

# Neuron ESB 3.0.3 Release! The Team that just keeps giving!

Just 2 months after we released our 3.0.2 version of Neuron ESB ([read about the 3.0.2 features](#)) ; we're following it up with a 3.0.3 update. This update not only fixes issues we've found or were reported, but we took the opportunity to include a number of new enhancements and features that customers asked for, and quite frankly, made a lot of sense to us. The following are just some of the things included in this release:

- SSL support for REST Service Endpoints
- Configuration encryption
- GZip and Raw Deflate Compression/Decompression Process Steps
- Dynamic support for Service and Xslt Parameters Process Steps
- Topics based on Rabbit MQ and Message Level TTL
- Parallel Process Step for Business Processes Designer
- SSL and Authentication support for SMTP Adapter
- JSON, polling and encoding support for Azure Service Bus Adapter

Along with new features, we've included a number of capability and user experience enhancements. Our goal is to always deliver a great experience to our customers when they develop solutions on the Neuron ESB platform.

All the changes included in the 3.0.3.128 update can be found in the [Neuron ESB Change Log](#) which gets installed with Neuron ESB. Users can download the latest Neuron ESB Build, which includes the 3.0.3.128 update, from the [Neuron ESB web site download page](#).

In this blog post, I thought I would elaborate on some (not all) of the new enhancements and features we've added to the update.

## Adapter Enhancements!

Adapters are key piece of capability in the world of an integration broker. They serve as the bridge to and from the bus between applications, databases and protocols. The completeness of "what" you ship as well as how easy it is for others to build their own adapters is critical in accelerating the development of any solution. Neuron ESB hands down has one of the easiest adapter frameworks to learn in the industry. However, we continue to deliver a more complete and richer set of adapters with every release. This update is no exception.

