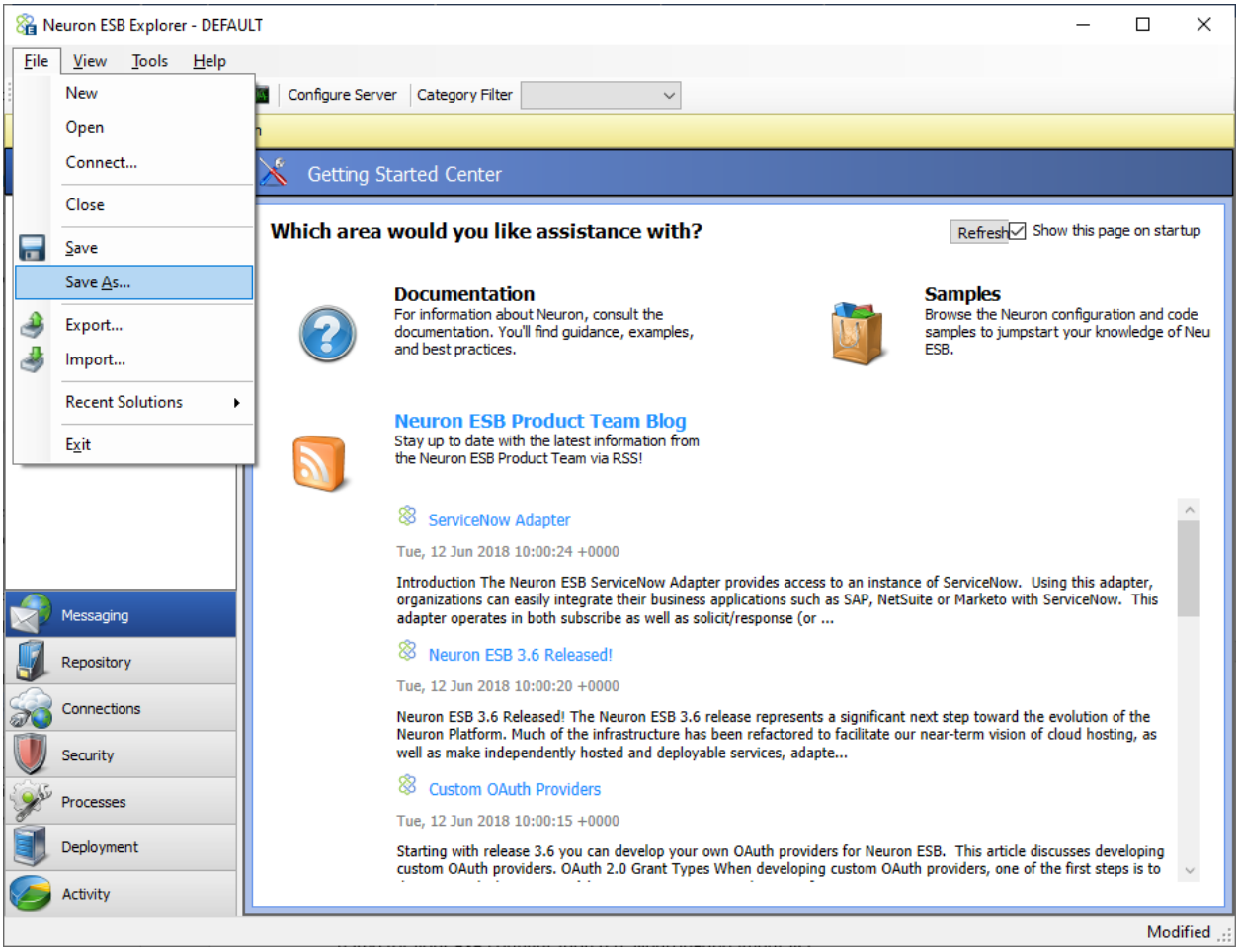


Notice that the Status Bar under the Menu items and above the Navigation Tabs is yellow. Also notice that it says New Solution.

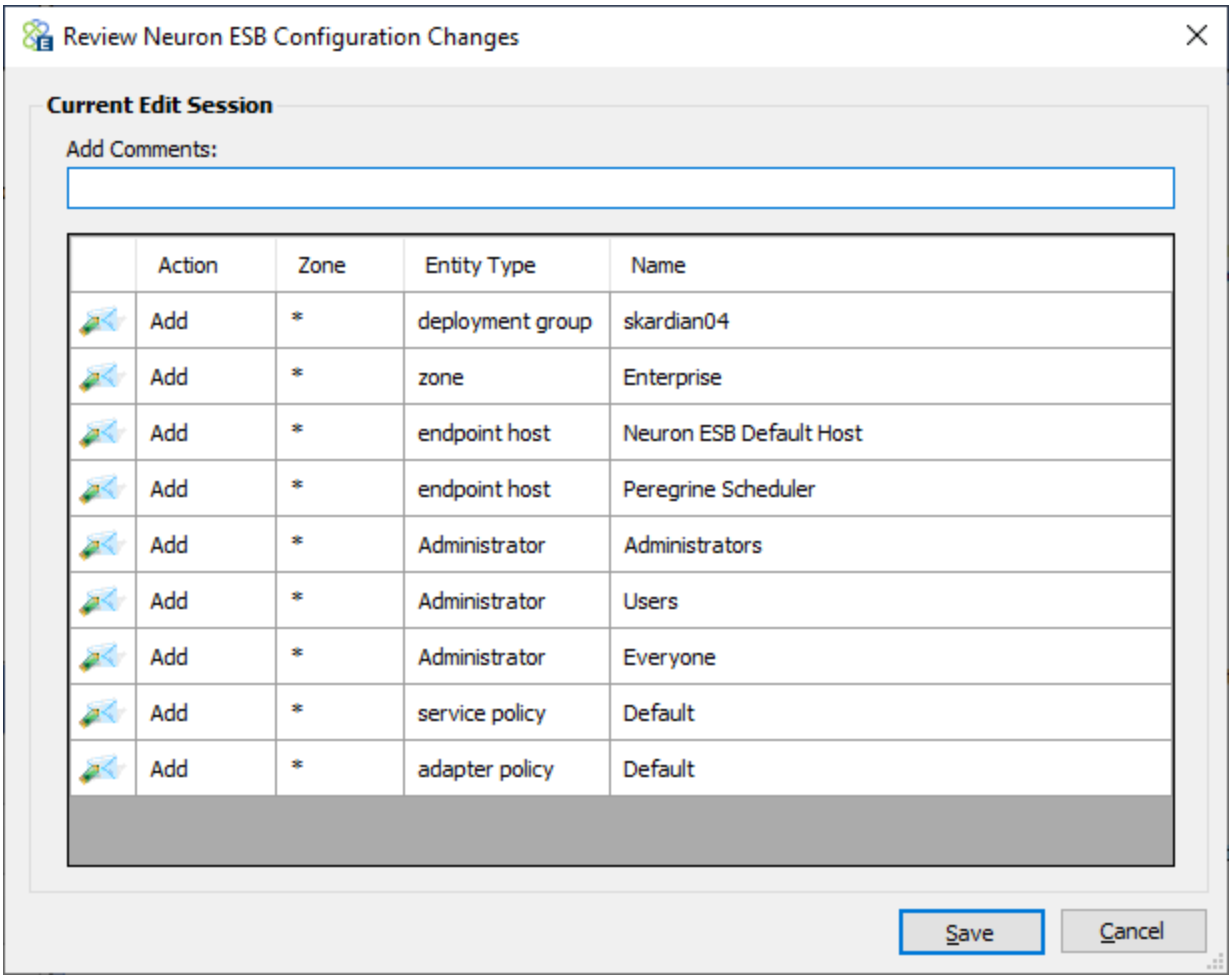
NOTE: A Yellow Bar indicates that the Neuron ESB Explorer has loaded an ESB configuration in “Offline” mode. Whereas a Green Bar indicates that the ESB Configuration has been loaded from a running Neuron ESB runtime service instance, i.e. in “Online” mode.

At this moment we are working in memory. No actual ESB configuration entities have been created yet. The first thing we’re going to do is save the ESB configuration to disk.

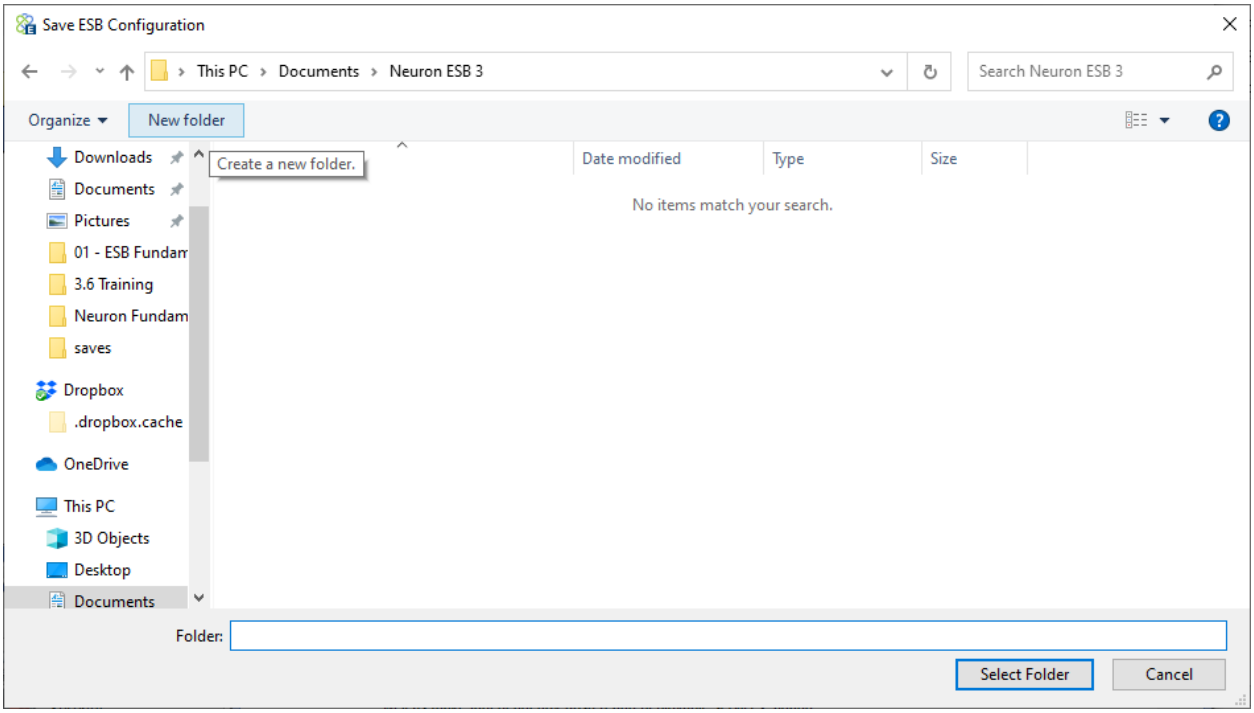
Choose File from the top-level Menu Items and then choose Save As…



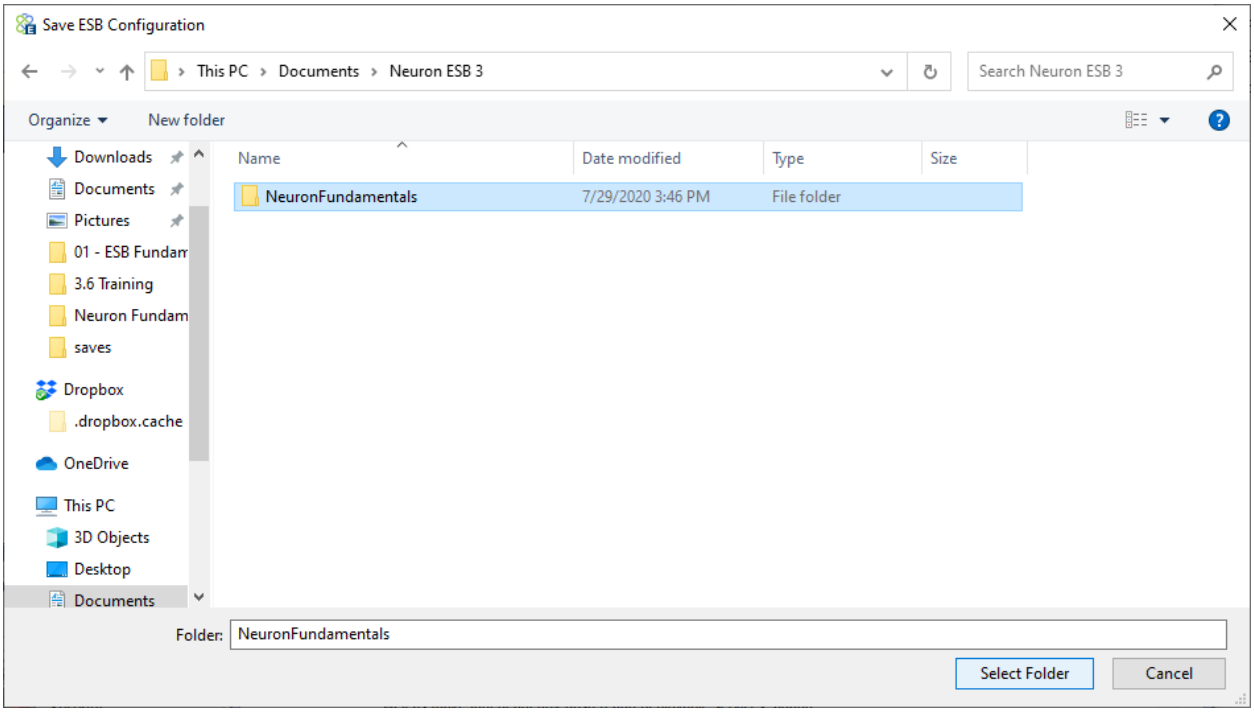
Next you will see a manifest of the changes that were made like the image below. Even though you haven't added any entities yet, some are automatically created when you create a new configuration:



Click Save, and a standard Windows File Dialog appears:

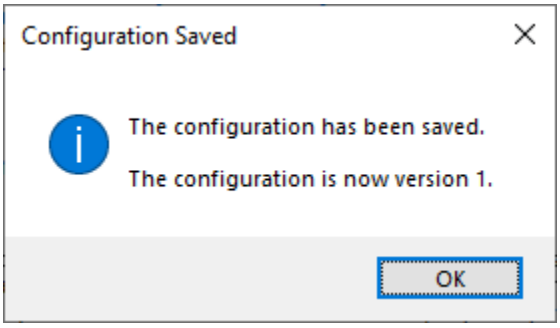


The Neuron ESB configuration is stored in a folder. Prior to saving you need to create the folder that will hold your configuration. Click the New Folder icon in the dialog (highlighted in blue above) and enter a name for your ESB configuration (i.e. NeuronFundamentals).

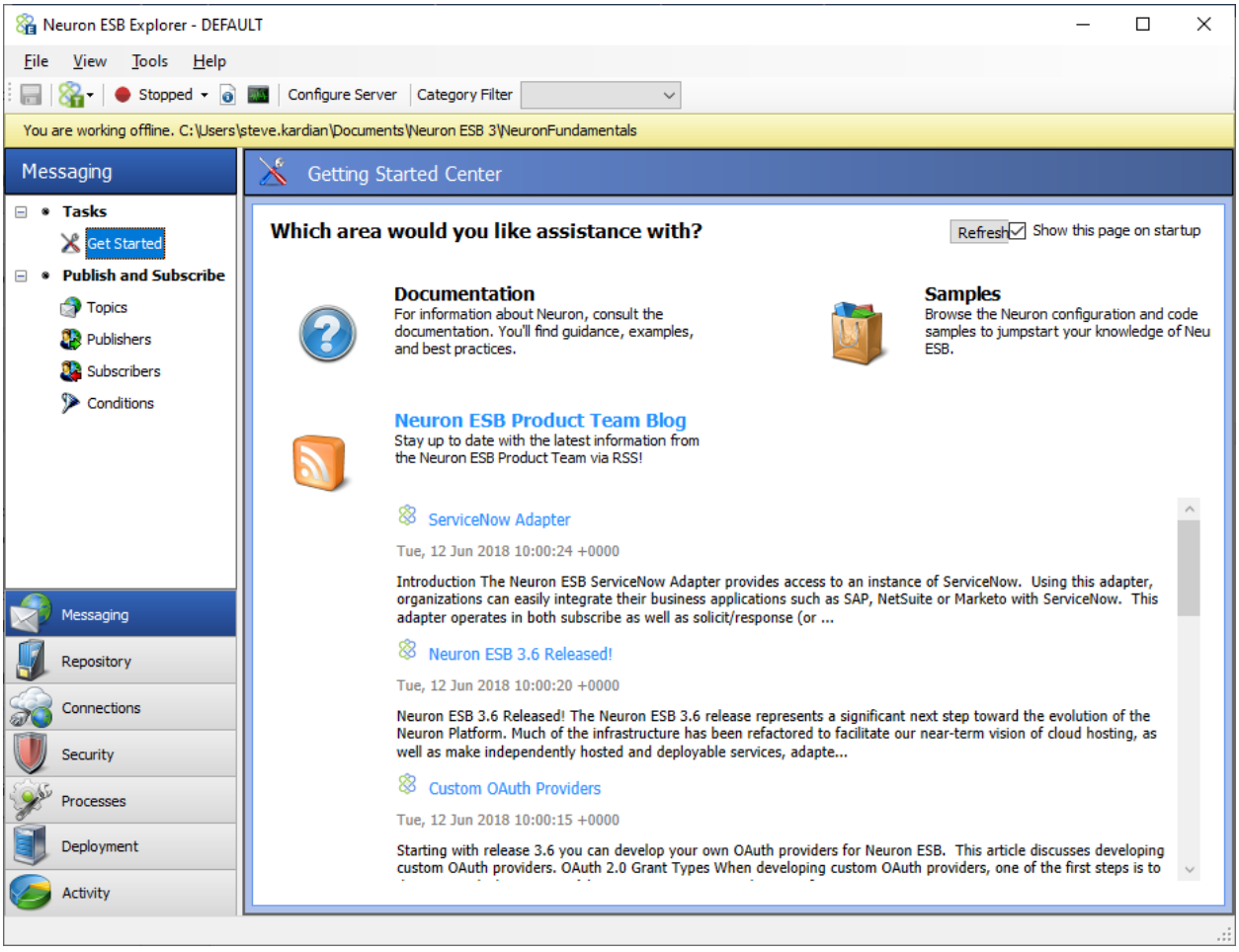


Finally, select the folder and click the Select Folder button.

A prompt is displayed indicating the configuration was saved and which version you are on:



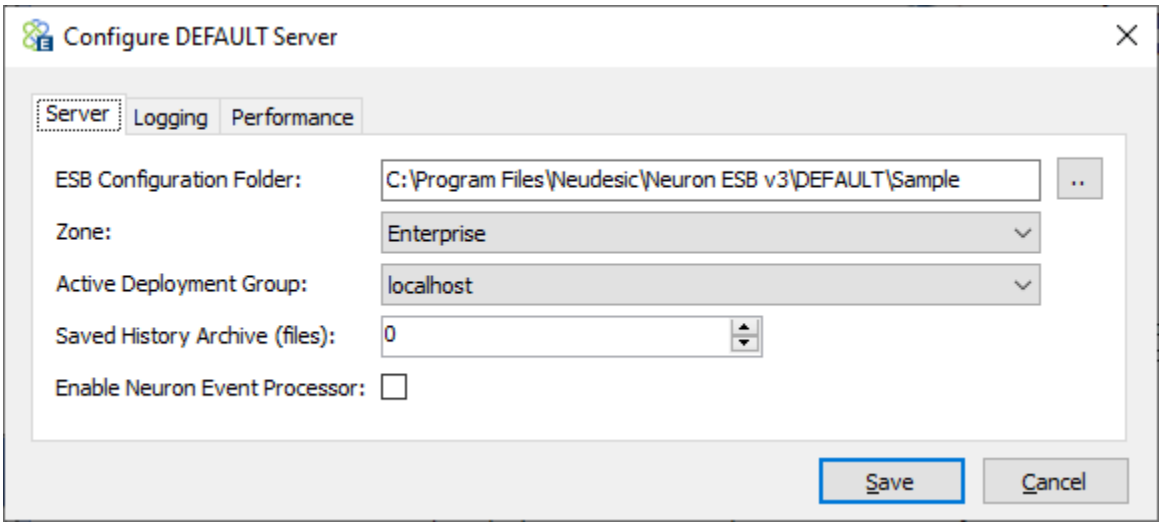
Click OK and you should see something similar to below.



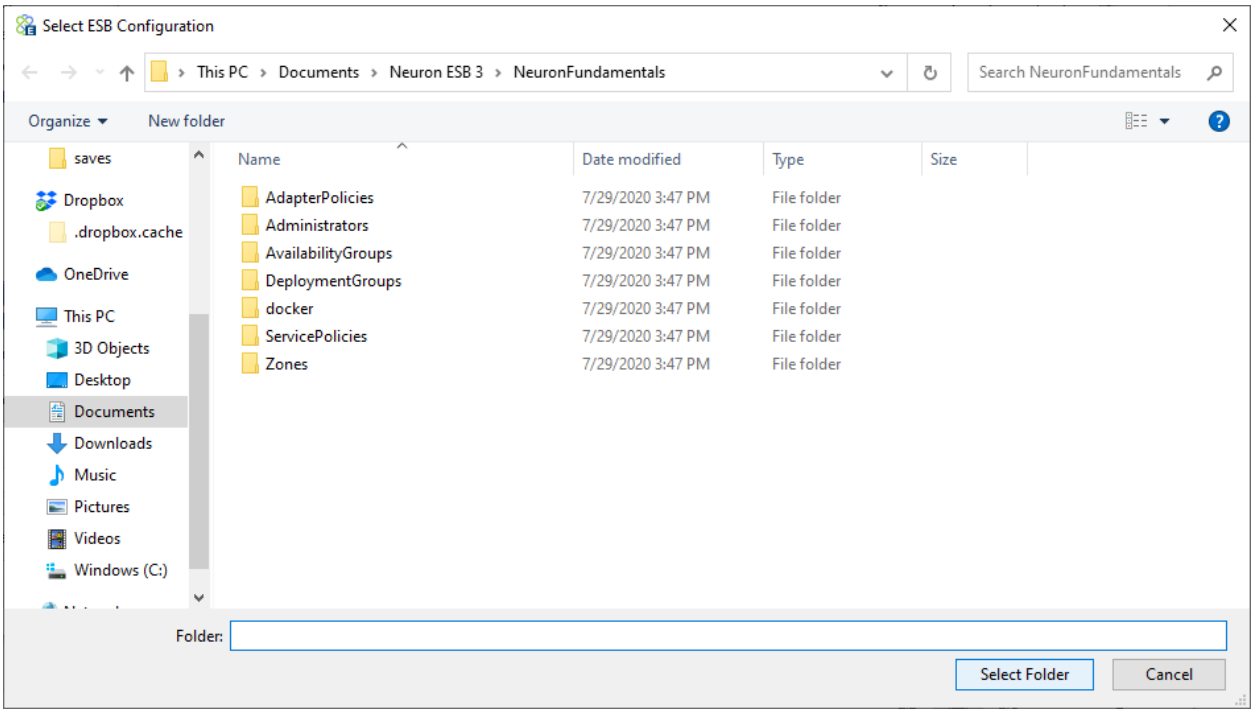
Notice that the path to the ESB configuration now shows in the Status Bar and the color of the bar is still yellow.

At this moment the new ESB configuration is not the ESB configuration that is currently running. For that we need to use Configure Server from the Menu Items.

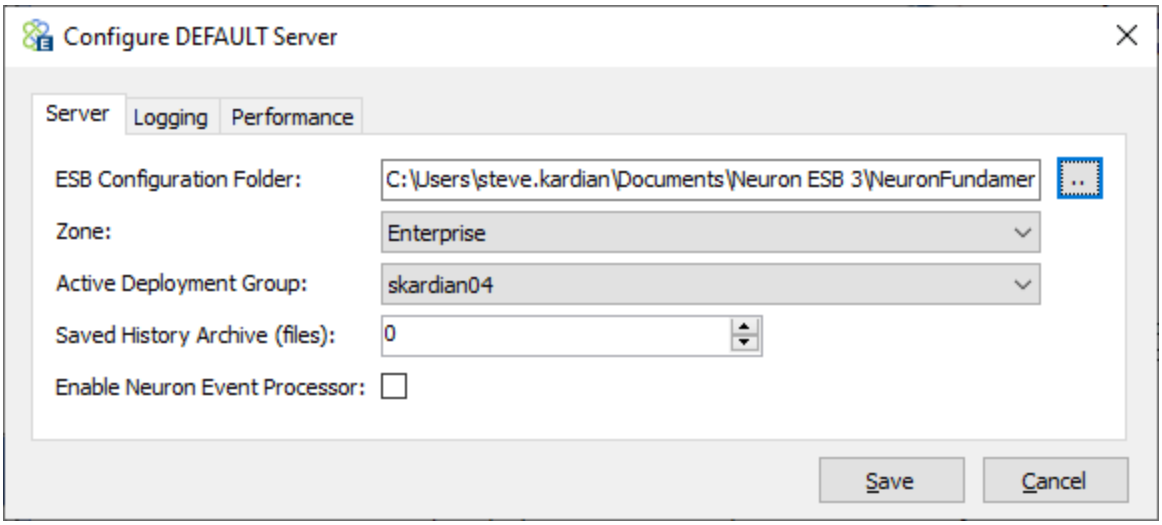
When you click Configure Server you will see the Dialog similar to the one depicted below (by default the Neuron ESB Service is not configured to run any configuration, previous in this guide you had configured it to use the Sample configuration):



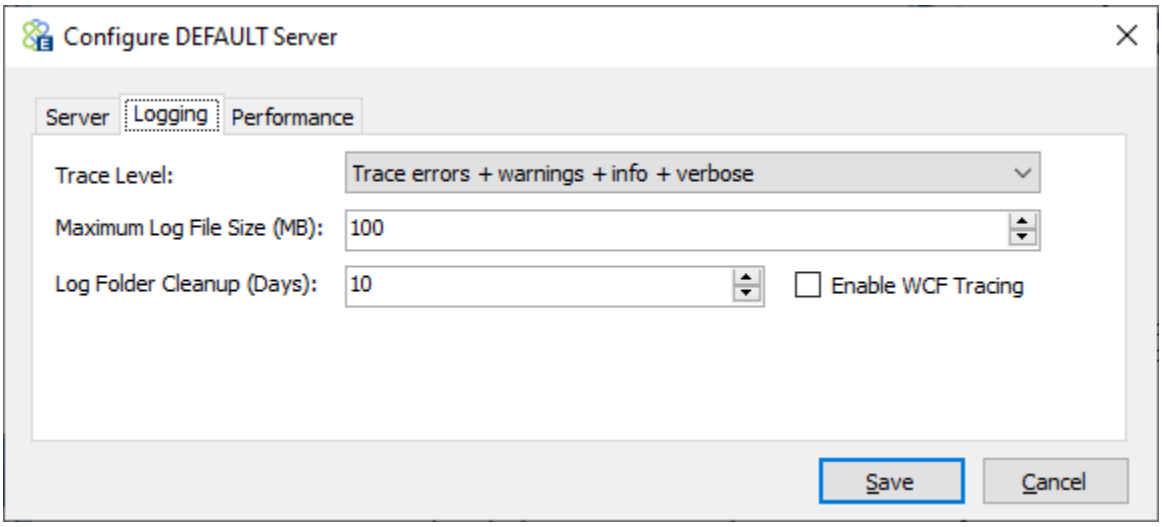
Select the ellipsis next to the ESB Configuration Folder text box to select the folder you previously created and click the Select Folder button:



Note that the Active Deployment Group changes to your machine name. When you create a new configuration, the default active deployment group is the name of your machine:



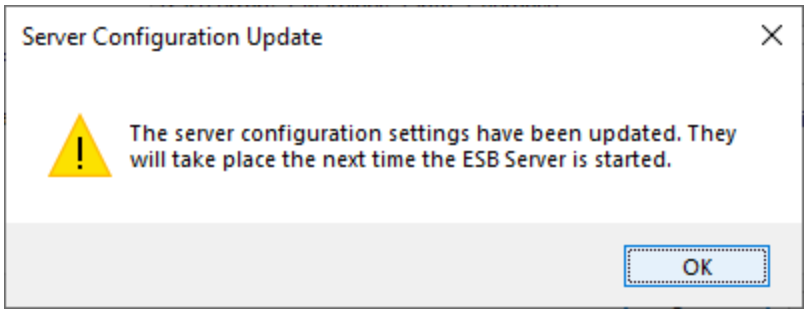
Also, adjust the Trace Level to verbose. Click on the Logging tab and verify the Trace Level dropdown list is set to Trace errors + warnings + info + verbose:



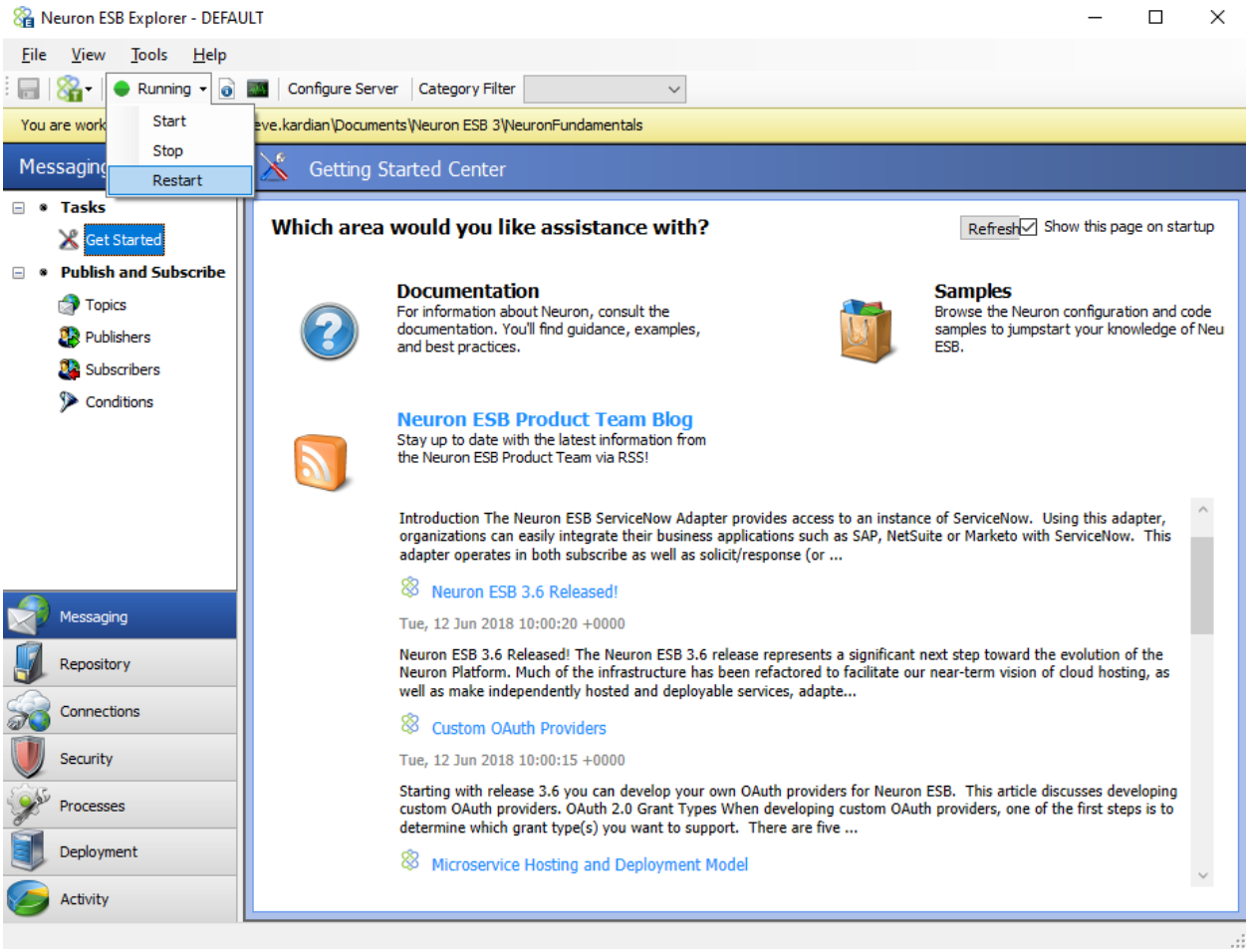
Verbose tracing can be of great benefit while developing especially in services scenarios. Each time the Neuron ESB Service is restarted a new log directory is created under the Neuron program files directory. There is a separate log created for many components in Neuron ESB including the Topic Publishing Services, Client Connectors, Service Connectors and Adapter Endpoints. This logging can reveal root causes of issues and help you develop services scenarios because the entire SOAP packet is logged when set to verbose.

NOTE: Most of this information is also written to the Neuron ESB v3 Windows Event log as well.

Click Save. When the Server Configuration Update window appears, click OK.



Use the “Running” Menu item to restart the Neuron ESB Windows Service.



The Icon will change color and the text will change showing the status. When the icon is green again the Neuron ESB runtime instance has been reconfigured to run this newly created solution.

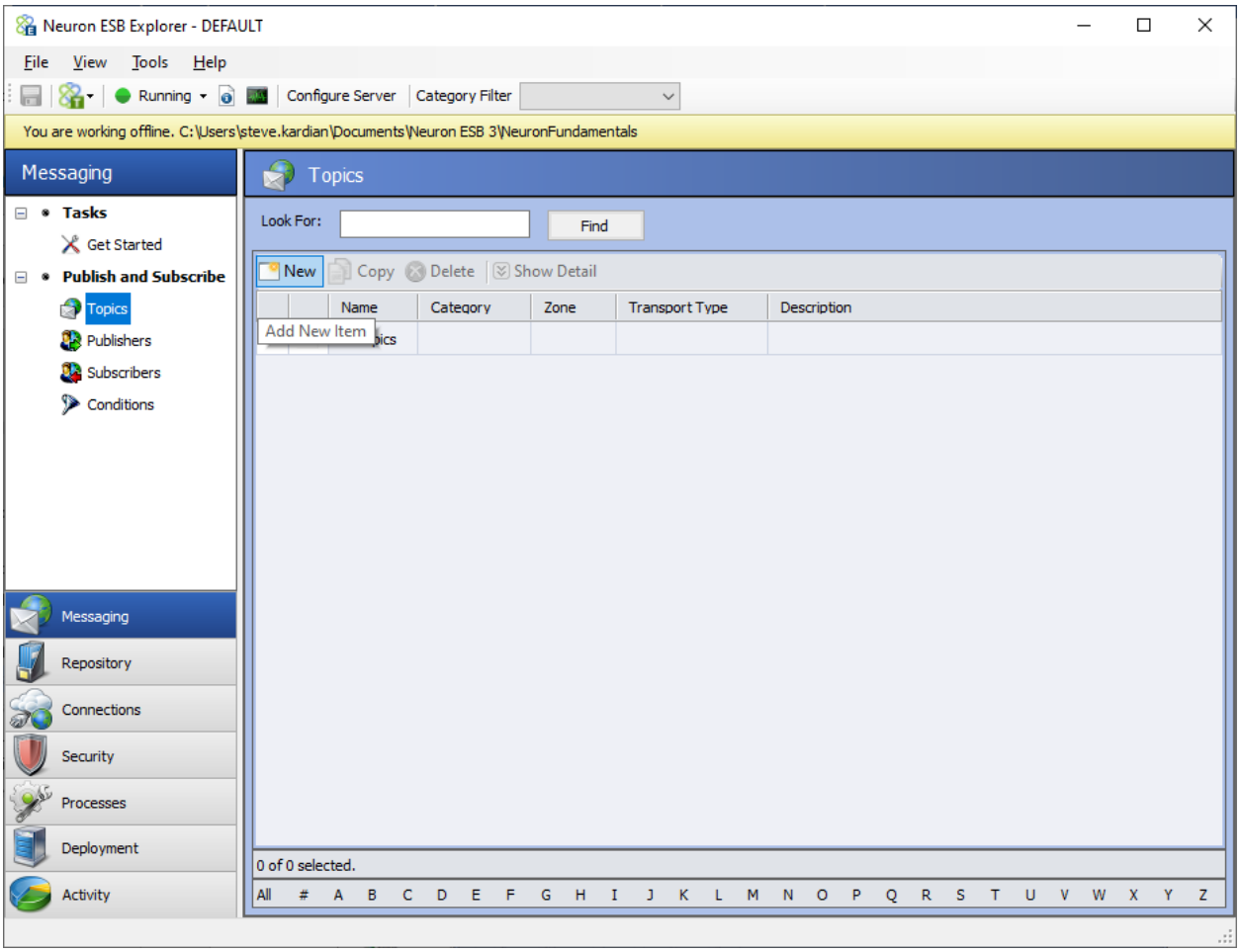
With the way our configuration is currently setup, there will be some functionality of Neuron ESB that is unavailable, specifically:

- Auditing
- Workflow

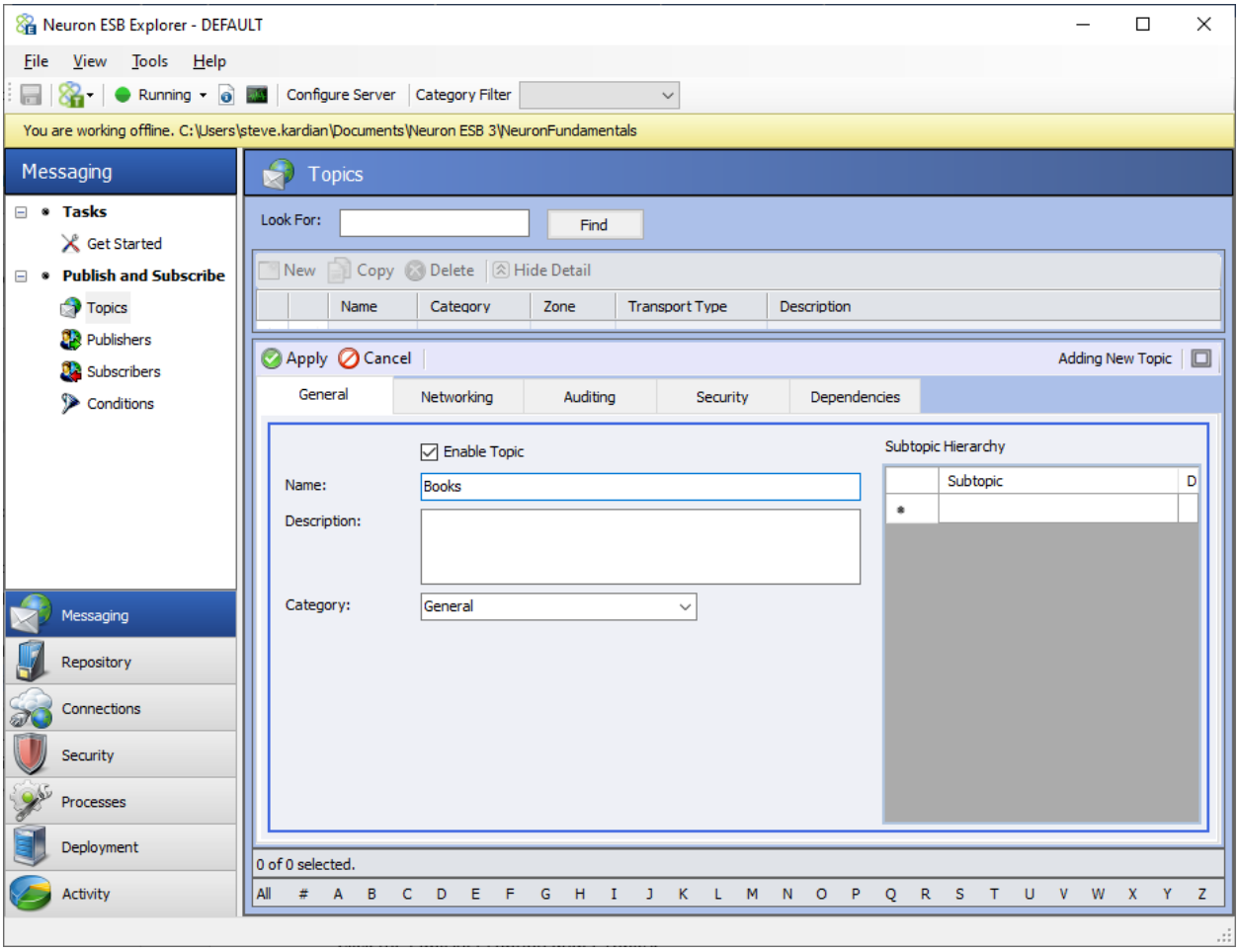
- Cluster support for endpoint hosts
- Single Instance Mode
- Activity Session Monitoring

This is because Neuron ESB requires a database for these functions to operate. You can setup a database for Neuron ESB after this tutorial: https://www.neuronesb.com/article/kb/resources/

Next, we will add a Topic and Two Parties. Navigate to the Messaging Tab and Choose Topics. Click on the New button. Note the location of the button in the image below. All Neuron ESB artifacts use a similar format for viewing, adding, editing and deleting:



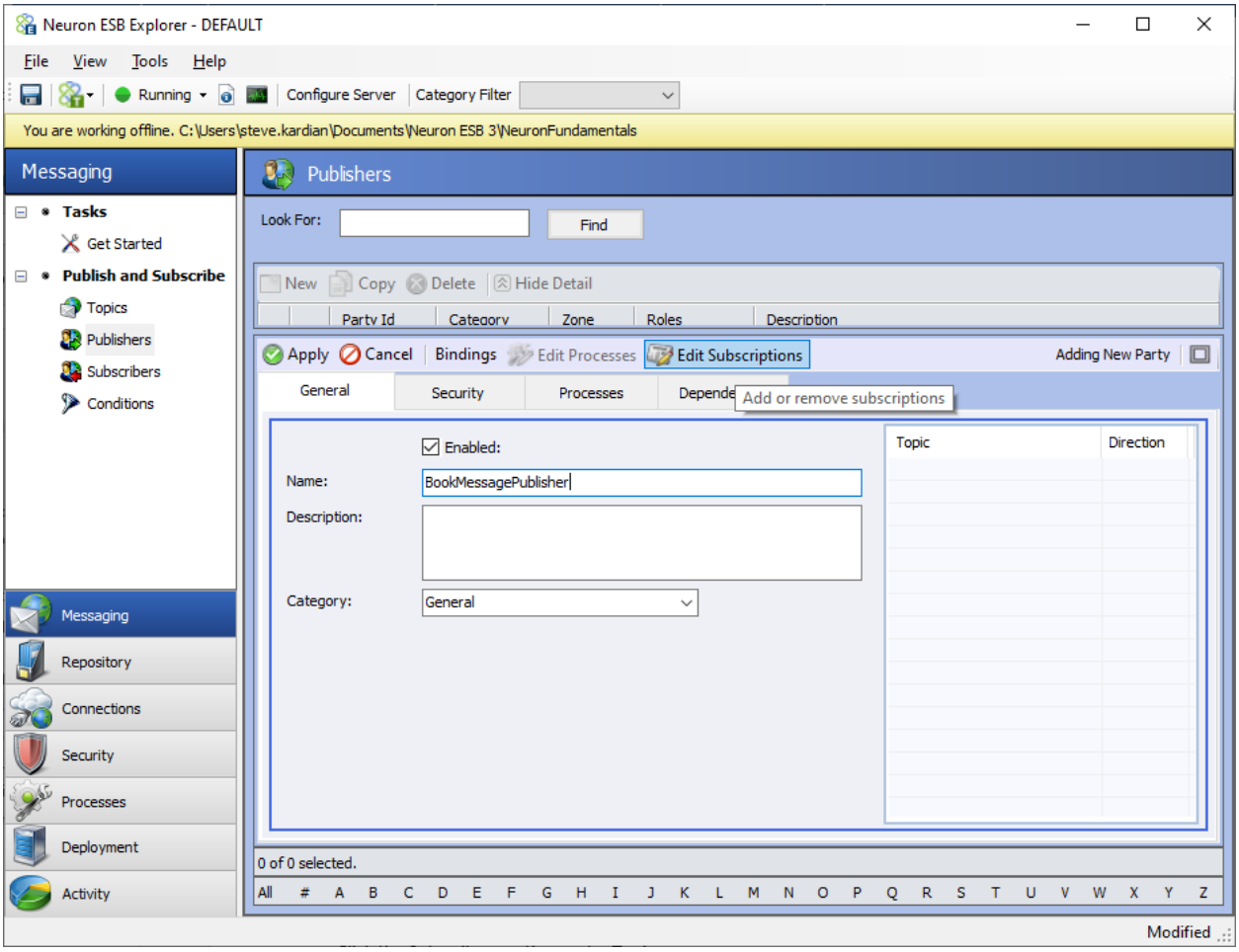
Rename the Topic Books:



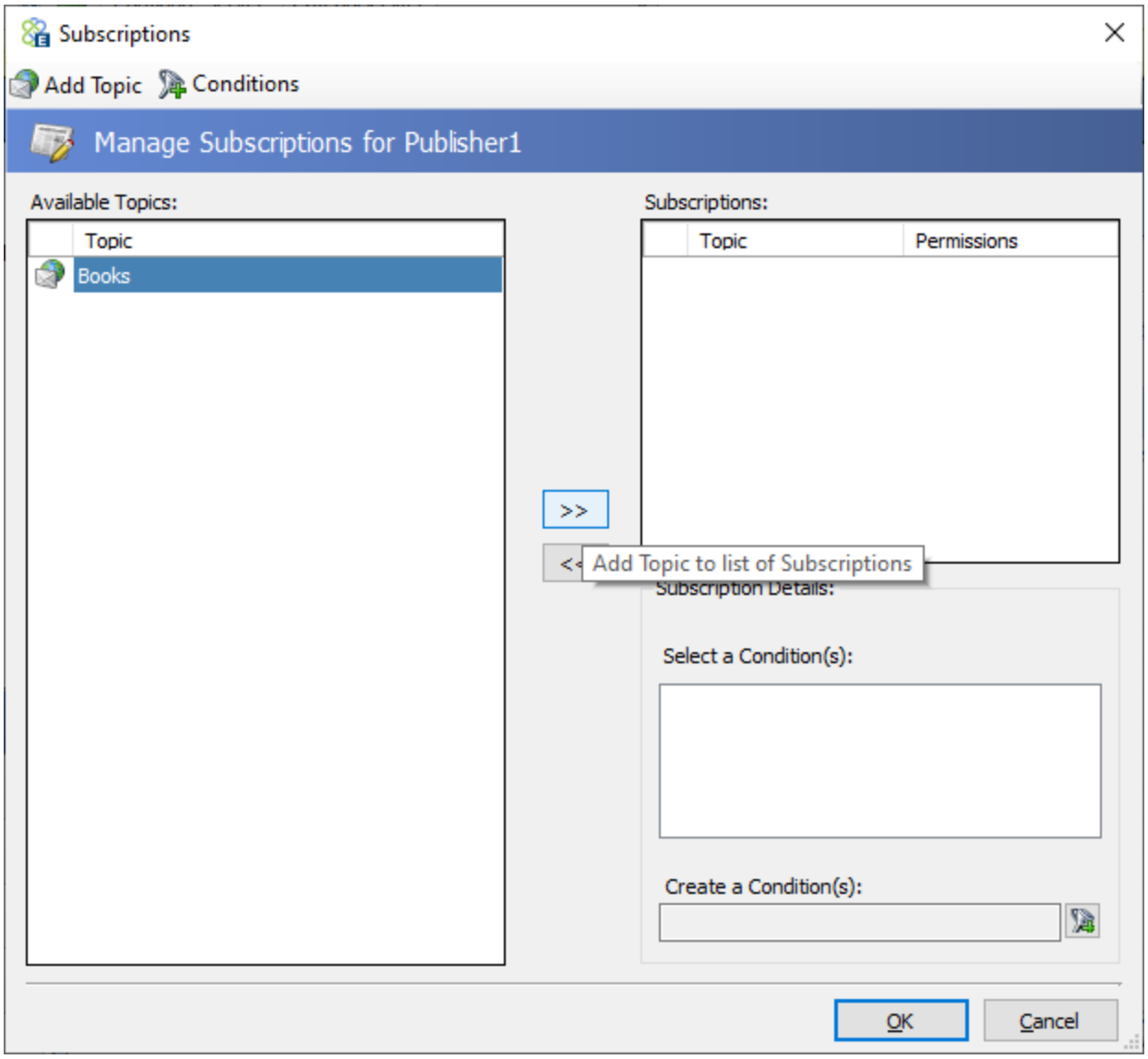
Press the Apply button.

Click the Publishers option under Topics.

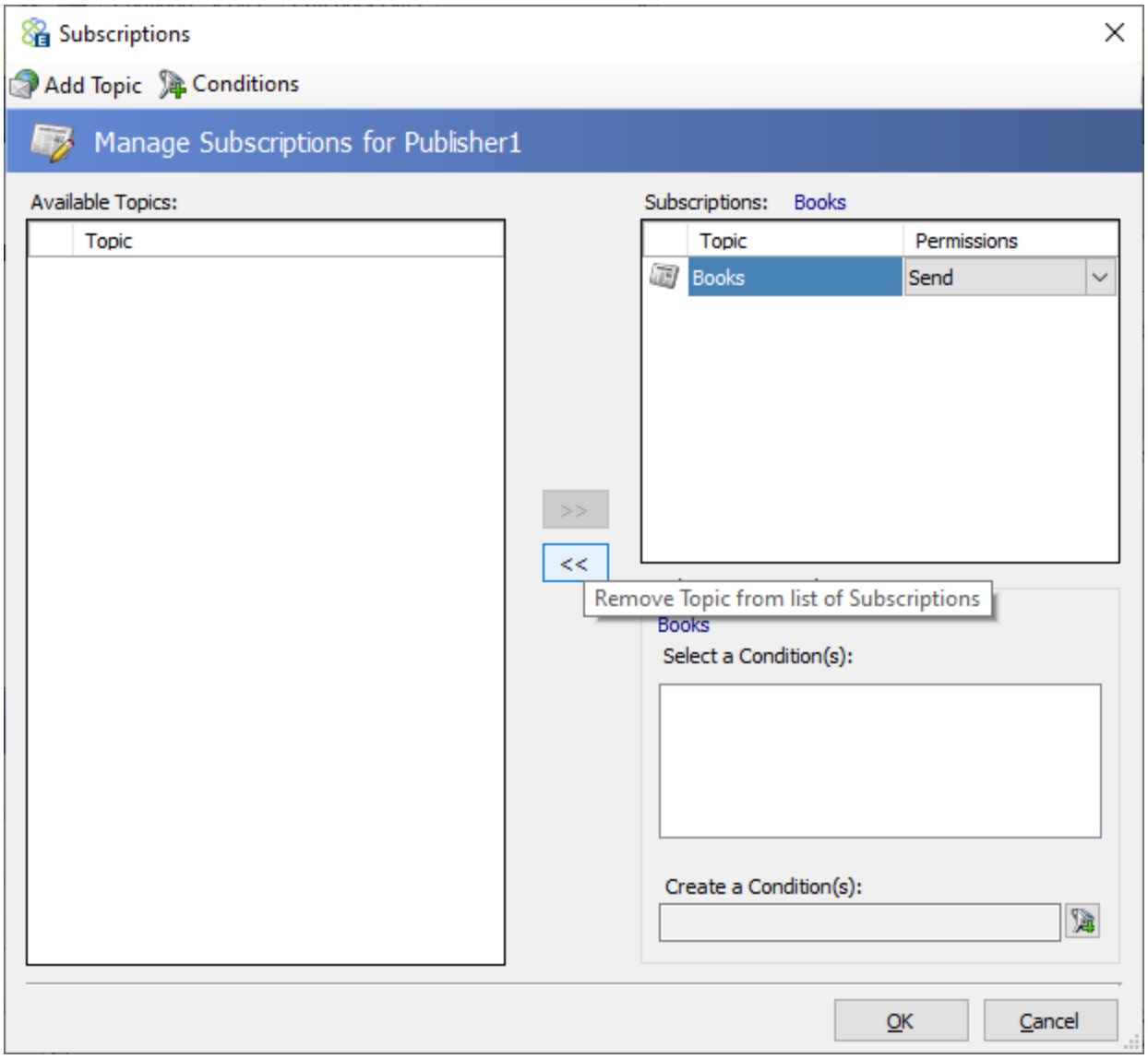
Click the New button. Change the name to BookMessagePublisher and Click the Edit Subscriptions button in the editor toolbar:



Select Books in the dialog and click the arrow button that points to the right (outlined in red below):

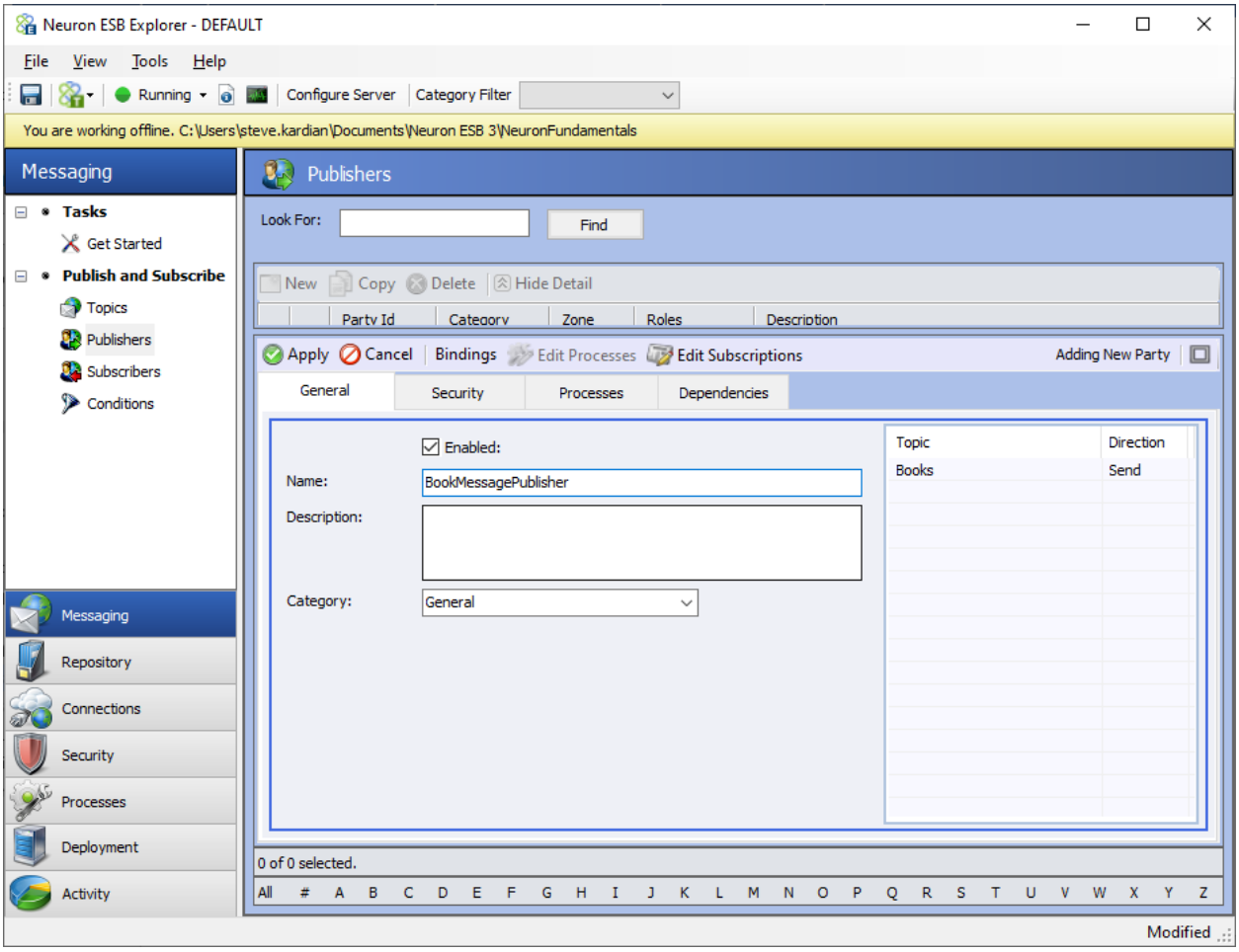


To remove a topic subscription, you would click the arrow button that points to the left.



A Party can be locked down to only Send or Receive. When you create a new Publisher the default permission is Send. For now, leave the default value and press OK and then press Apply.

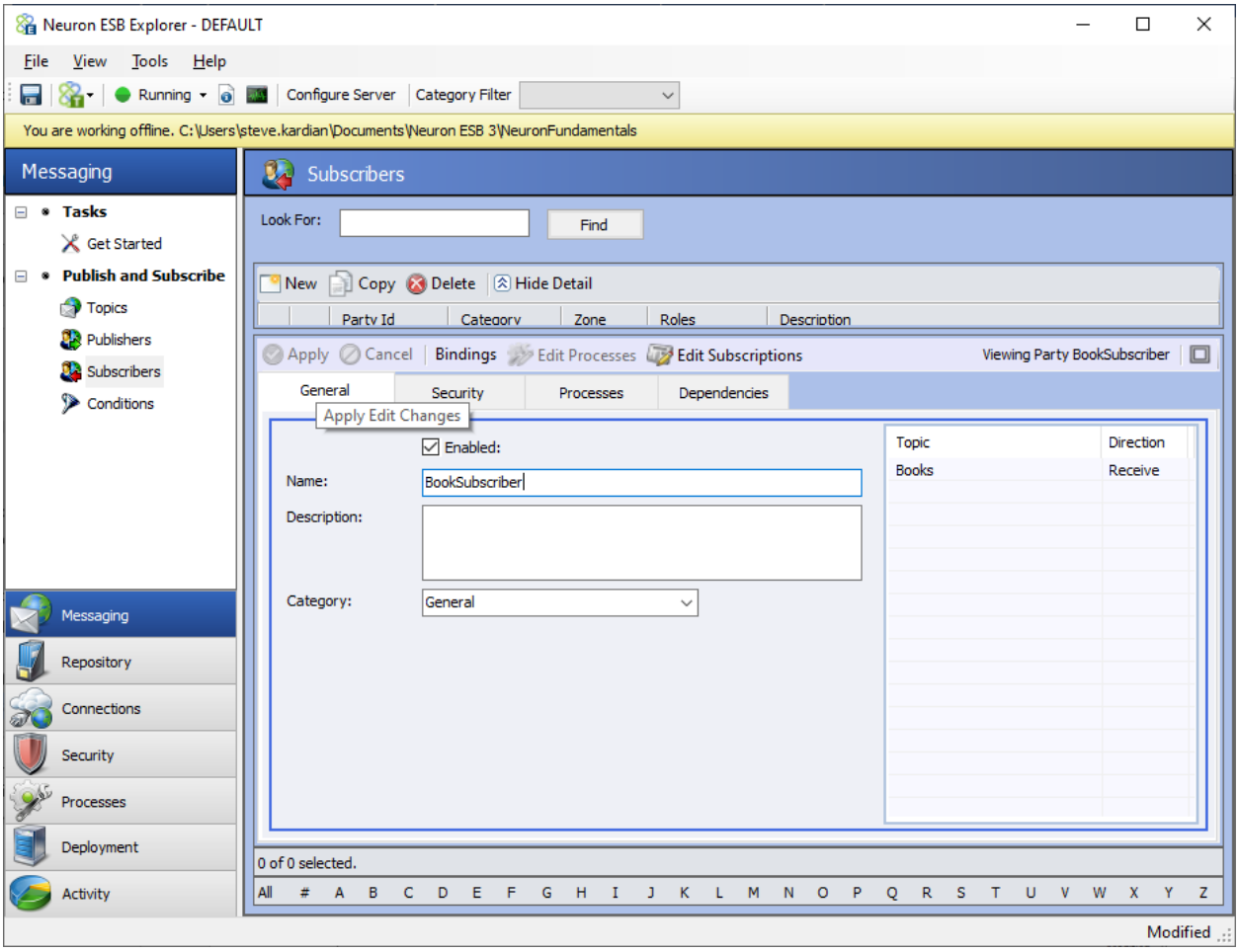
You should now see something like this:



Click the Subscribers option under Topics.

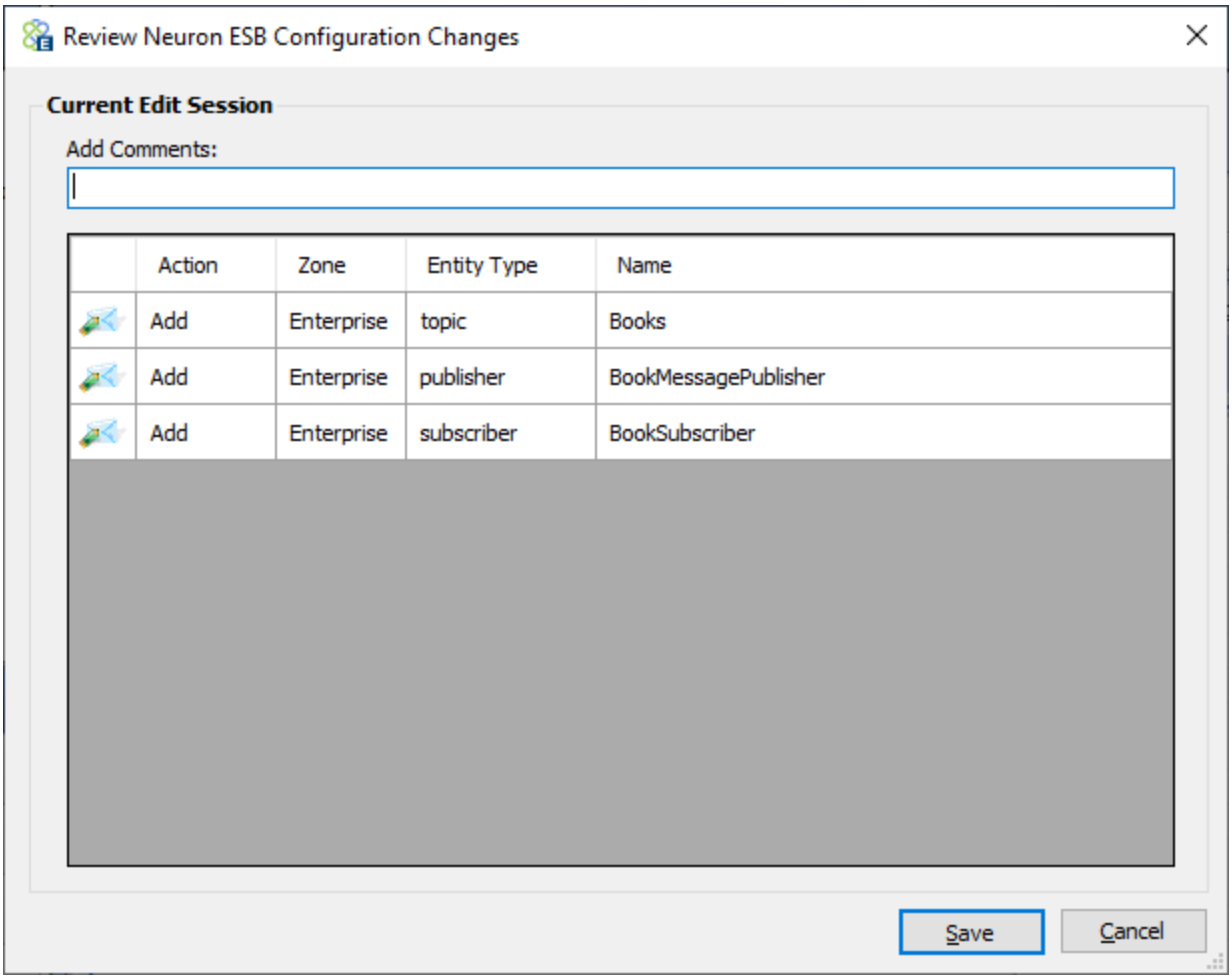
Click the New button. Change the name to BookSubscriber and Click the Edit Subscriptions button in the editor toolbar. Follow the steps above to add a subscription to the Books topic. Notice that when you add the subscription to Books, the permissions for a subscriber default to “Receive”. Leave the default value and press OK and the press Apply.

You should now see something like this:



Now hit Save.

The following dialog should appear:



The dialog shows the changes you have made to the ESB configuration since last changed.

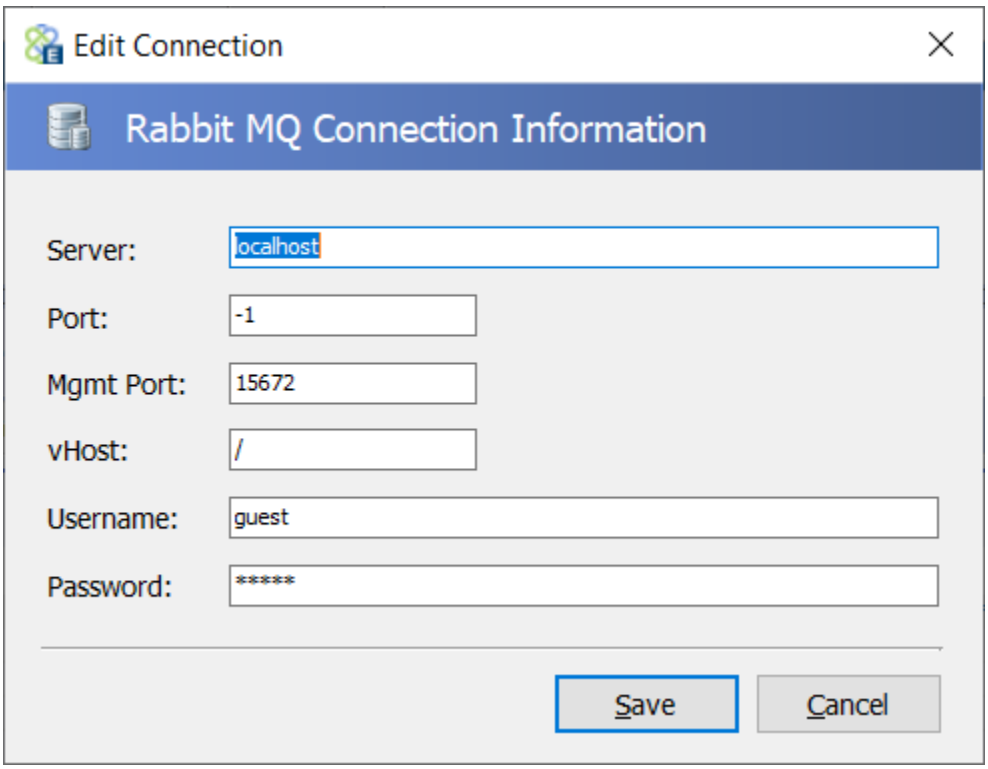
Choose “Save” and then examine the next dialog before choosing OK. You’ll see that the dialog informs you of the version of the ESB configuration that is being saved. By default, a new ESB configuration is saved each time you hit Save and the previous version is stored in the **history** sub directory within the folder you created to store the ESB configuration.

Next, we will set up a Topic using the RabbitMQ Transport and then we will subscribe to that Topic using our existing Parties. In order to be able to add a queued Topic and be able to use it without restarting you must do things in the following order.

If you installed Rabbit MQ with its default settings on the same box as Neuron ESB, the existing configuration being used for this exercise does not need to be modified. However, if the default settings for Rabbit MQ were modified or, Rabbit MQ is installed on a remote box, then the existing configuration must be modified by doing the following.

1. Edit the existing Rabbit MQ server for the Active Deployment Group. Navigate to Deployment ->Environments->Deployment Groups and select the “localhost” Deployment Group.

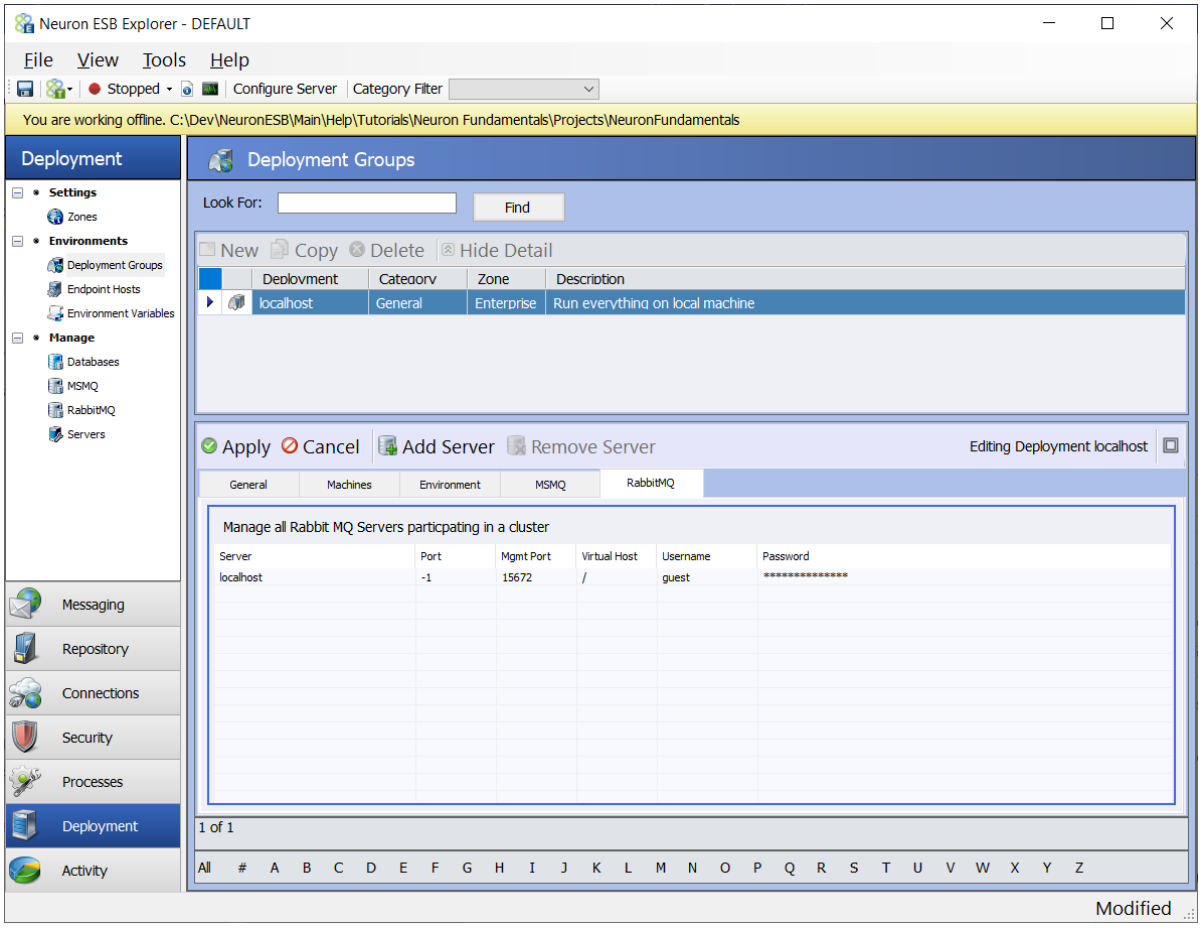
2. Click on the “RabbitMQ” tab, and double click on the existing Rabbit MQ entry to display the Edit dialog box as shown below. Make any necessary edits and click “Save”:



3. If on a remote machine, ensure that the Rabbit MQ Management Pack is enabled on the Rabbit MQ instance.

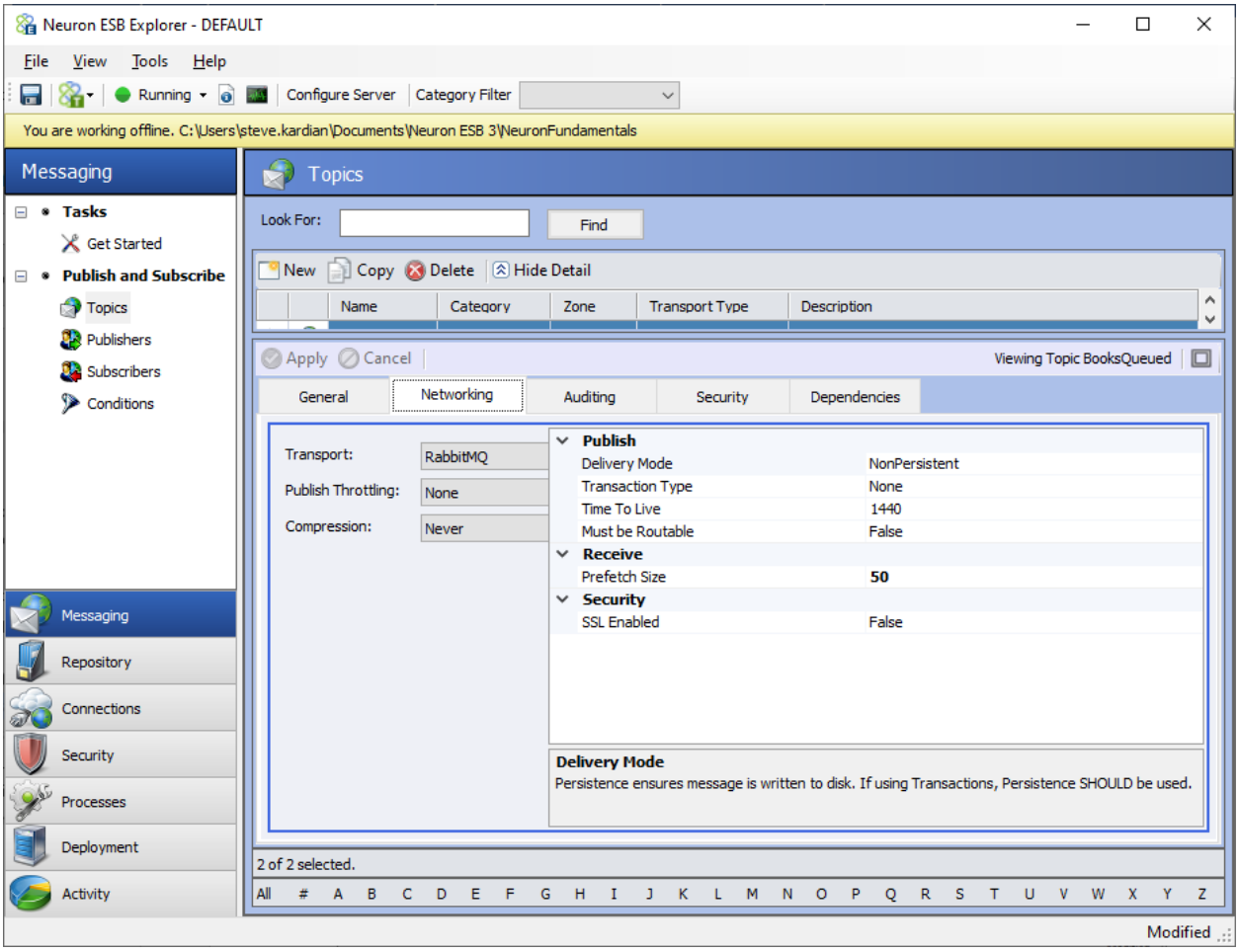
Once edited, the Deployment Group should appear as follows:

In this tutorial we are assuming that RabbitMQ is co-located with your Neuron ESB instance, and so the default settings for RabbitMQ are adequate. However, for more information on configuring RabbitMQ, to include how to connect to a remote instance of RabbitMQ, please refer to the article here:
https://www.neuronesb.com/article/rabbit-mq-topics/



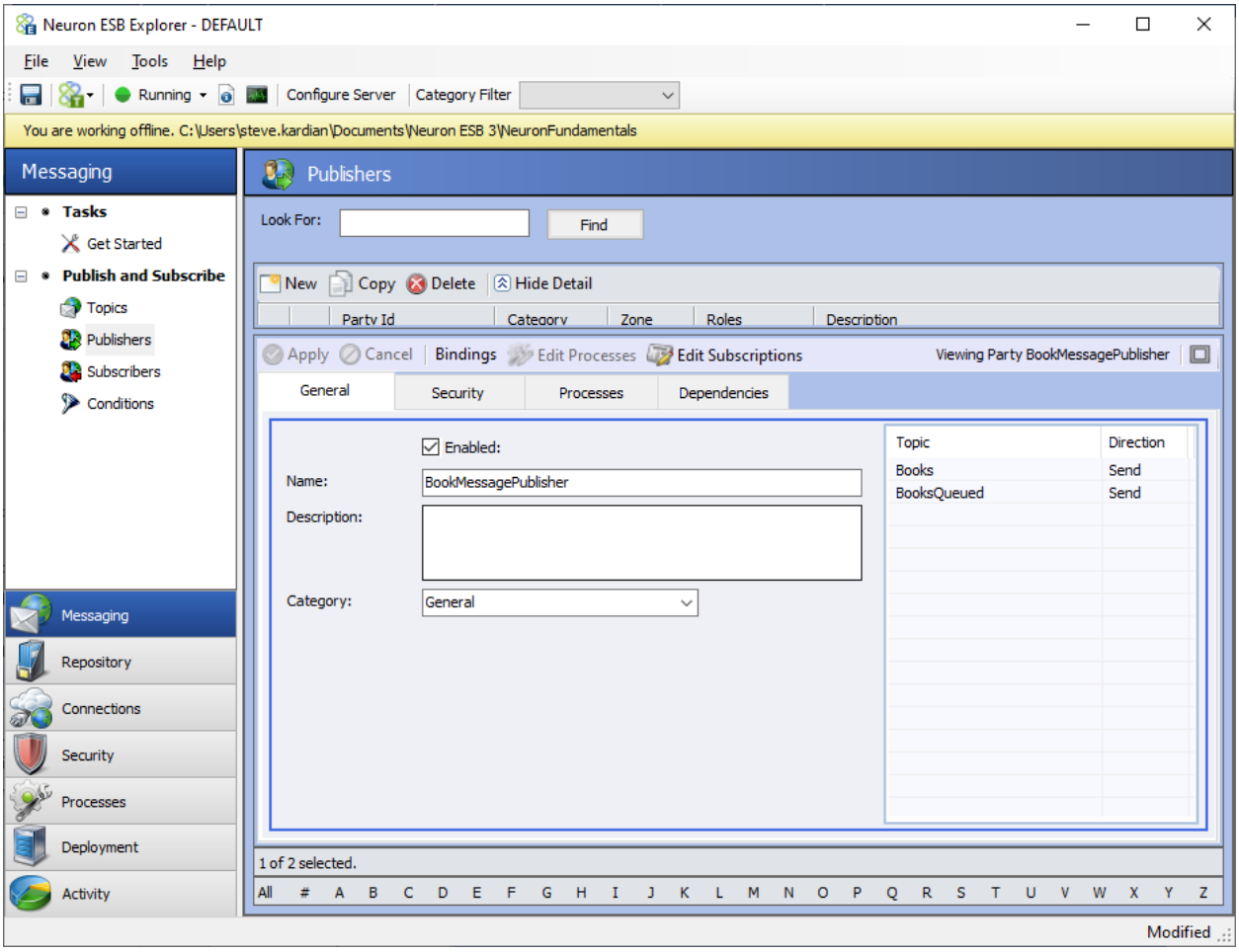
Hit the Apply button followed by the File->Save to persist the changes.

Add a new Topic (name it BooksQueued), use the Networking tab to choose RabbitMQ and then hit Apply. DO NOT PRESS SAVE YET.



Next add a subscription to the Topic for the Publisher and Subscriber previously created. Again, use the Apply button but DO NOT PRESS SAVE.

The subscription for BookMessagePublisher should look similar to below. The subscription for BookSubscriber should be similar, with the Direction equal to Receive.



Now you can click Save.

This particular order of configuration is required because of how Neuron ESB creates a publishing service for a Topic. If this process is done in the wrong order with a queued Topic then you will have to restart to see messages flow as expected.

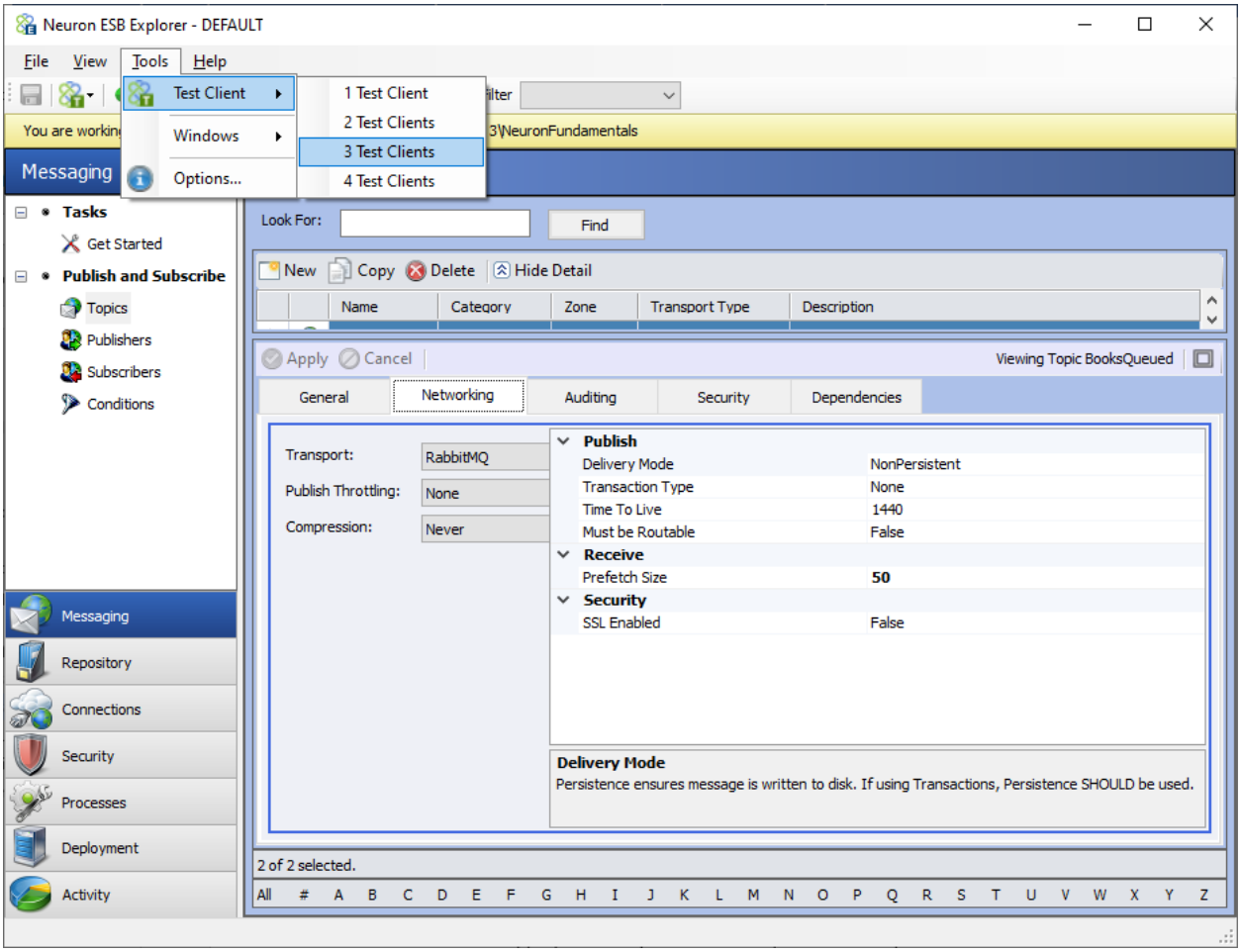
You are now ready to use the Neuron ESB Test Client to verify and test your ESB configuration.

Neuron ESB Test Client

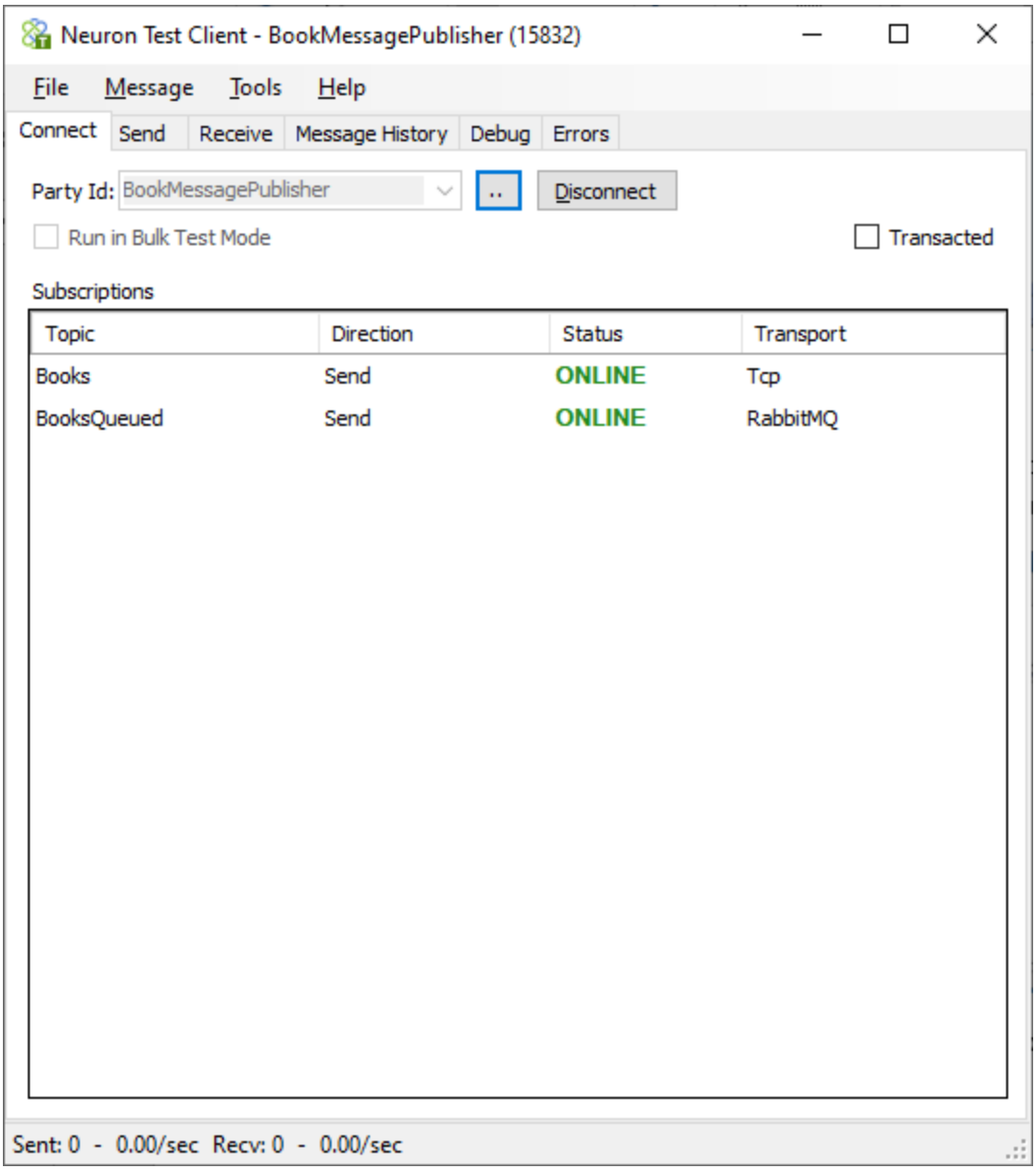
The Neuron ESB Test Client is one of the best tools you can use to verify and test your ESB configuration and to verify that your Topic Taxonomy is configured as you expect it to be.

The Neuron ESB Test Client is a separate executable from Neuron ESB Explorer that can be accessed directly via the main toolbar or through the Tools menu option in Neuron ESB Explorer. The Neuron ESB Test Client is essentially a graphical UI wrapper around the Neuron ESB Client API. This same API can be used and hosted in custom .NET applications to interact with the Neuron ESB bus in an event driven manner.

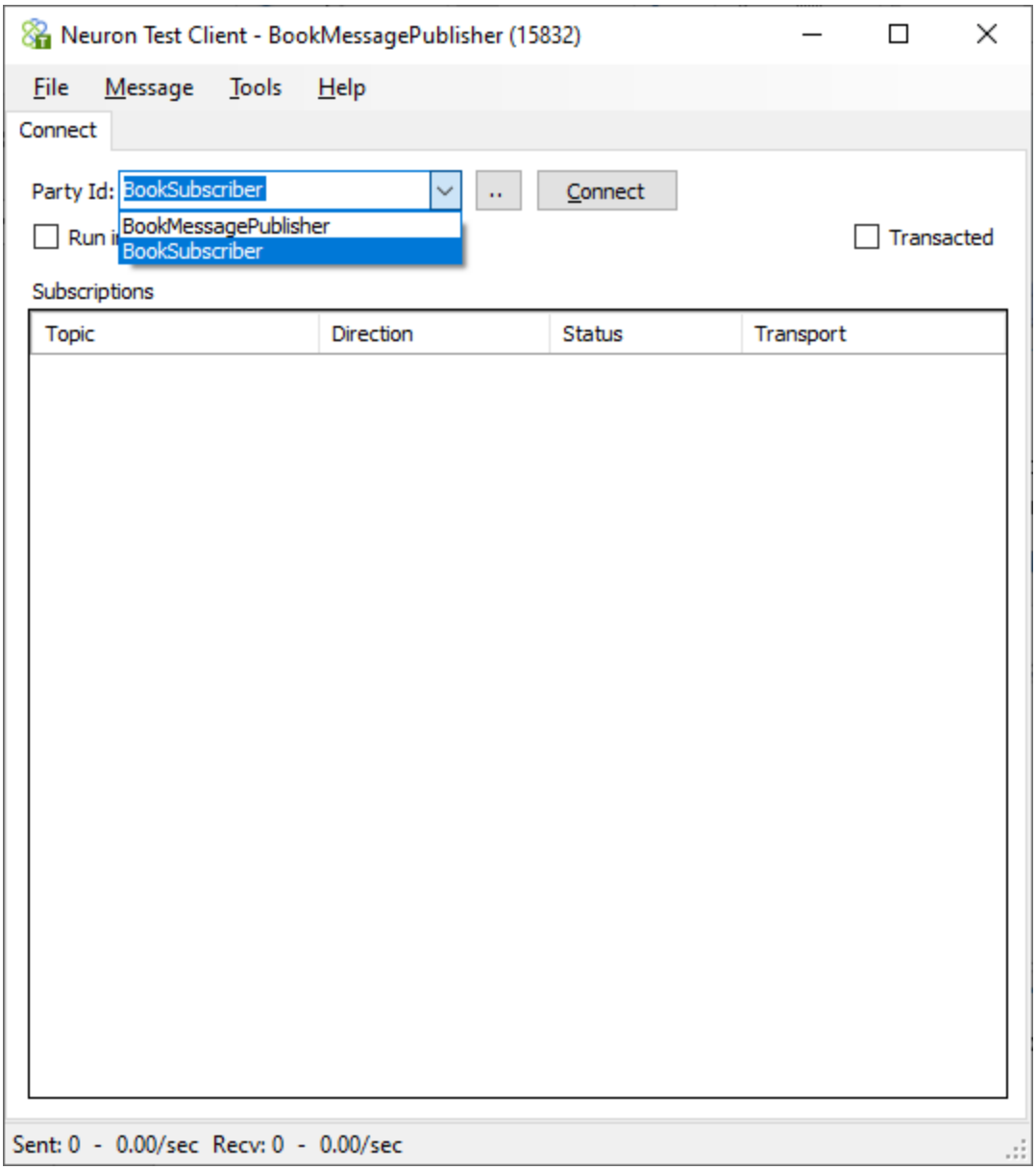
For this training we'll use the Tools menu in Neuron ESB Explorer and we will choose to launch 3 instances:

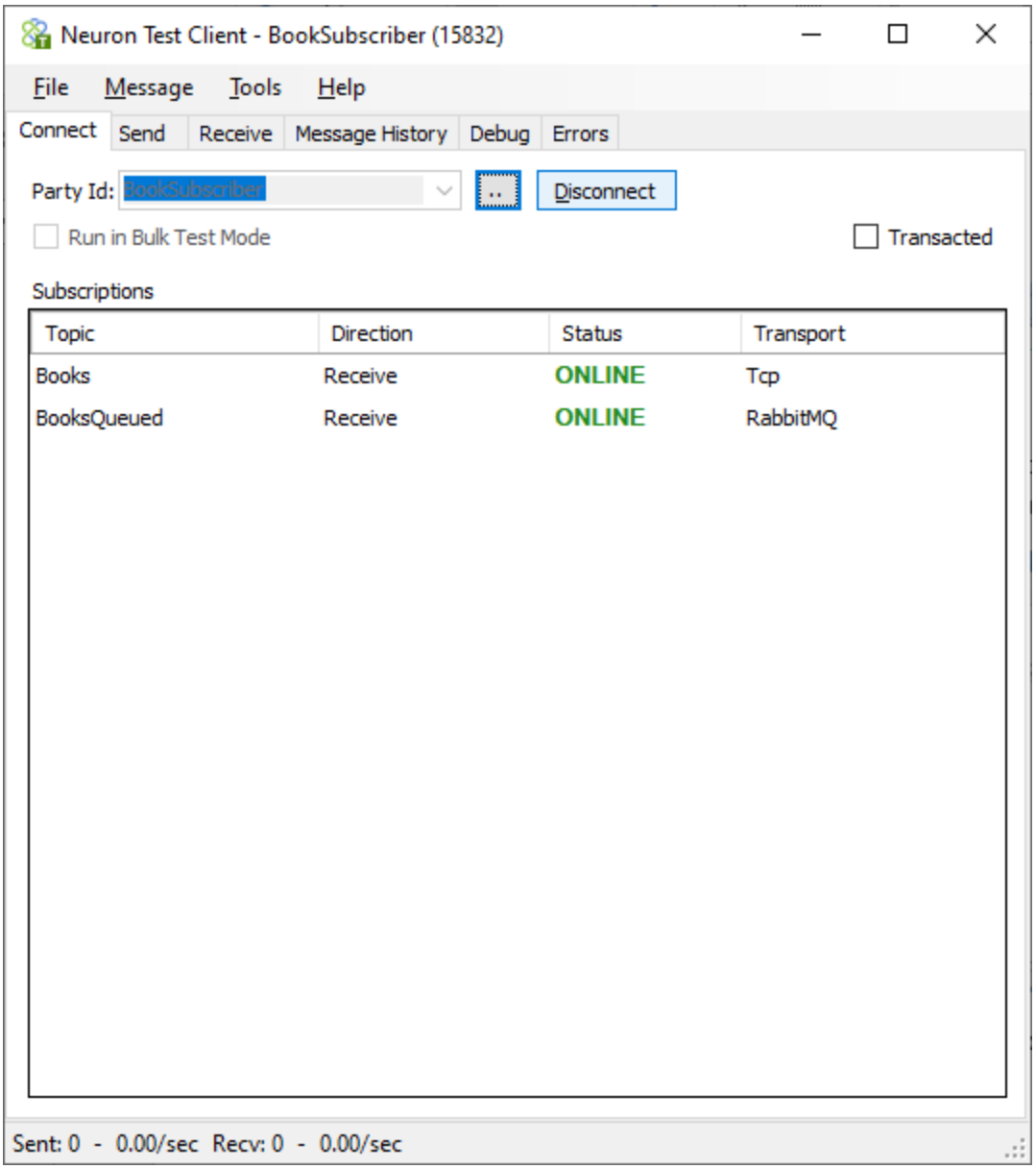


Connect one of the test clients as BookMessagePublisher.



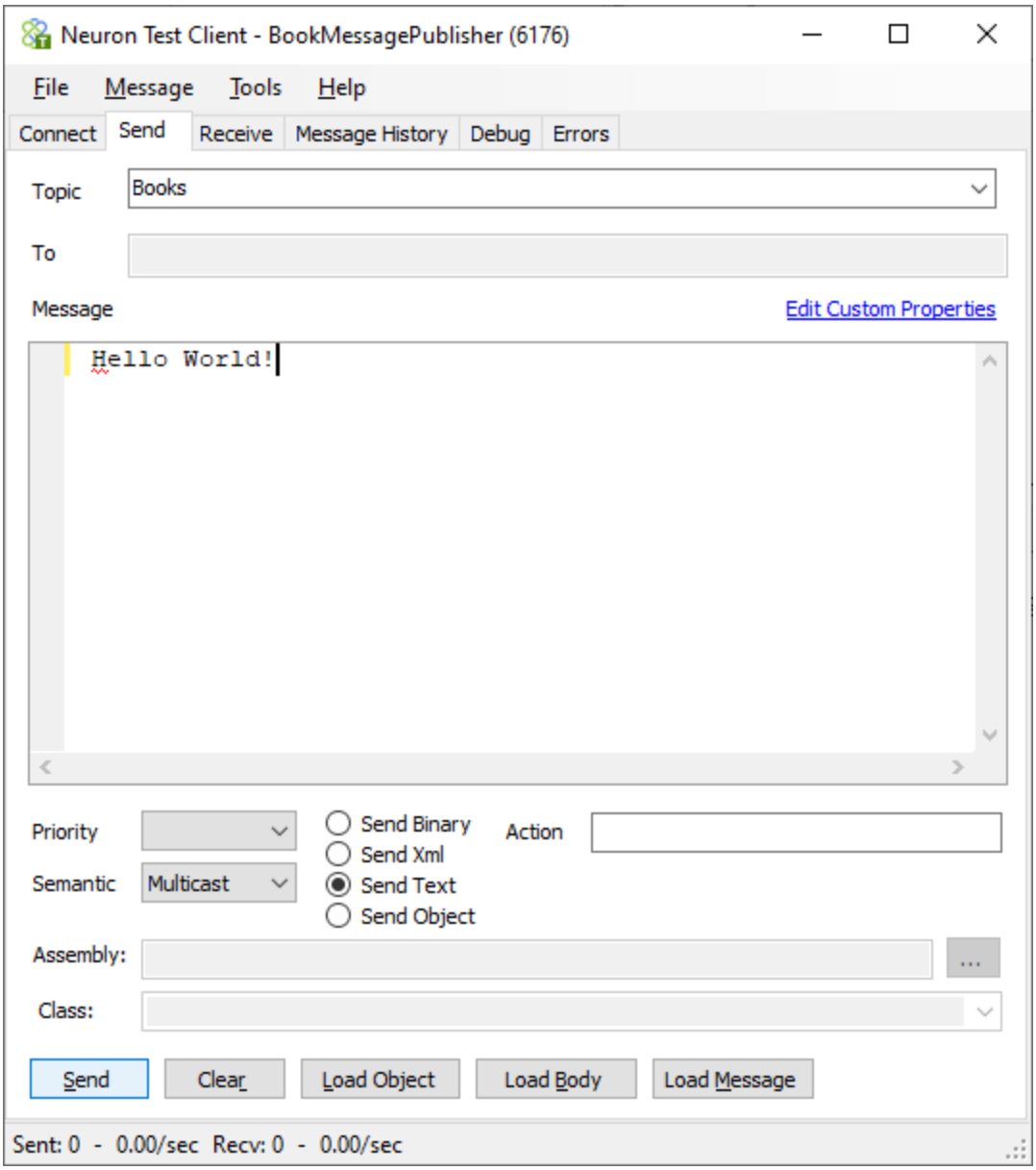
Connect two of the test clients as BookSubscriber. To do this, select the appropriate Party Id from the dropdown list and press the Connect button:


Neuron Test Client - BookMessagePublisher (15832)
File
Message
Tools
Help
Connect
Party Id:
BookSubscriber
..
Connect
BookMessagePublisher
BookSubscriber
Transacted
Subscriptions
Topic
Direction
Status
Transport
Sent: 0 - 0.00/sec Recv: 0 - 0.00/sec

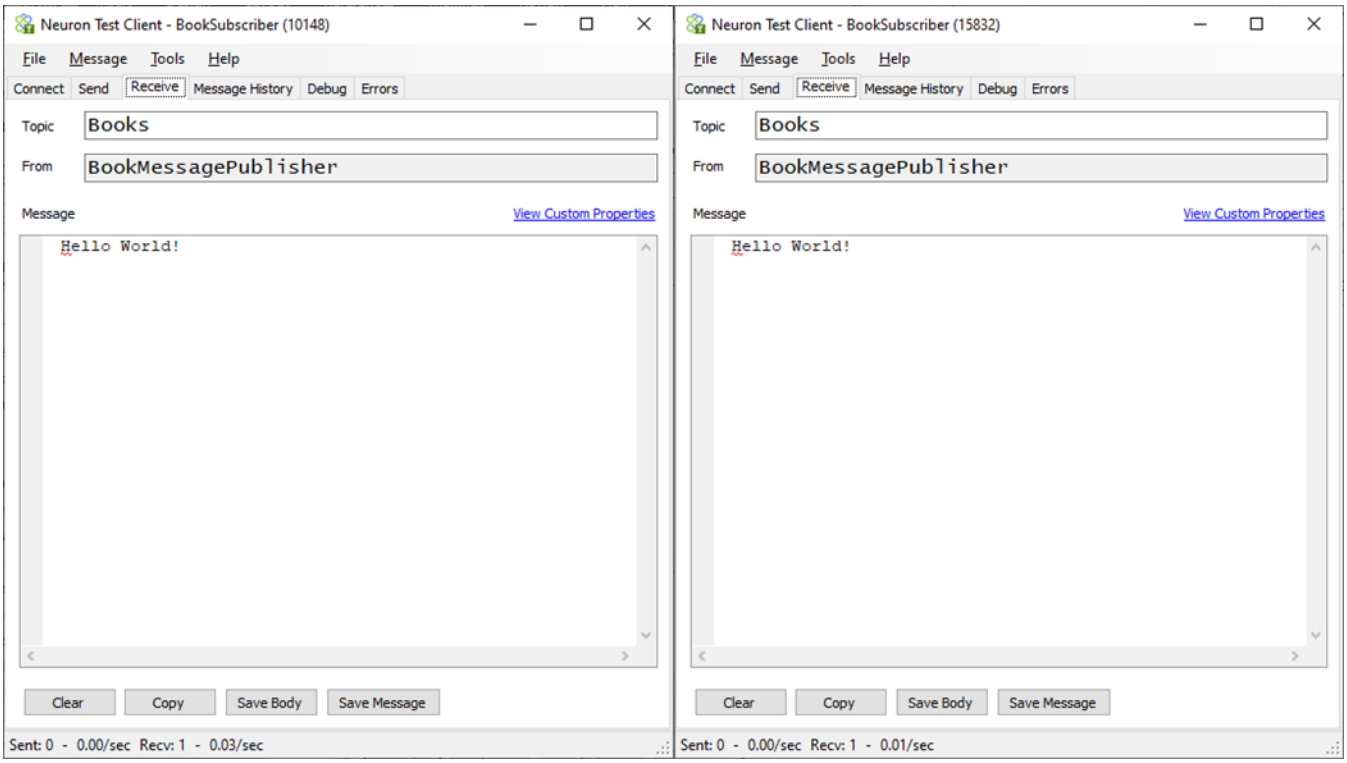


Using the BookMessagePublisher client click the send tab and choose the Books using the Topic drop down.

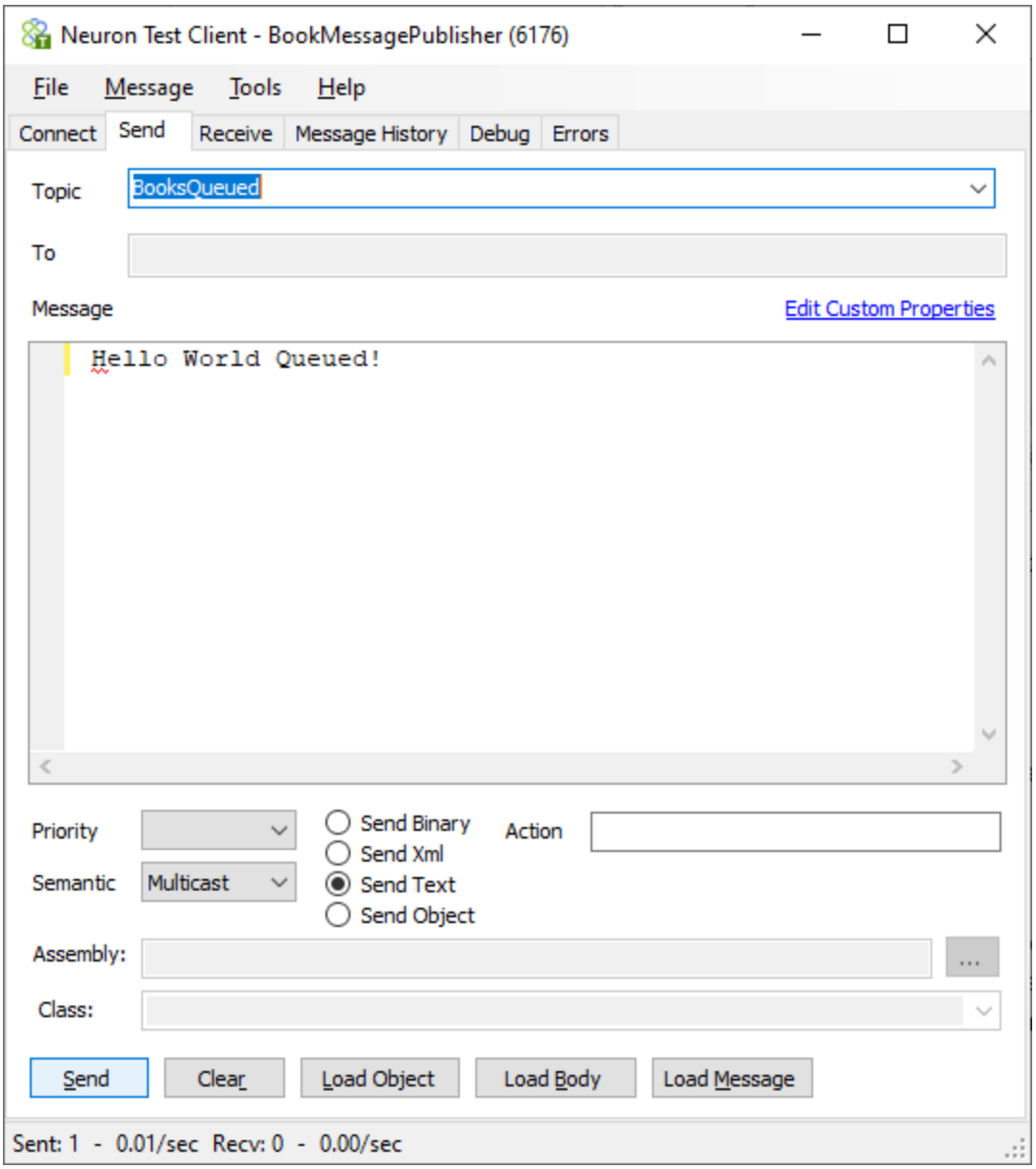
Now type “Hello world” into the large text box and press the Send button:



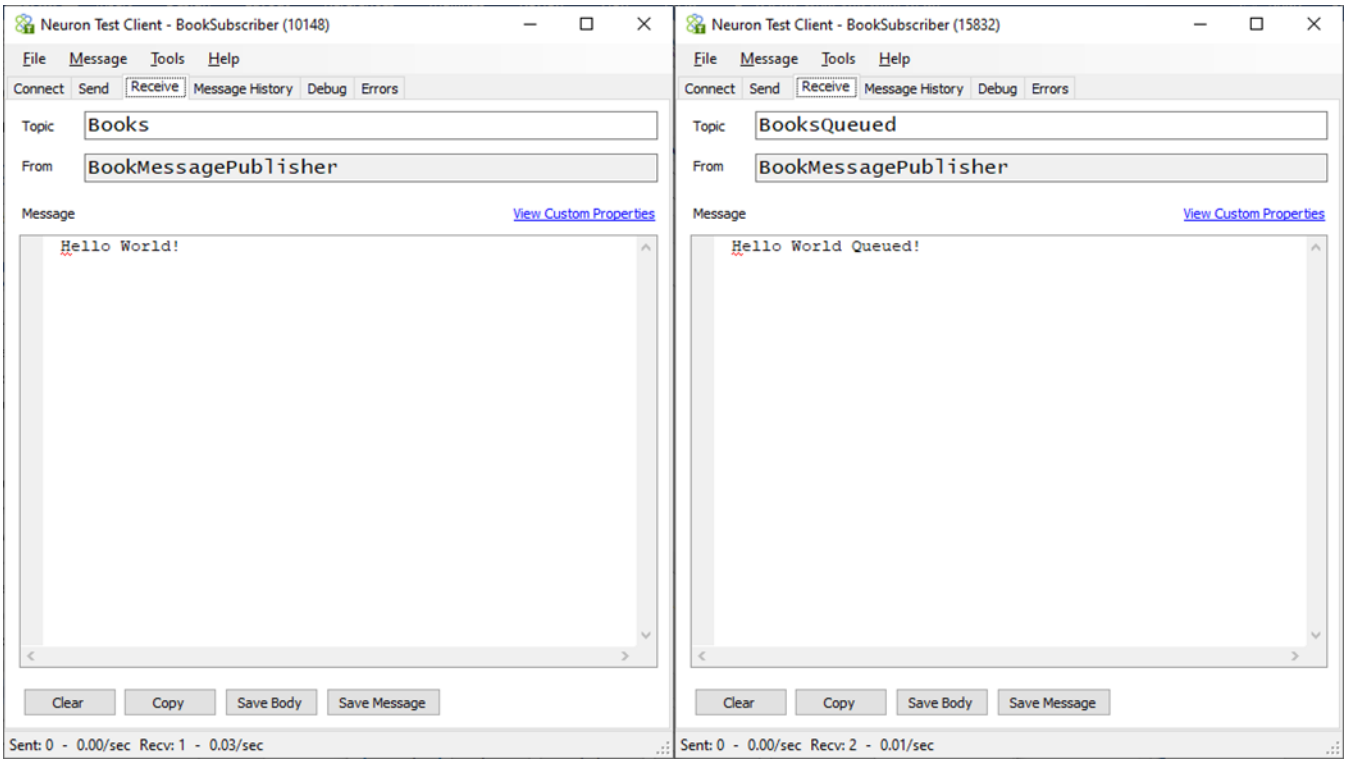
“hello world” will appear in **both** BookSubscriber Receive tabs and the message count will be 1 in each



Now change the text in BookMessagePublisher to “hello world queued” and change the Topic to BooksQueued.


Neuron Test Client - BookMessagePublisher (6176)
File
Message
Tools
Help
Connect
Send
Receive
Message History
Debug
Errors
Topic
BooksQueued
To
Message
Edit Custom Properties
Hello World Queued!
Priority
Semantic
Multicast
Send Binary
Send Xml
Send Text
Send Object
Action
Assembly:
...
Class:
Send
Clear
Load Object
Load Body
Load Message
Sent: 1 - 0.01/sec Recv: 0 - 0.00/sec

Press Send

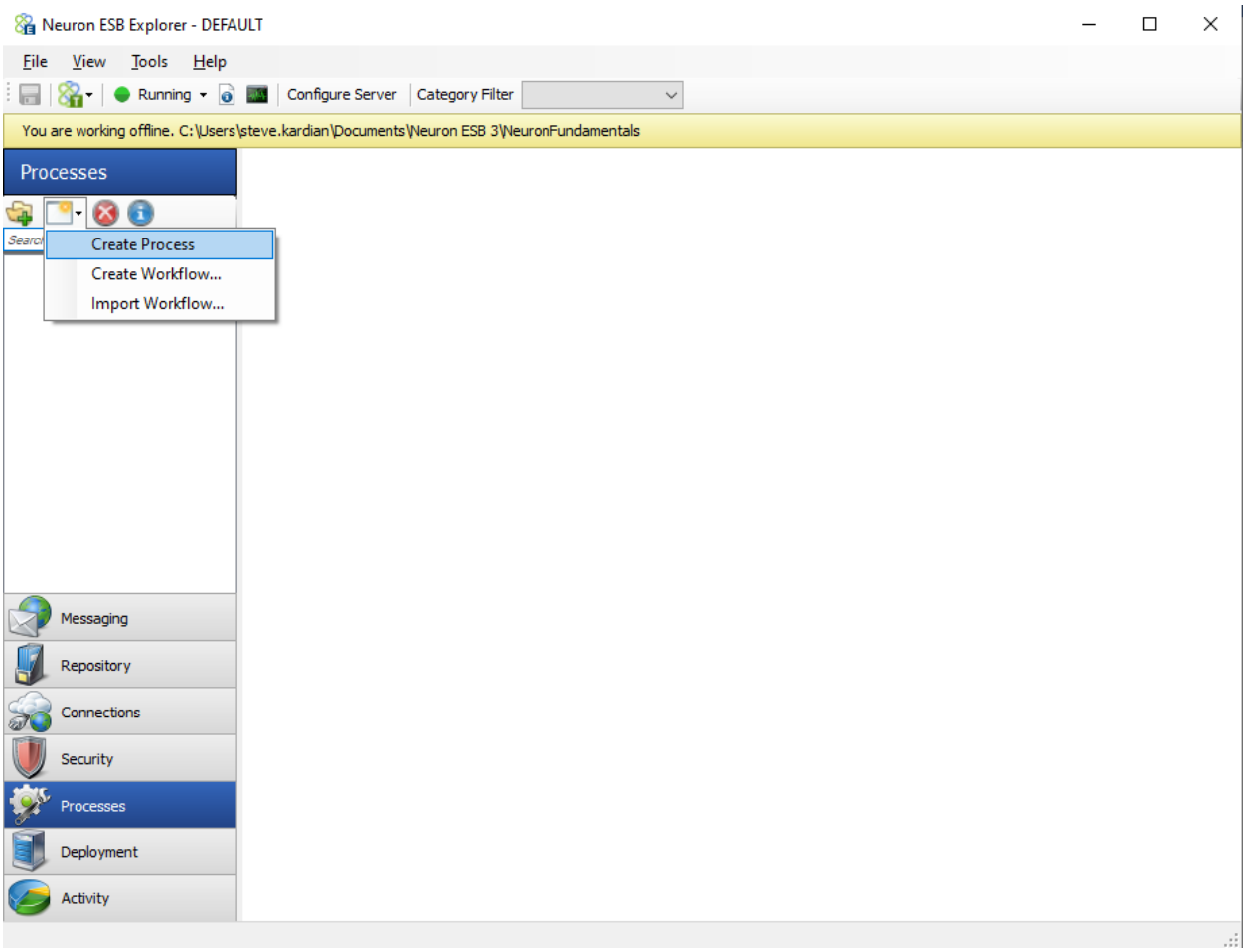


Notice only **one** of the test clients receives the message and increments the counter. This is the expected behavior. When using Queued Topics only one instance of a given Subscriber Id will receive the message. This makes sense when you realize that under the covers individual queues are created for each subscriber id so the effect of two instances connecting with the same subscriber id will essentially mean two clients using the same queue name thus accessing the same queue.

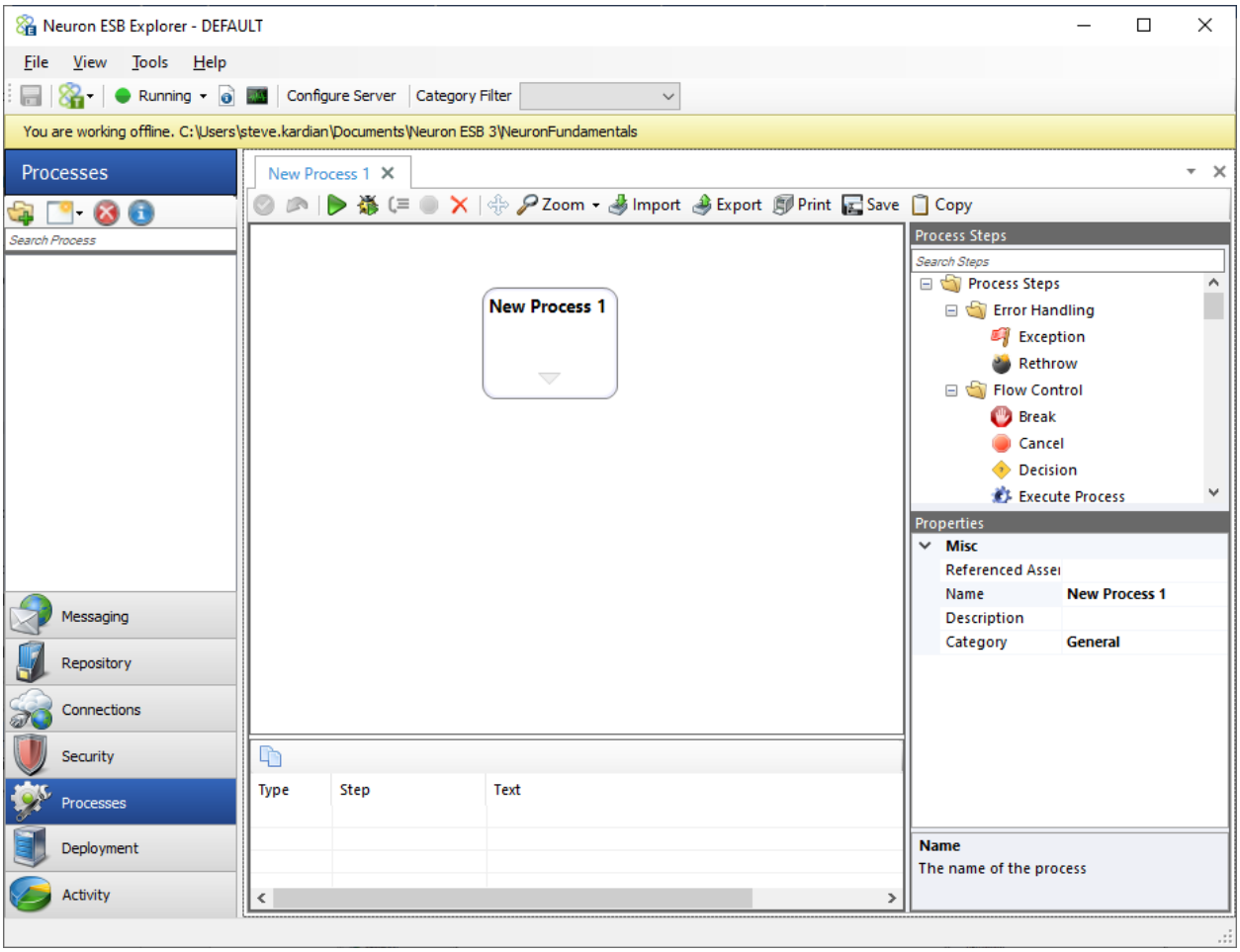
Close all three test clients.

Adding a Business Processes

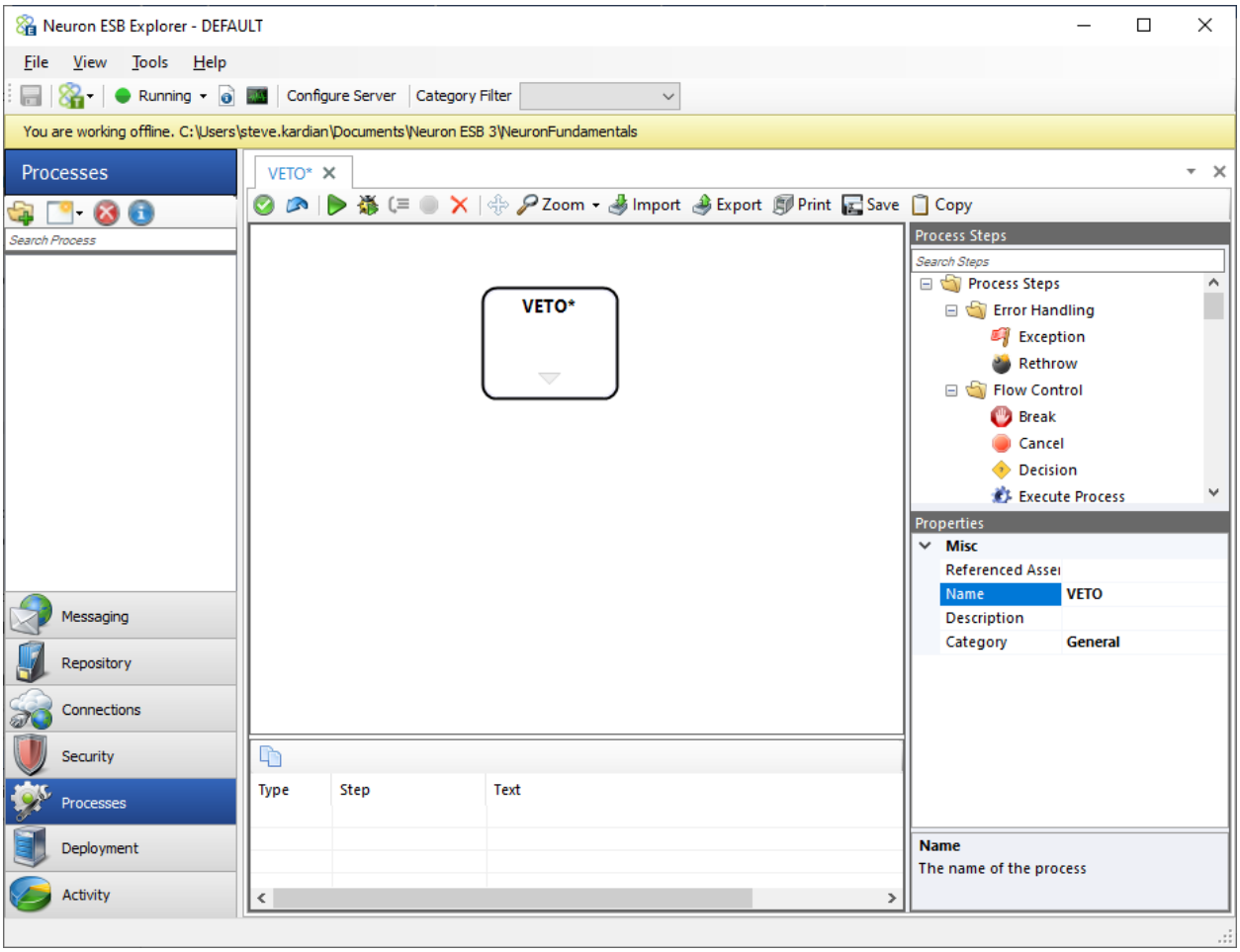
Now that you have successfully tested this configuration, you are ready to add a Business Process utilizing the VETO pattern discussed earlier in this guide. In the Neuron ESB Explorer, navigate to Processes. Click on the New dropdown outlined in red below and select Create Process:



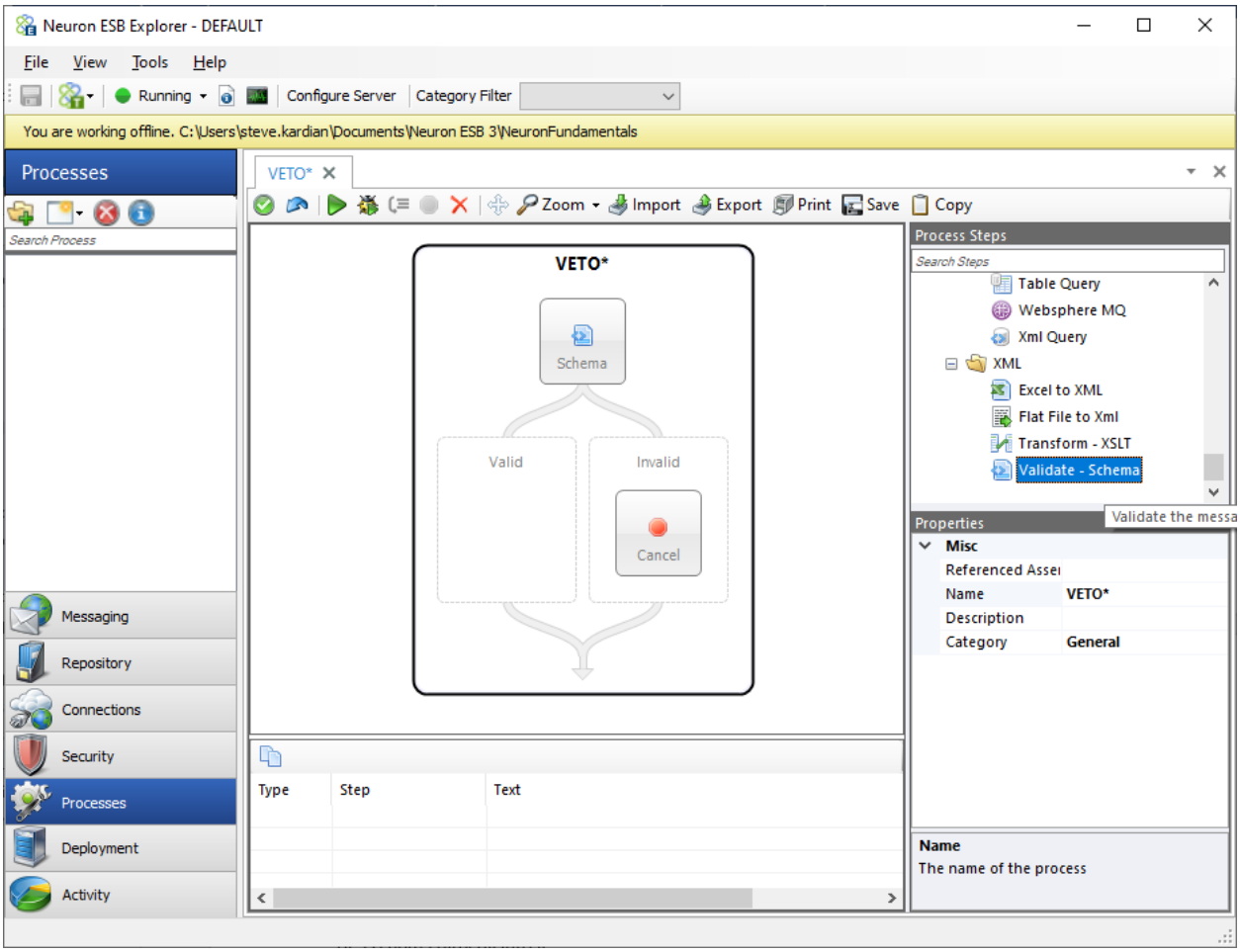
You will see a blank Business Process as shown below:



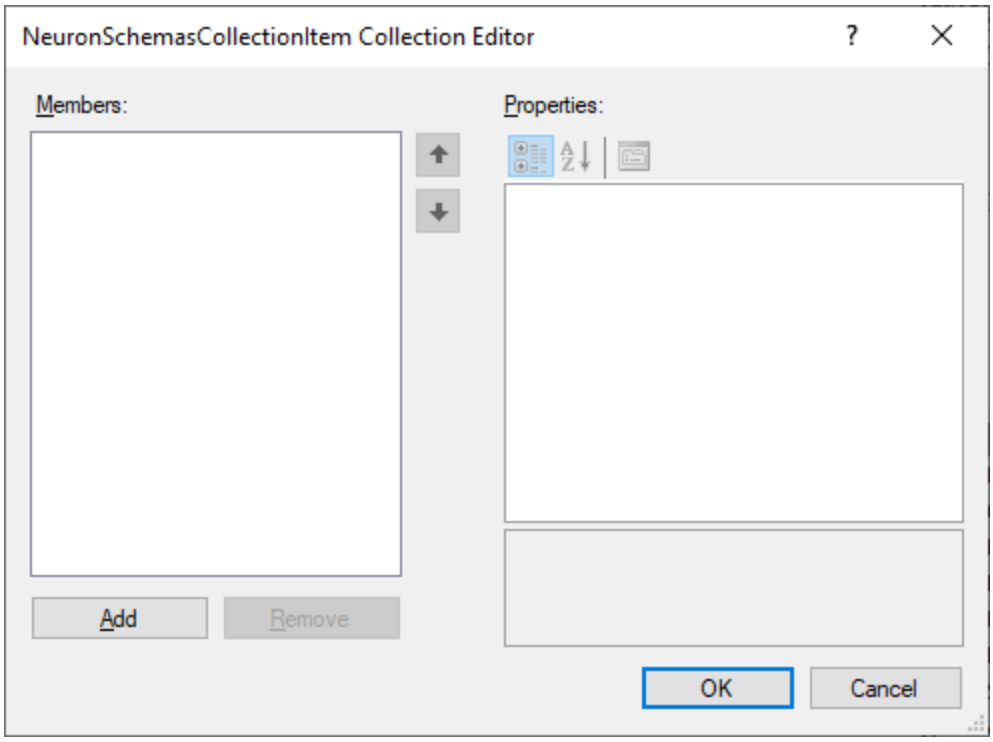
Click on the shape in the middle and you will see the Properties for the Process in the lower-right corner. Change the Name property to VETO and press Enter:



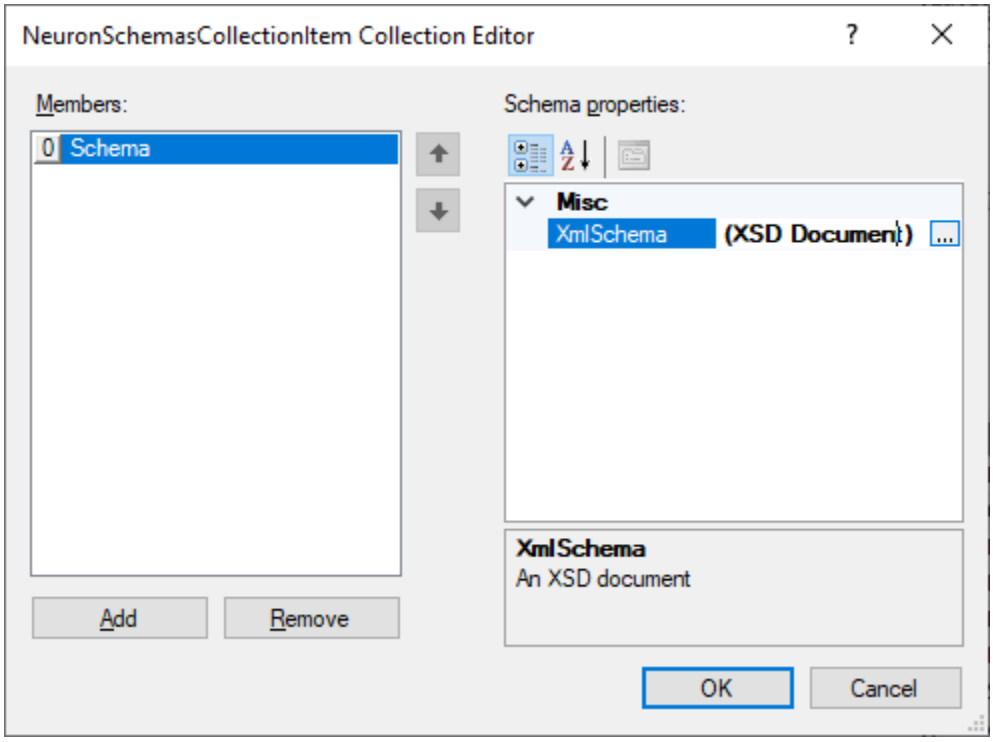
Now you are ready to start adding Process Steps to the designer surface. On the right side of the Neuron ESB Explorer is a complete list of all the built-in Process Steps that ship with Neuron ESB. Take a few minutes to look at the steps available. If you were to create any custom Process Steps and add them to Neuron ESB they would be listed here as well. Adding a Process Step to the designer is very easy: click on the step you want to add and drag it to the place on the designer surface to add it. To start building the Business Process, scroll through the Process Steps until you see the ***Validate – Schema*** step and drag it to the shape titled VETO in the Process Designer:



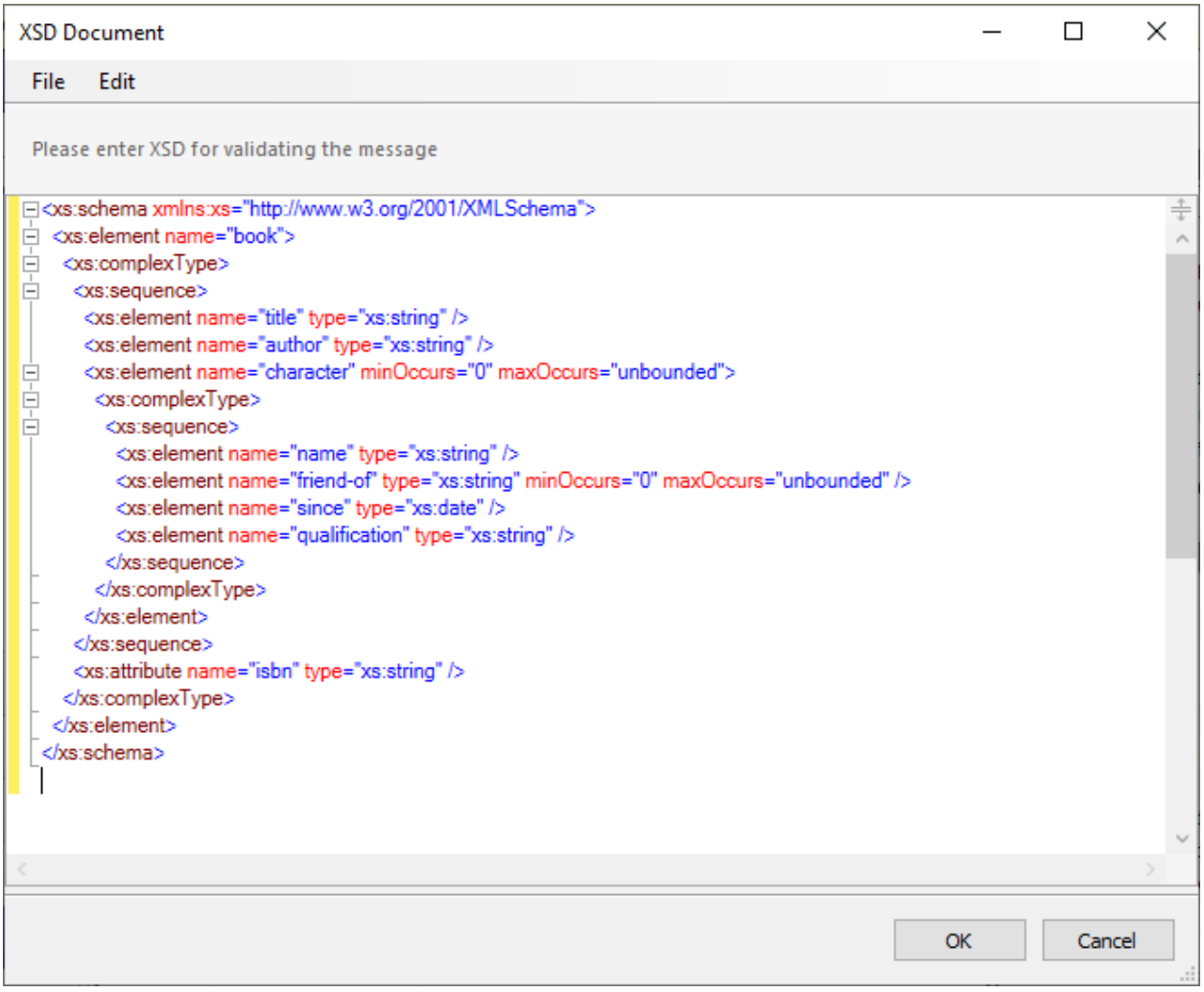
The ***Validate – Schema*** step takes the message received by the Neuron ESB party and validates it against a schema. To configure this step, click on the step where it says “Schema”. To specify a schema, click on the Schemas property in the property grid and then click on the ellipsis button. The NeuronSchemasCollectionItem Collection Editor will open:



Click the Add button and a new item will be added to the Members list on the left. Select the new item (Schema) to see the properties listed on the right. Select the XmlSchema property and click the ellipsis button to open the XML Document editor:



In the XML Document editor you can use the File menu to Open a schema or you can paste the contents of a schema directly into it:



The schema above can be found in the file BookSchema.xsd in the Projects folder included with this guide, or you can copy/paste this one:

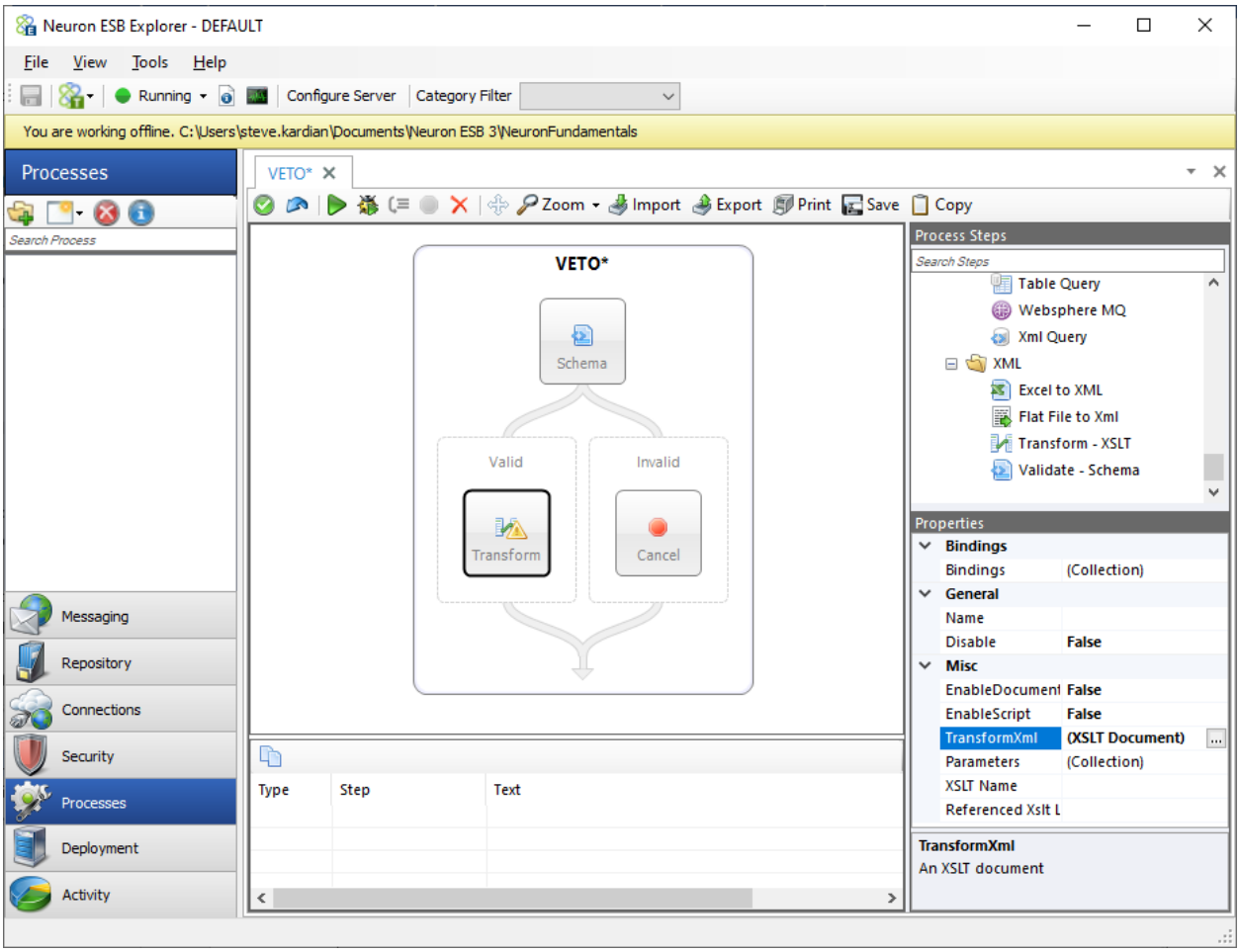
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string" />
        <xs:element name="author" type="xs:string" />
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element name="friend-of" type="xs:string" minOccurs="0" maxOccurs="unbounded"
/>
              <xs:element name="since" type="xs:date" />
              <xs:element name="qualification" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
```

```
      <xs:attribute name="isbn" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

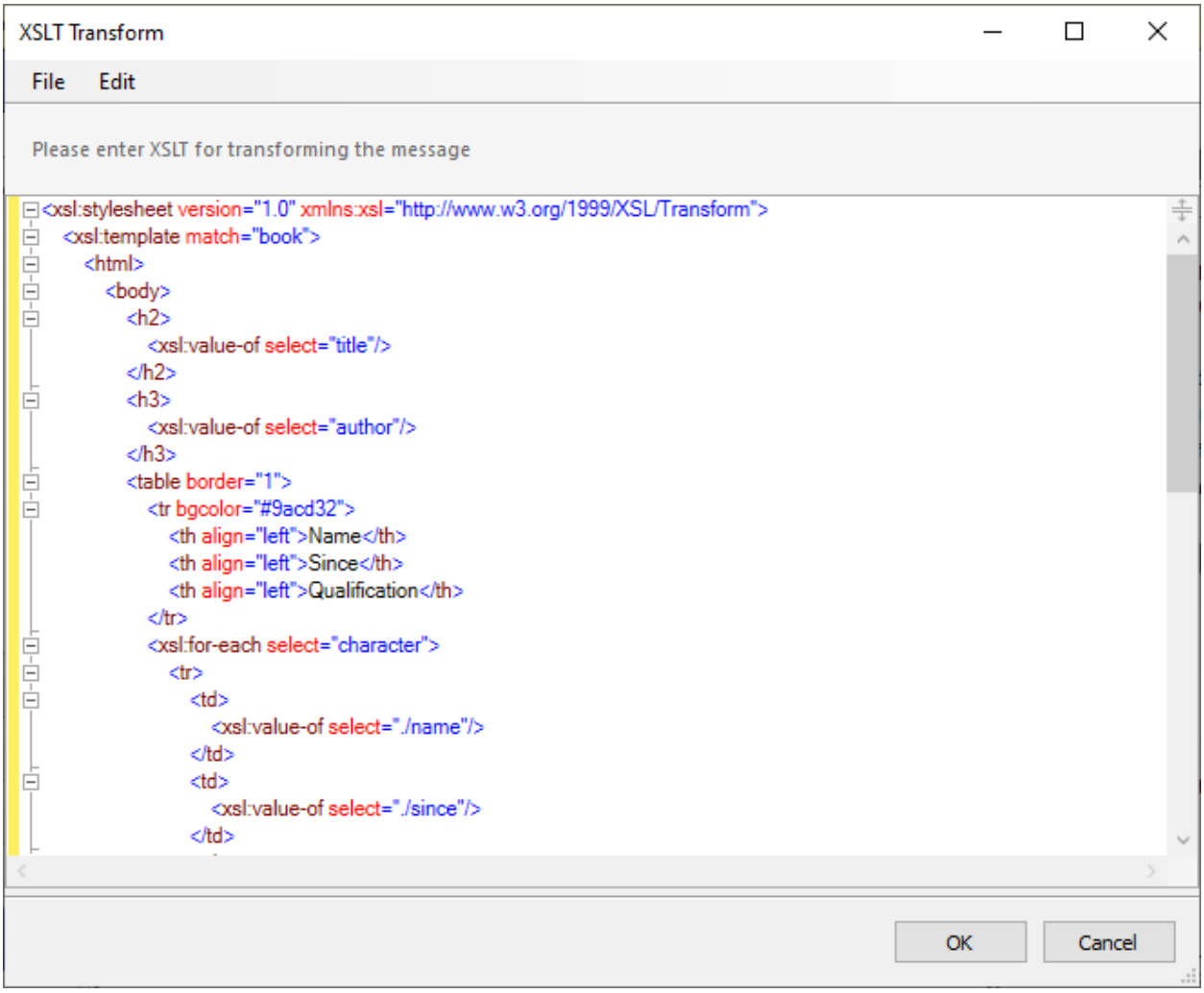
After the schema is added to the dialog, click OK twice to close the dialog and the schema collection editor. When you configured the ***Validate – Schema*** step you might have noticed that you can add more than one schema to the collection. This allows Neuron ESB to validate a message against a number of schemas at once. If you add multiple schemas to the collection, Neuron ESB will first attempt to match the root element and target namespace against each schema in the list. Only when it finds a match does Neuron ESB do a full schema validation. If the messages successfully validates against one of the schemas in the list, the process flow will continue through the "Valid" execution block. If the message fails validation against all the schemas, the process flow will continue through the "Invalid" execution block.

NOTE: The Validate-Schema process step can also be configured to directly reference XSD Schemas stored in the Neuron ESB Repository located within the Repository tab of the Neuron ESB Explorer.

Now that the ***Validate – Schema*** step has been configured we can add the transformation logic to our process. Scroll through the Process Steps until you find the ***Transform – Xslt*** step. Drag it onto the Process Designer, into the execution block for the "Valid" branch of the ***Validate – Schema*** step:



Just like with the Validate – Schema step, when you click on the ***Transform – Xslt*** step you will see a tooltip stating that something isn't correctly configured. In this case you need to add an Xslt. To configure the transformation, select the Transform step and then the TransformXml property. Then click the ellipsis button next to the property to open the XSLT Transform editor. Just like the Schema collection editor you can use the File menu to Open an Xslt or you can paste the contents of an Xslt directly into it:



The Xslt above can be found in the file BookTransformation.xslt in the Projects folder included with this guide, or you can copy/paste this one:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="book">
    <html>
      <body>
        <h2>
          <xsl:value-of select="title"/>
        </h2>
        <h3>
          <xsl:value-of select="author"/>
        </h3>
          <table border="1">
            <tr bgcolor="#9acd32">
              <th align="left">Name</th>
              <th align="left">Since</th>
              <th align="left">Qualification</th>
              </tr>
                <xsl:for-each select="character">
              <tr>
              <td>
                <xsl:value-of select="./name"/>
              </td>
              <td>
                <xsl:value-of select="./since"/>
```

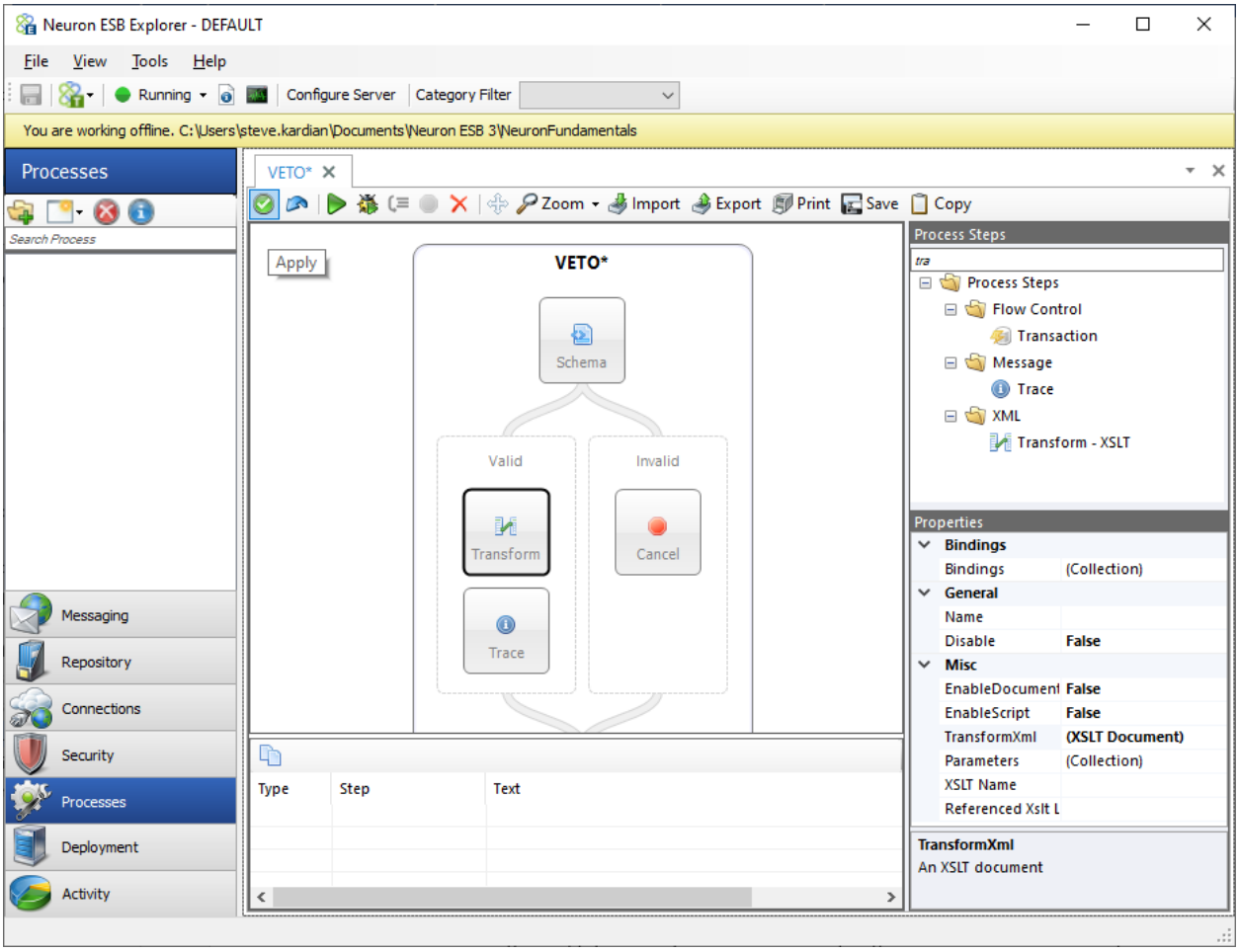
```
            </td>
            <td>
              <xsl:value-of select="./qualification"/>
            </td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

After the Xslt is added to the dialog, click OK to close the Xslt Transform editor.

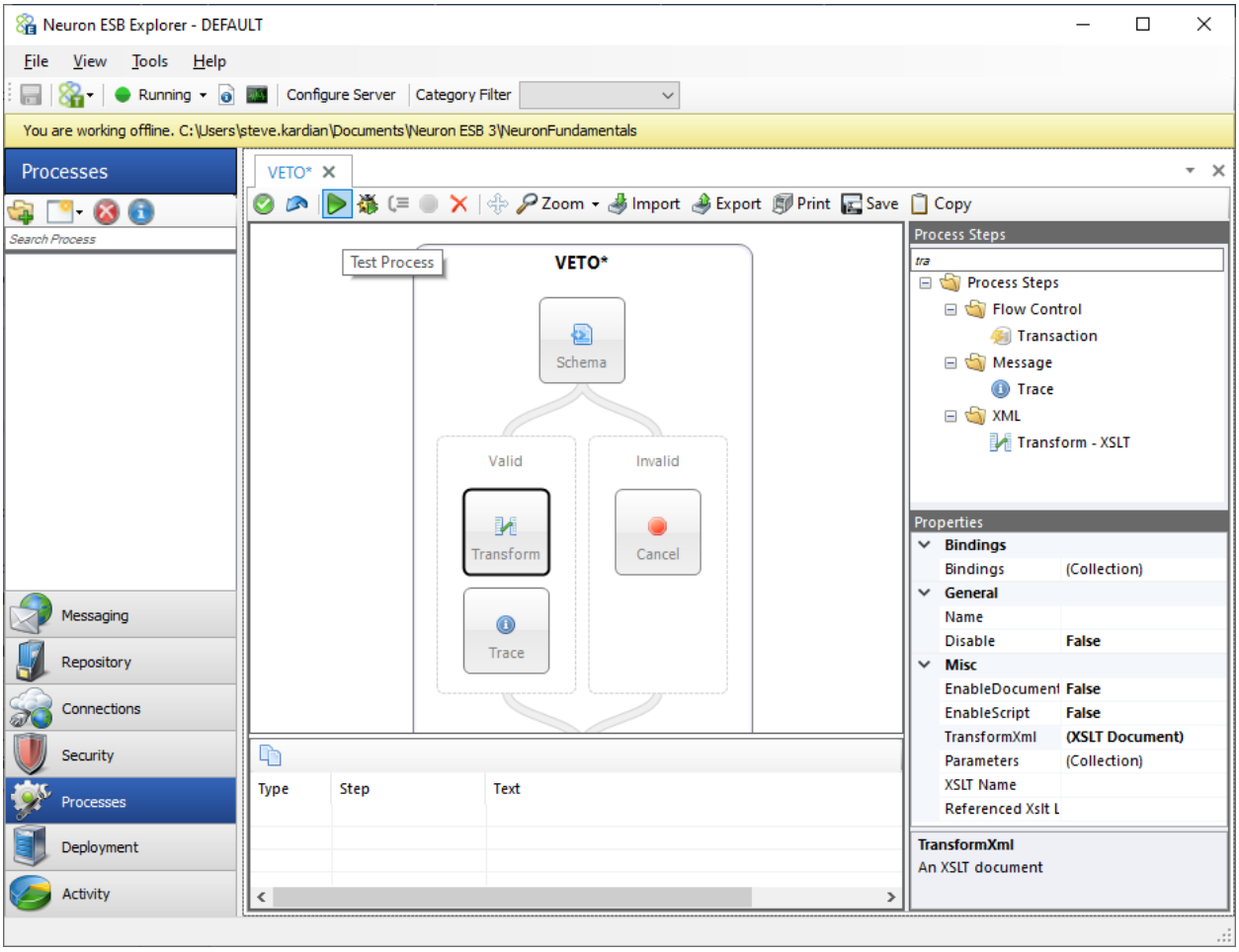
NOTE: The Transform – Xslt process step can also be configured to directly reference XSLT documents stored in the Neuron ESB Repository located within the Repository tab of the Neuron ESB Explorer.

To see the output of the ***Transform – Xslt*** step you can add a Trace step to the process. The ***Trace*** step behaves differently depending on the environment it's being used in. If you are testing in design mode, the ***Trace*** step outputs the contents of the message to the output window for viewing. During runtime mode the ***Trace*** step will output the contents of the message to the Neuron log files (if the ESB Service is configured to trace level of at least Info). To add the ***Trace*** step, scroll through the Process Steps until you find it and drag it onto the Process Designer right underneath the ***Transform -Xslt*** step. If you select the ***Trace*** step and look at the properties, you will notice that it's only property is Name.

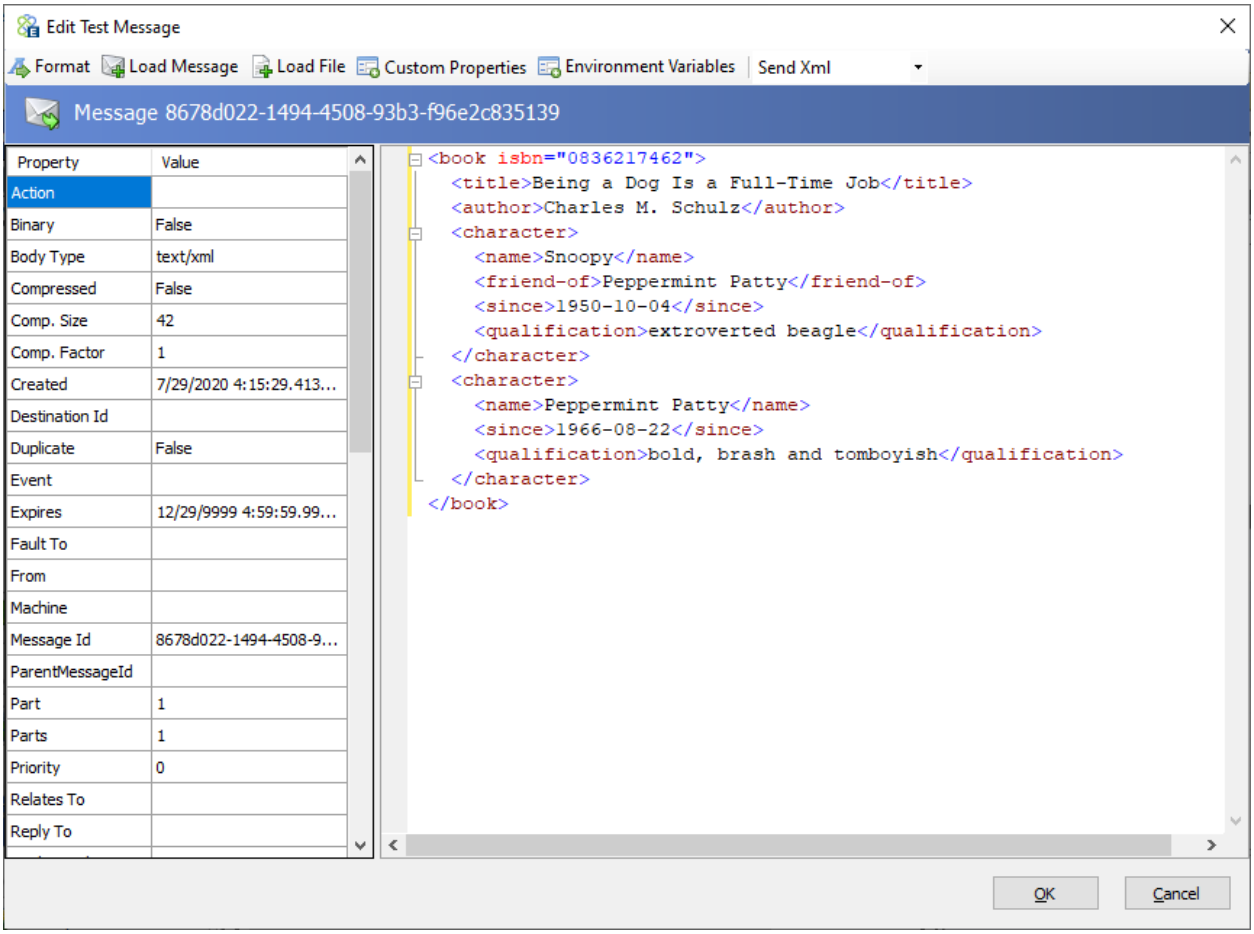
The last step to developing this process is to take action when the messages fail validation. For this example, we are just going to throw an exception which will be returned to the sender as well as logged in the Neuron ESB v3 event log and the Neuron log files. Throwing an exception in a ***Validate – Schema*** step is very easy. When the message fails validation, the reason for failure is automatically saved. All you have to do is add a ***Rethrow*** Process Step to the "Invalid" branch to have this failure logged as an exception. Scroll through the Process Steps until you find the ***Rethrow*** step. Drag it onto the Process Designer, into the execution block just above the ***Cancel*** step in the "Invalid" branch. Select this step; you will notice that the only configurable properties are the Name and Disable property. Because the ***Rethrow*** step will cause execution to stop, you can delete the ***Cancel*** step that follows it. To do this, right-click on the ***Cancel*** step and select Remove. Your Business Process should now resemble this:



Click the Process Designer “Apply” button (highlighted in blue above). A great feature in Neuron ESB Explorer is the ability to test processes directly from the Process Designer. To test a process, click on the “Test Process” button (highlighted in blue below):



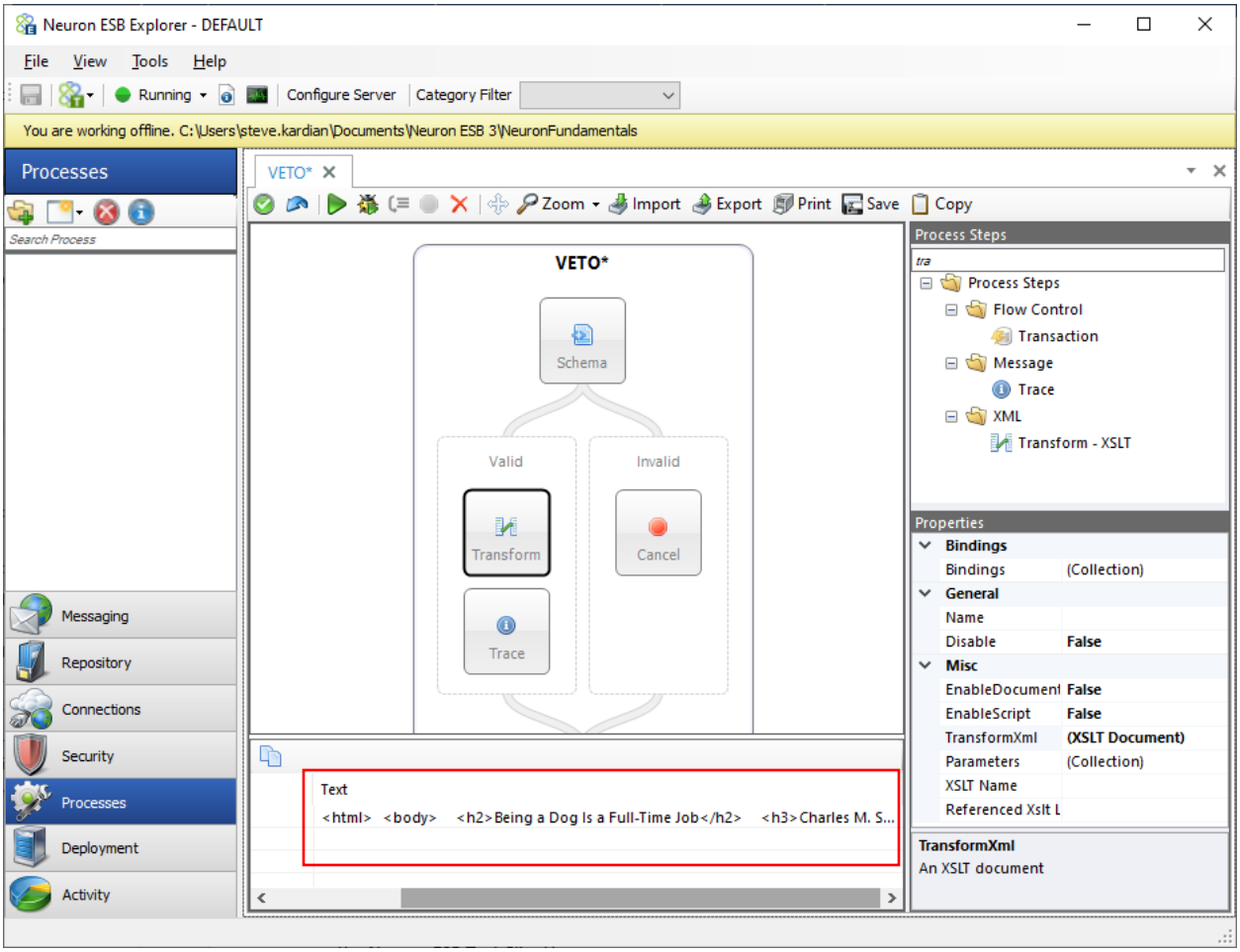
The Edit Test Message dialog will open. From this dialog you can enter any message you want to test with your Business Process. You can also set message header properties and custom message properties. For this test we only need to modify the message itself. You can either type the message into the message pane or you can click the Load File button circled in Red below and open the test message BookSampleData.xml included in the Projects folder:



If you want to copy/paste the XML, use this:

```
<book isbn="0836217462">
  <title>Being a Dog Is a Full-Time Job</title>
  <author>Charles M. Schulz</author>
  <character>
    <name>Snoopy</name>
    <friend-of>Peppermint Patty</friend-of>
    <since>1950-10-04</since>
    <qualification>extroverted beagle</qualification>
  </character>
  <character>
    <name>Peppermint Patty</name>
    <since>1966-08-22</since>
    <qualification>bold, brash and tomboyish</qualification>
  </character>
</book>
```

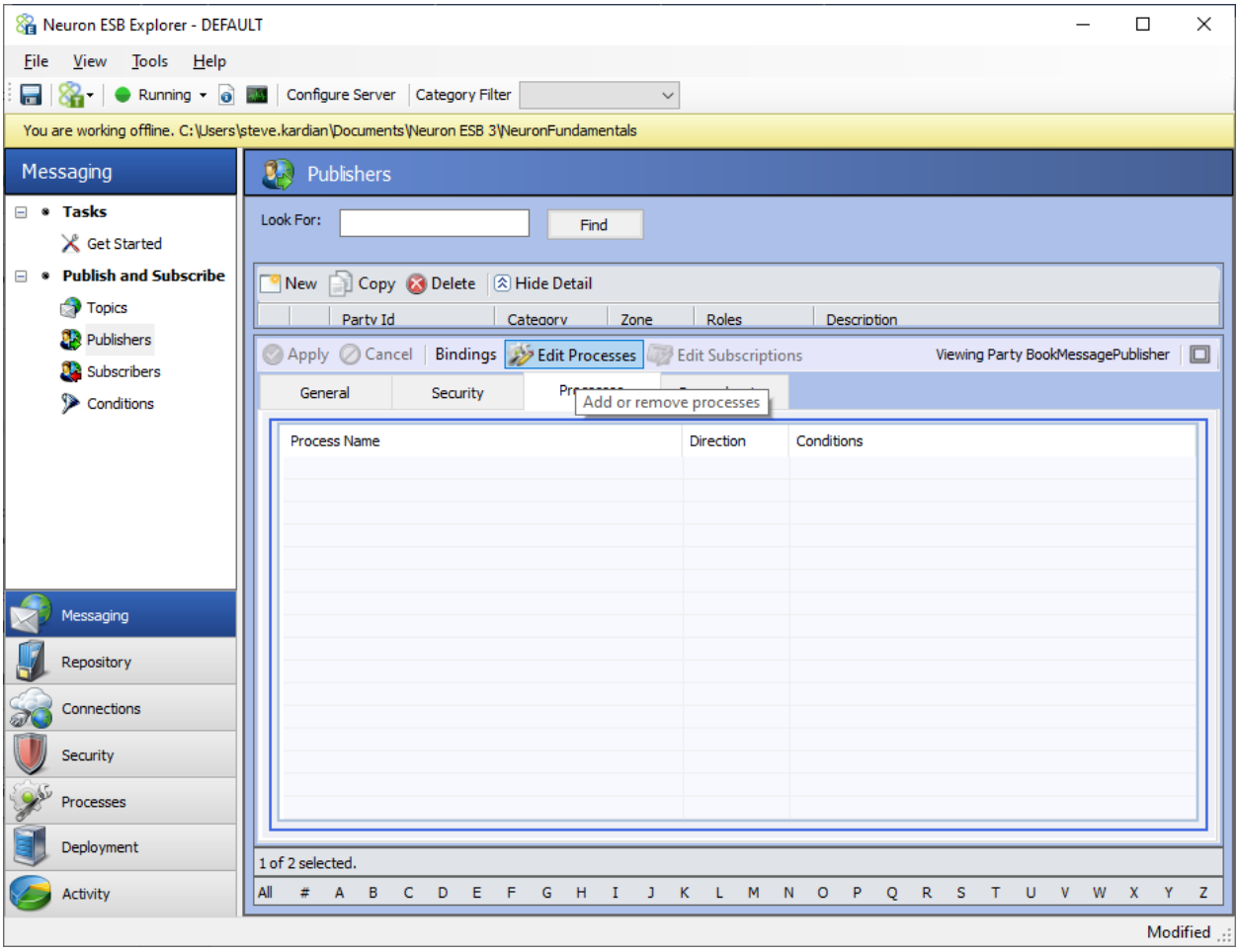
Click the OK button to begin the test. The dialog will close, and each step will be highlighted in Green as the Process is executed. When the execution reaches the Trace step, the contents of the newly transformed message is written to the output window:



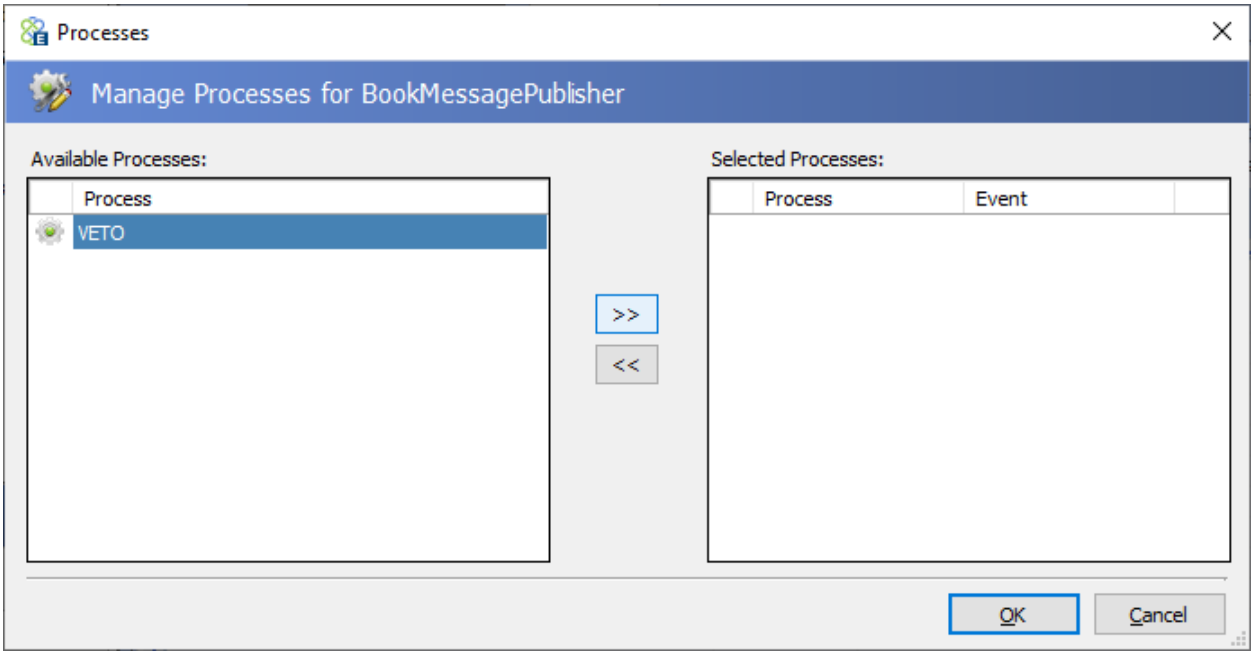
Now that you've created and tested a Business Process, you're ready to attach it to a party and test with the Neuron ESB Test Client!

Full documentation to the Business Process Designer and Business Process Steps can be found here:
https://www.neuronesb.com/article/kb/business-processes-overview/

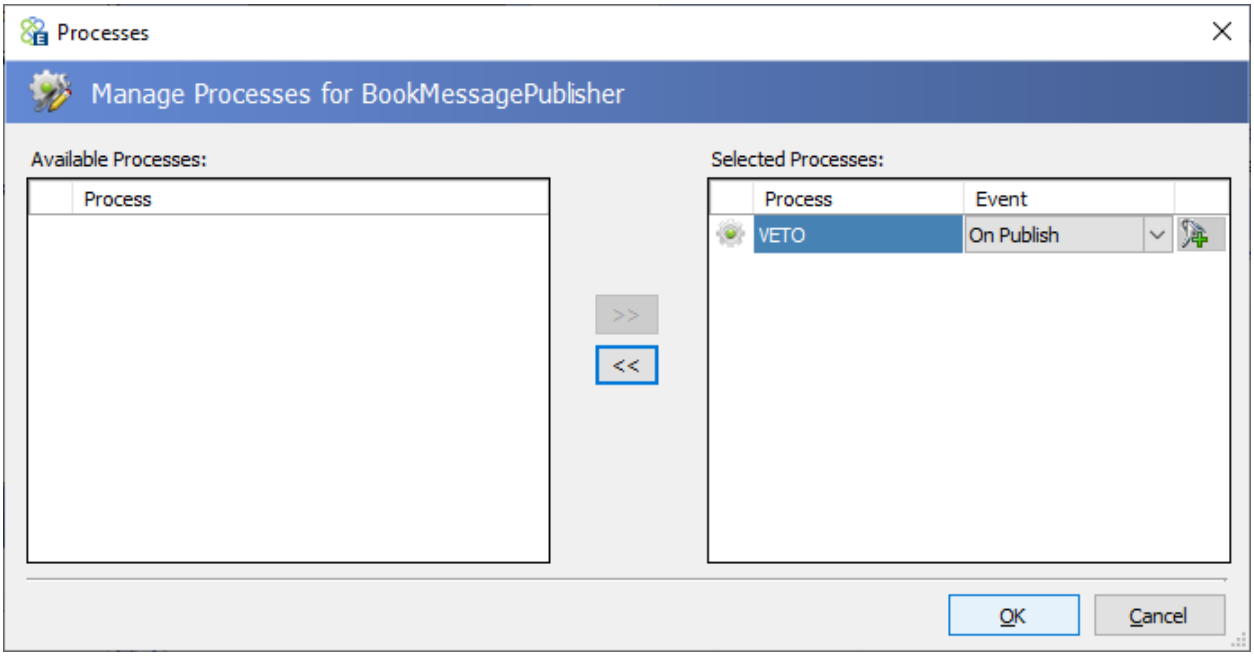
In Neuron ESB Explorer, navigate to Messaging and then Publishers. Select the BookMessagePublisher from the list and the click the Edit Processes button:



Click the Edit Processes button to open the Processes dialog:

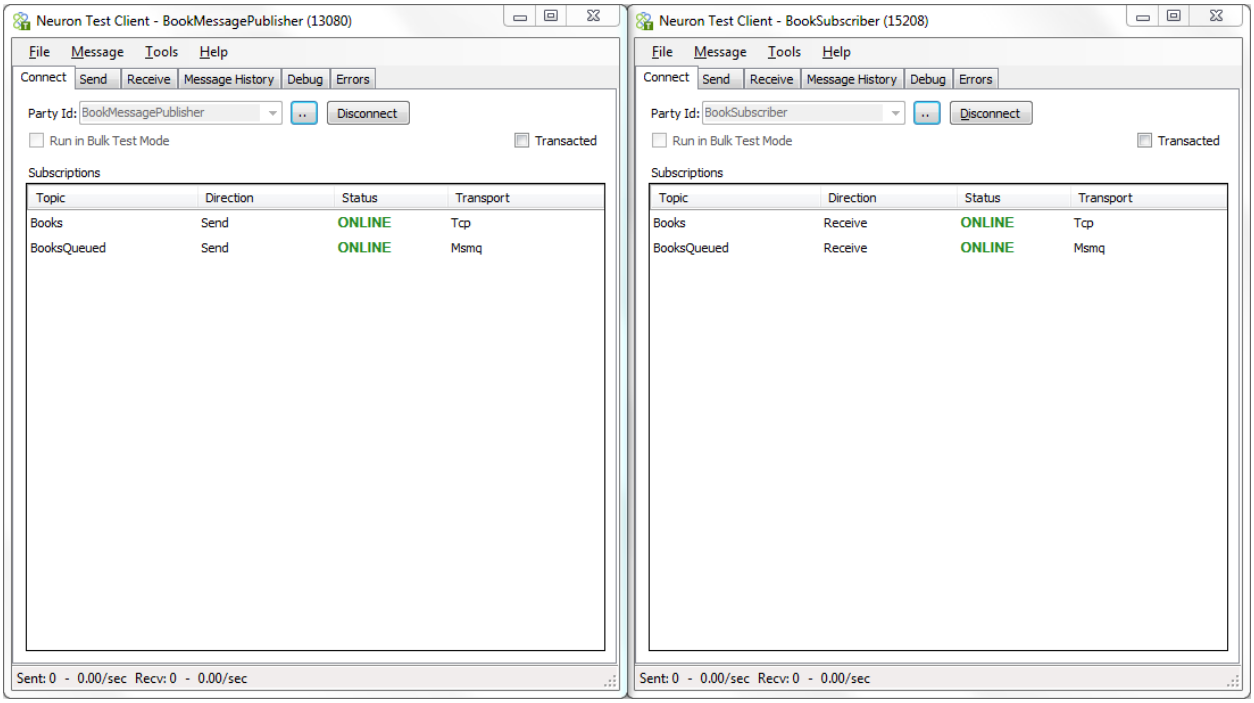


Select the VETO Process from the available processes list and click the arrow button that points to the right (highlighted in blue above). The VETO process is added to the list of current processes for this party, and because this party was selected from a list of Publishers, the On Publish event is the default event. This instructs Neuron ESB to execute the Business Process when the BookMessagePublisher receives a new message from an external source and right before it publishes the message to the ESB. All parties, regardless of whether they are publishers or subscribers, can execute Processes on either the On Publish or On Receive events, or both.

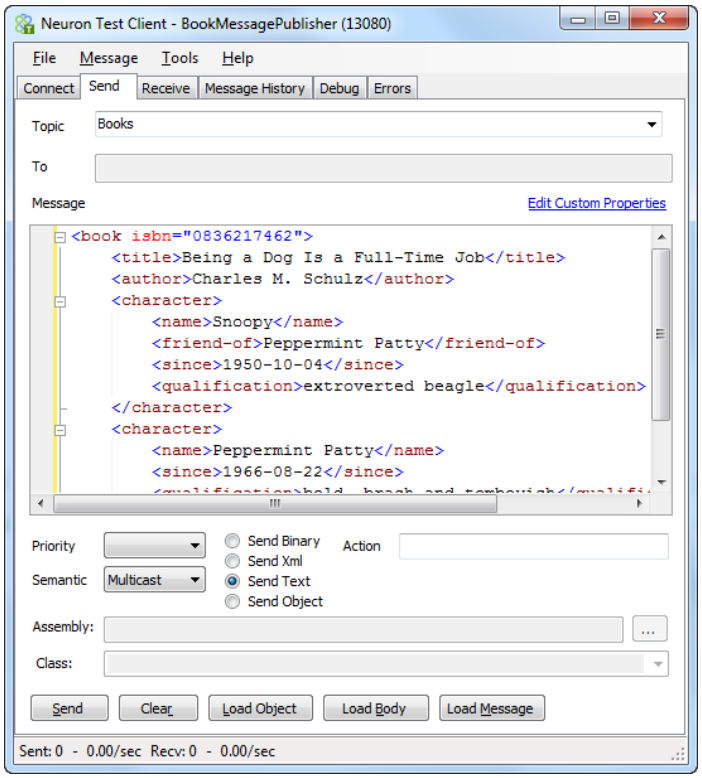


Click OK and then Apply the changes to the party. Now Save the Neuron ESB Configuration.

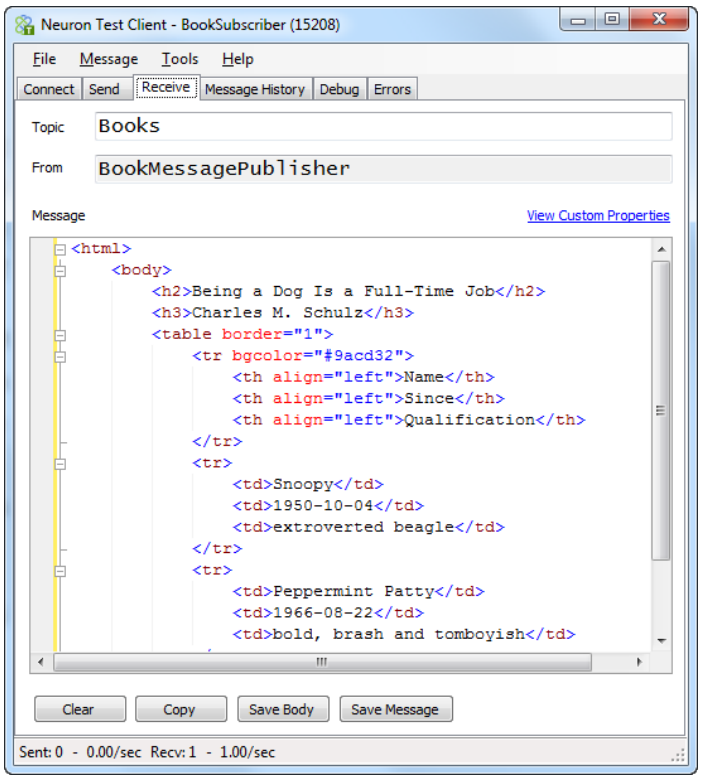
Open two Neuron ESB Test Clients from the Neuron ESB Explorer by clicking Tools->Test Client->2 Test Clients. Configure one test client to use the BookMessagePublisher and the other to use the BookSubscriber.



Click on the Send tab for the BookMessagePublisher test client and paste the XML into the Message pane (or use the Load Body button to load the message from the Exercises folder). Click the Send button:



You should see a received message in the BookSubscriber test client similar to below. To “pretty print” the Xml in the test client, select Message->Format XML from the menu:



You have now successfully created and tested a Business Process in Neuron ESB!

Neuron ESB Client API

Up to this point you have configured Topics, Parties and Transports and tested this configuration using the Neuron ESB Test Client. The final section of this training will involve peeling away the layers you have been working with and programming directly with the Neuron ESB Client API. This section assumes you have Visual Studio and are familiar with programming .NET applications.

The Neuron ESB Client API is an event based, distributable API that can be hosted in .NET applications to publish and receive messages to and from Topics. Topics are the core of the Neuron ESB messaging sub system and are hosted by the Neuron ESB runtime windows service. At the core of the Neuron ESB Client API are the Publisher and Subscriber .NET classes, which are derived from the Party class. Generally, users will create either an instance of Publisher or Subscriber class (or Party class) in .NET code, Connect to the bus and either send or receive messages.

Creating an instance of either the Publisher or Subscriber requires passing in several parameters listed in the table below:

SubscriberId	Name of the Publisher or Subscriber object as defined within the Neuron ESB Explorer

Zone	The name of the Zone defined within the Neuron ESB Explorer. It can be found by navigating to Deployment-> Settings-> Zone. By default, this should always be "Enterprise" and never change.
ServiceAddress	Sometimes called the "Bootstrap" address. This is the net.tcp address for the Neuron ESB server and its configuration port. The default address should be "net.tcp://localhost:50000". The actual Port number can be changed by navigating to the Port tab on Deployment-> Settings-> Zone screen within Neuron ESB Explorer. This is the address that a Party will initially establish a connection with to download its respective configuration so that it knows how to and which Topics to Connect to. NOTE: If Port Sharing is enabled the ServiceAddress must be appended with the name of the Neuron ESB runtime instance. For example: "net.tcp://localhost:50000/Default" https://www.neuronesb.com/article/port-sharing/
ServiceIdentity	This is used only under some circumstances when Kerberos or delegation is configured and the Neuron ESB runtime windows service is set to run under a specific domain user account. In which case the serviceIdentity can be a UPN value i.e. **`upn:CORP\DomainUser.`**

The parameters can be passed either by embedding them within the app.config file (demonstrated in the Practice section of this document) of your project, or by specifically declaring them using the SubscriberConfiguration class. The example below demonstrates how to create an instance of a Publisher object, using the SubscriberConfiguration class:

```
SubscriberConfiguration config = new SubscriberConfiguration("AccountPublisher", "Enterprise",
                                                "net.tcp://localhost:50000", "");
using (Publisher client = new Publisher(config))
{

}
```

Although the example above created the Publisher object within a "using" statement block, its common to create these types of objects at a more global level and then manage calling "Dispose()" on the object when you ready to disconnect from the bus.

Once an instance of the Publisher or Subscriber class is created, you can wire in any number of events for notification if something of interest occurs. There are several events, some of which are listed in the table below.

OnOnline	This will fire once for each Topic that the client (Publisher or Subscriber) successfully connects to and receives an online event from the Neuron ESB Server. The client can only send messages to Topics that have generated an Online event. A client can be "connected" to a Topic yet still be in an Offline state.
OnOffline	This will fire once for each Topic that the client receives an Offline event from the Neuron ESB Server. For example, if the Topic's publishing service is shut down or disabled on the server side, the client will

	receive an Offline notification. This will cause any message sent to that specific Topic from the client to fail.
OnReceive	This is usually used by Subscriber objects to receive messages in real time as they are published to the bus. The OnReceive handler will pass both the message as well as an instance of the current Subscriber object.
OnSend	Whenever a message is published, the OnSend event handler will be invoked. This event occurs just before the message is actually sent and therefore, should not be used as confirmation of a sent message.
OnSubscriberDisabled	This will fire if the client is disabled within the Neuron ESB Explorer on the server side.
OnConfigurationChanged	By default, clients communicate with the Neuron ESB runtime windows service every 15 seconds to determine if the current running ESB Configuration has changed. This can be changed by navigating to the Server Tab located on the Deployment-> Settings-> Zone screen within Neuron ESB Explorer. For example, someone may change the Topics, or the Topic configuration or the Processes associated with the client. If that happens, the client will detect and download the changes and refresh its runtime state. If the client detects such changes, this event will fire.
OfflineNeuronWsDiscovery	Occurs when online instance of runtime shuts down – WS-Discovery broadcast message
OnlineNeuronWsDiscovery	Occurs when online instance of runtime starts up...and occurs on a regular interval defined by esbservice.exe.config – WS-Discovery broadcast message

After an instance of either a Publisher or Subscriber class is created, it must “Connect” to the Neuron ESB runtime windows service i.e. the “bus”. The connection process does several things, first and foremost it establishes a secure connection with the bus and downloads the sections of the Neuron ESB Configuration that it needs to finish connecting to Topics as well as to later run any specific Business Processes that may be attached to it. Next, it will establish connections to specific sub systems within the bus, like the Neuron Auditing service (if being used). Lastly, it will establish a connection to each Topic’s publishing service hosted by the bus. The example below demonstrates connecting to the bus while examining any errors during the connection process:

```
using (Publisher publisher = new Publisher(config))
{

   // catch any exceptions that may occurr while connecting to each individual topic
   PartyConnectExceptions exceptions = publisher.Connect();
   if (exceptions != null && exceptions.Count > 0)
   {
      foreach (var e in exceptions.GetResults())
            Console.WriteLine(string.Format("An error occurred connecting '{0}' to Topic, '{1}'. {2}",
            publisher.Context.PartyId, e.Exception.Message, e.Topic));
   }
}
```

Each connected Topic is represented by its own individual connection and state within the client. Hence, within the internals of the Publisher/Subscriber object the collection of Topic connections is represented

as a collection of ESBTopicContext objects. For example, the connected and online state of a specific Topic connection could be determined using the example below before a client publishes a message to the bus:

```
Neuron.Esb.Channels.ESBTopicContext topicContext =
        publisher.Context.TopicContexts[Neuron.Esb.Internal.ESBHelper.TopicRoot("Accounts.Loan")];
if (topicContext.IsOnline && topicContext.Connected)
 {
 }
```

After a client has connected to the bus, it can either send and or receive messages. There are several "Send" methods off of the Publisher class for sending a variety of different messages types. All Send methods publish data to a specific Topic supplied at runtime. The most common Send methods are listed in the table below:

SendXml	Used to publish XML data
SendMessage	Used to publish a Neuron ESB Message. An ESB Message is a special class that can encapsulate any data type. It has a rich set of header properties which govern how the bus will operate and route the message. For example, the Topic to publish the message to, as well as the message pattern can be set as a property within the class. Additionally the ESB Message supports the use of "custom" properties or meta data. Custom meta data can be added at runtime and live with the message through its duration on the bus. This data can be accessed, audited as well as modified within a Neuron Business Process or Workflow.
Send	Used to publish any binary or .NET Serializable object.

In the example below, the client publishes both a System.DateTime object as well as XML data to the bus. For the latter, the Semantic property has been changed from its default of Multicast (indicating Asynchronous messaging) to Request (indicating Synchronous messaging). When changed to Request, the Send method will block the caller until it either receives a response message from the receiving subscriber or the request times out:

```
#region Send a .NET serializable object like datetime
// Send anything .net serializable like date time or custom class
publisher.Send("Accounts.Loan", DateTime.Now);
#endregion

#region Send a simple request message that will return a reply
// this means that the subscriber must send back a reply message
ESBMessage response = publisher.SendXml("Accounts.Loan","<Test>MyRequest</Test>",
        "",SendOptions.Request);
#endregion
```

When subscribing to messages from the bus, the OnReceive event handler must be used. Within the event handler, messages will be received as they are published to and routed by the bus. Each time a message is published a list of eligible subscribers is computed (in memory) either at the server level or in some cases, within the Publisher object instance (depending on the Transport configuration of the Topic that the message is published to). Once a message is received, its data type can be inspected and the message can be cast into its specific datatype using the GetBody<>() method of the ESBMessage class as in the example below:

```
private static void OnReceive(object sender, MessageEventArgs e)
{
    ESBMessage message = e.Message;

    if (message.Header.BodyType != null)
    {
        if (message.Header.BodyType.Equals("text/xml"))
            Console.WriteLine(message.GetBody<string>());

        if (message.Header.BodyType.Equals("DateTime"))
            Console.WriteLine(message.GetBody<DateTime>().ToString());

        if (message.Header.BodyType.Equals("TestSerialization"))
            Console.WriteLine(message.GetBody<TestSerialization>().Name);
    }
    else
    {
        Console.WriteLine(message.Text);
    }
}
```

Lastly, if a Topic is configured with a Transaction Transport such as MSMQ, the ambient transaction can be used within the receive handler to roll the received message all the way back to its underlying Queue for another redelivery attempt:

```
private static void OnReceive(object sender, MessageEventArgs e)
{

    Transaction tx = System.Transactions.Transaction.Current;
    if (tx == null) Console.WriteLine("    null transaction");
    else Console.WriteLine("   transaction exists");

    tx.Rollback();

}
```

Excercise - Using the Client API

Open Visual Studio and create a new Console Application called NeuronAPITraining

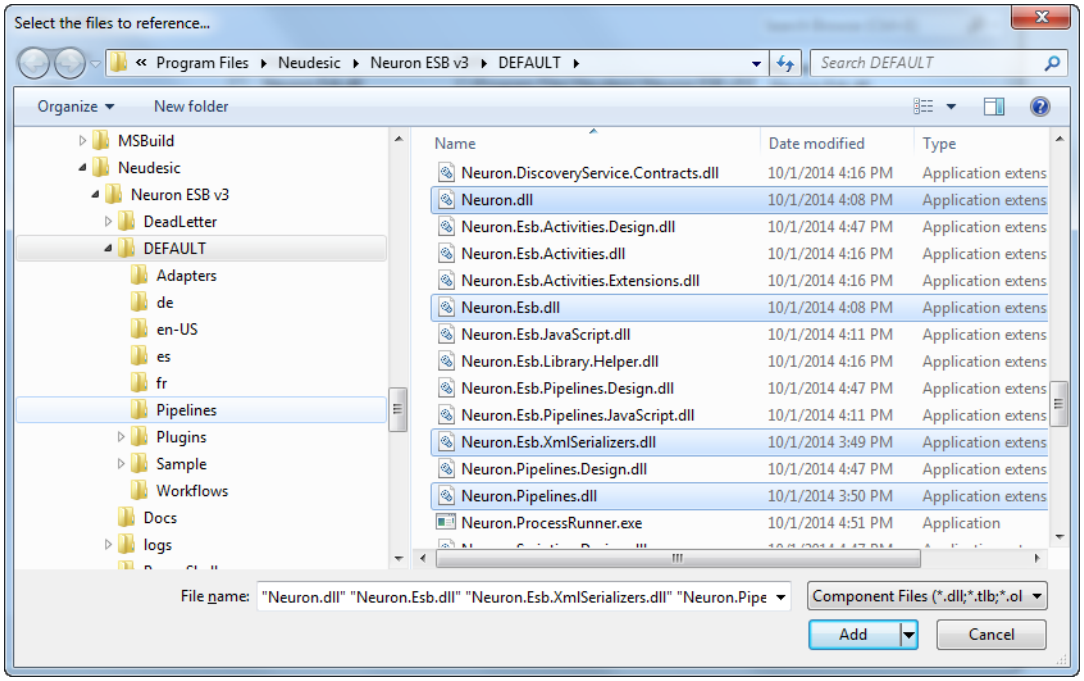
Right client the Project and select Properties. On the Application Tab, verify the Target Framework is set to .NET Framework 4.7.2. When prompted to verify the change, press Yes.

Right click the Project, Add Reference and use the Browse option to navigate to the Neuron program files folder (C:\Program Files\Neudesic\Neuron ESB v3\DEFAULT).

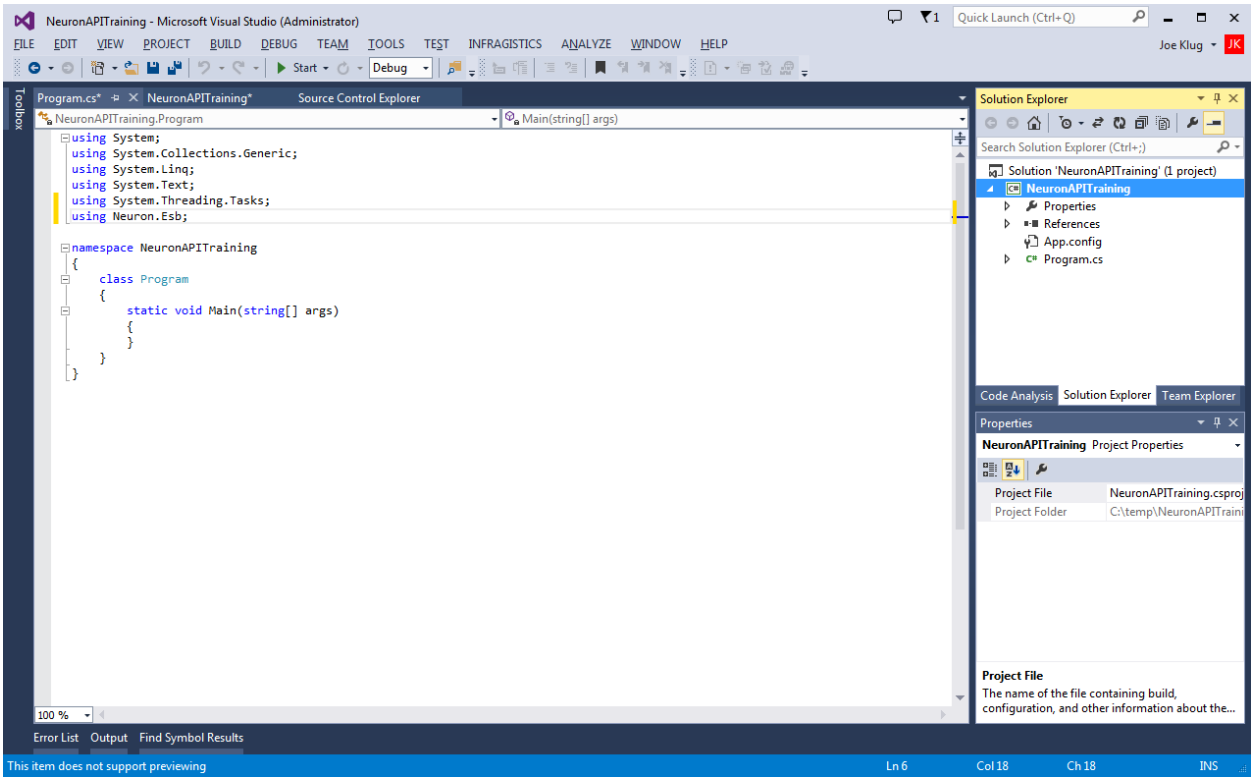
Highlight the following assemblies:

- Neuron.dll
- Neuron.Esb.dll
- Neuron.Esb.XmlSerializers.dll
- Neuron.Pipelines.dll

- Neuron.Scripting.dll



Click “OK” to add the references and add a *using* statement for Neuron.Esb to the top of the Program.cs file.



If not already present, add an Application Configuration file to the project and add the following entries:

```
<appSettings>
    <add key="esbZone" value="Enterprise"/>
    <add key="esbServiceAddress" value="net.tcp://localhost:50000/"/>
    <add key="esbServiceIdentity" value=""/>
</appSettings>
```

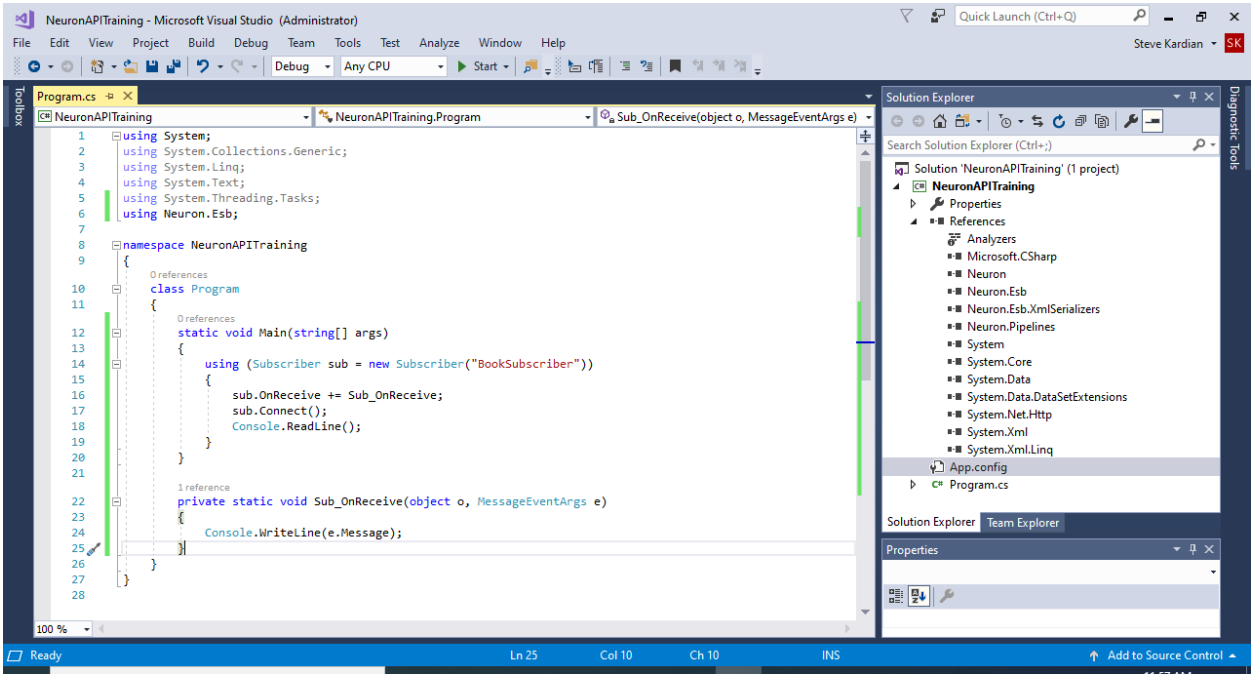
Add the following code to the main method. For the OnReceive event, if you press TAB, IntelliSense automatically completes the statement for you and displays the event handler reference as selected text in the Code Editor. To complete the automatic event hookup, IntelliSense prompts you to press the TAB key again, to create an empty stub for the event handler.

```
static void Main(string[] args)
{
        using (Subscriber sub = new Subscriber("BookSubscriber"))
        {
                sub.OnReceive += Sub_OnReceive;
                   sub.Connect();
                   Console.ReadLine();
        }
}
```

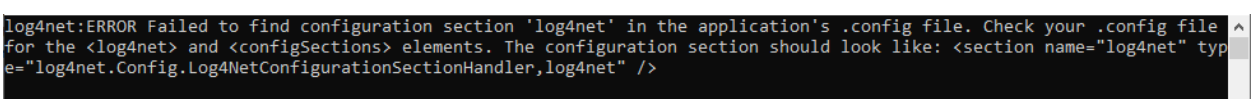
Replace the auto-generated code in the event handler with the following:

```
        Console.WriteLine(e.Message);
```

Your project should now look like this:



Before we run our project we have to add a configuration section for log4net to the App.config file. If you do not add this section you will receive the following console screen when running the project.



Add the following section to your App.config file

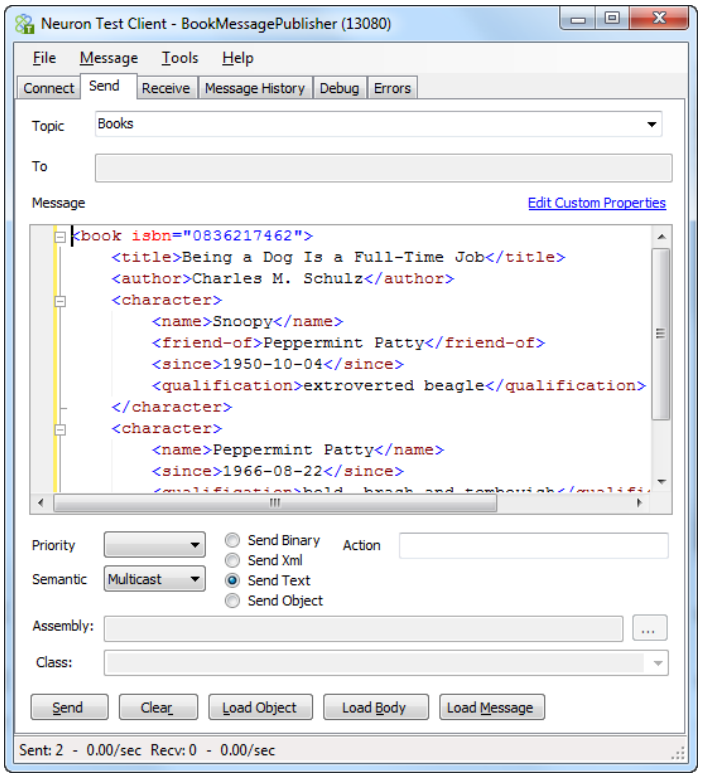
```
<log4net>
	<appender name="EventLogAppender" type="log4net.Appender.EventLogAppender">
		<logName value="Application"/>
		<applicationName value="ESBv3_DEFAULT"/>
		<layout type="log4net.Layout.PatternLayout">
			<conversionPattern value="%message%newline%exception"/>
		</layout>
		<filter type="log4net.Filter.LevelRangeFilter">
			<levelMin value="WARN"/>
			<levelMax value="FATAL"/>
		</filter>
	</appender>
	<root>
		<level value="INFO"/>
		<appender-ref ref="EventLogAppender"/>
	</root>
</log4net>
```

Hit F5 to save and build the project. A window should appear with your Console Application running. From the Neuron ESB Explorer, open a single Test Client and connect as BookMessagePublisher. Arrange the windows so you can see both your Console Application and the Test Client.

Click the Send Tab in the Test Client. Change the Topic using the Topic dropdown to Books. Copy and paste the XML below into text area.

```
<book isbn="0836217462">
  <title>Being a Dog Is a Full-Time Job</title>
  <author>Charles M. Schulz</author>
  <character>
    <name>Snoopy</name>
    <friend-of>Peppermint Patty</friend-of>
    <since>1950-10-04</since>
    <qualification>extroverted beagle</qualification>
  </character>
  <character>
    <name>Peppermint Patty</name>
    <since>1966-08-22</since>
    <qualification>bold, brash and tomboyish</qualification>
  </character>
</book>
```

Finally, press Send. You should see something similar to the following:




```
<html>
  <body>
    <h2>Being a Dog Is a Full-Time Job</h2>
    <h3>Charles M. Schulz</h3>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th align="left">Name</th>
        <th align="left">Since</th>
        <th align="left">Qualification</th>
      </tr>
      <tr>
        <td>Snoopy</td>
        <td>1950-10-04</td>
        <td>extroverted beagle</td>
      </tr>
      <tr>
        <td>Peppermint Patty</td>
        <td>1966-08-22</td>
        <td>bold, brash and tomboyish</td>
      </tr>
    </table>
  </body>
</html>
```

Congratulations! You've just written your first application using the Neuron API.

But we're not done. You are now going to modify the code so that depending on the Semantic requested by the original Party you will optionally return a response if required.

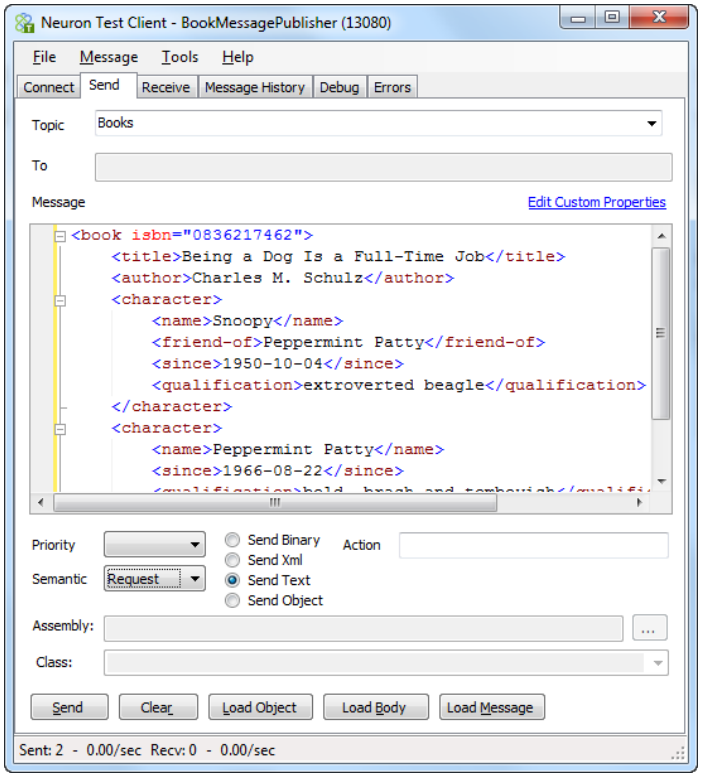
Stop your Console Application and add the following code to the sub_OnReceive method:

```
if (e.Message.Header.Semantic == Semantic.Request)
{
    ESBMessage retVal = e.Message.CreateReplyMessage();
    retVal.FromString("Message Received!");
```

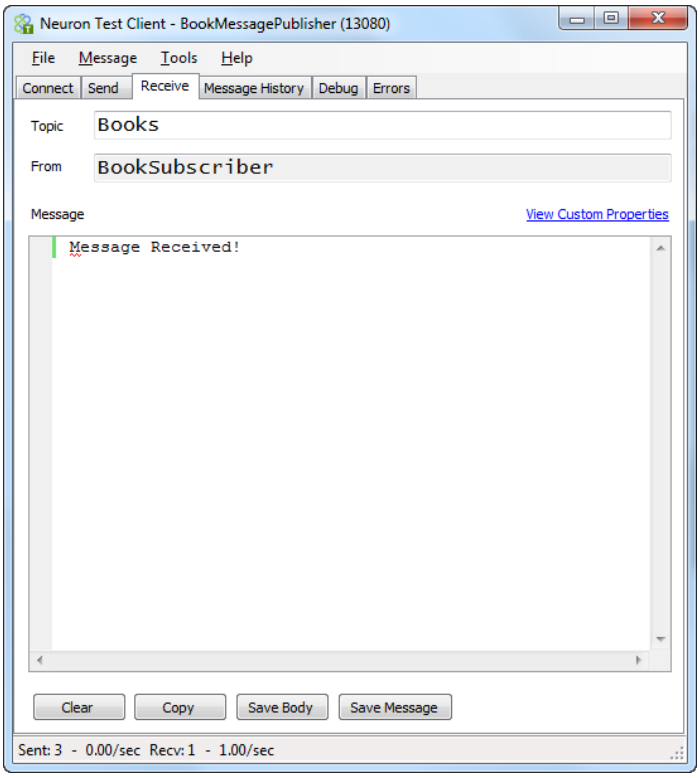
```
    Subscriber sub = o as Subscriber;
    sub.SendMessage(retVal);
}
```

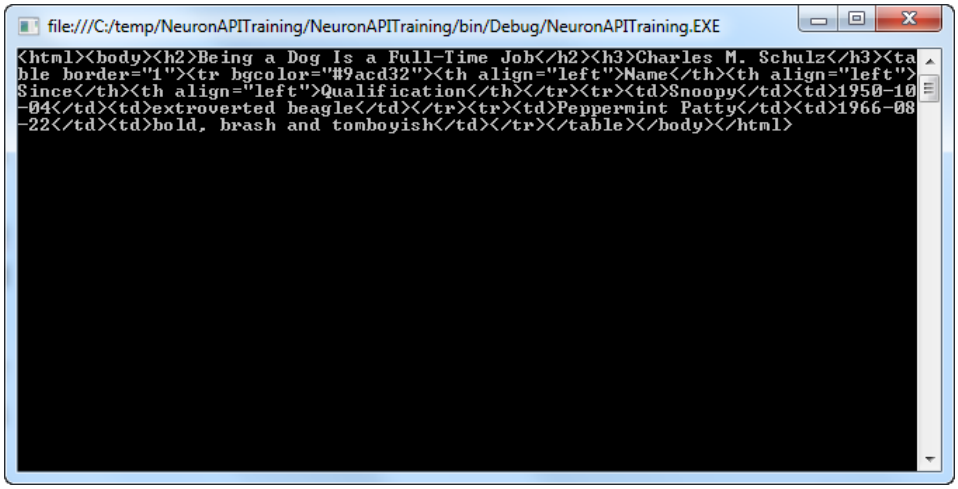
Now hit F5. Once again, arrange the windows so you can see both the Test Client and the Console Application.

In the Test Client Send Tab change the Semantic to Request.



Press Send. This time you should not only see the transformed message appear in your Console Application but you should also see the response appear in your Receive tab as well.





Using a Request Semantic means the client is expecting a synchronous reply to its message. When this Semantic is used the client will block until it receives the response or until a timeout occurs.

You do not have to use synchronous messaging. BookMessagePublisher and BookSubscriber can both be configured to send to the Books Topic using a Multicast Semantic and both Parties will receive the message. Synchronous messaging is useful when you are implementing a scenario where the Request Reply Message Exchange Pattern is the norm such as in Neuron ESB Client Connectors which publish SOAP/REST requests to the bus and receive SOAP/REST responses from Neuron ESB Service Connectors.

Review

Neuron ESB is a .NET based Application, Service and Workflow Integration Server which utilizes publish / subscribe as its foundation to deliver the functionality of Message Oriented Middleware, a Services Intermediary and Enterprise Application Integration.

Messages flow over Topics in Neuron ESB and are published and subscribed to by Parties. Parties can be publishers, subscribers or both.

Topics can have their Transport changed to deliver a different Quality of Service depending on scenario.

Messaging in Neuron ESB can be synchronous or asynchronous. This is controlled by the Semantic property contained in the header of the ESB message.

Messages can be manipulated by Business Processes or Workflow. Business Processes are attached to Parties, Client Connectors or Adapters in publish mode. Business processes assigned to a Party that is hosted by Neuron ESB will be hosted by an Endpoint host, as will a business process assigned to a client connector, an adapter, or a workflow endpoint. However, a business process assigned to a party that is hosted by a custom .NET application will be hosted by that .NET application.

Neuron ESB's configuration is stored in a Windows file folder as a series of XML files. The Neuron ESB Explorer is used to create and manipulate this configuration. An ESB configuration can be worked on in "Offline" mode. Online mode is used to monitor remote running Neuron ESB solutions. Offline mode can work with any ESB configuration, but any changes will not take effect until the Neuron ESB Windows Service has been configured to use that configuration and the service is restarted.

The Neuron ESB Test Client is a separate executable that is accessible either directly or by using the Neuron ESB Explorer Tools Menu. It is an excellent tool for testing baseline communication.

Quiz

1. True or false: Neuron ESB Explorer is a server application?
2. What activities require a Neuron ESB database to function?
    a. Workflows
    b. Auditing
    c. Activity Session Monitoring
    d. All of the Above
3. The term Neuron ESB uses for message pattern at the API level is
    a. Direction
    b. Semantic
    c. CorrelationExpectation
4. True or false: Neuron ESB can only pass XML data?
5. True or false: Processes can be attached to client connectors and adapter endpoints as well as parties?
6. In what mode do adapter endpoints need to be running in order to attach a business process?
    a. Publish
    b. Subscribe
7. Which of the following statements is false?
    a. Neuron ESB Processes are maintained on the server and sent in a serialized form to any Parties they are attached to.
    b. Neuron ESB Processes only execute on the Server.
    c. Neuron ESB Processes support nearly any processing imaginable.
    d. Parties in Winform apps running Processes can have those Processes utilize resources that will only exist on that client machine thereby enabling certain dynamic behaviors.
8. Workflows are attached to:
    a. Topics
    b. Parties
    c. Workflow Endpoints
9. A workflow can span:
    a. Less than a second
    b. Hours
    c. Days
    d. All of the above

Appendix

The following Artifacts accompany this training:

- Neuron Fundamentals Answers.docx – This document contains the answers for the questions in the Exercises section that do not involve creating esb files or Visual Studio projects
- Projects directory – This directory Contains the Following:

- Completed Neuron ESB Configuration NeuronFundamentals (all steps completed in this guide)
- NeuronAPITraining Visual Studio Solution
- Pub-Sub API Visual Studio Solution – a more complete example of using the Client API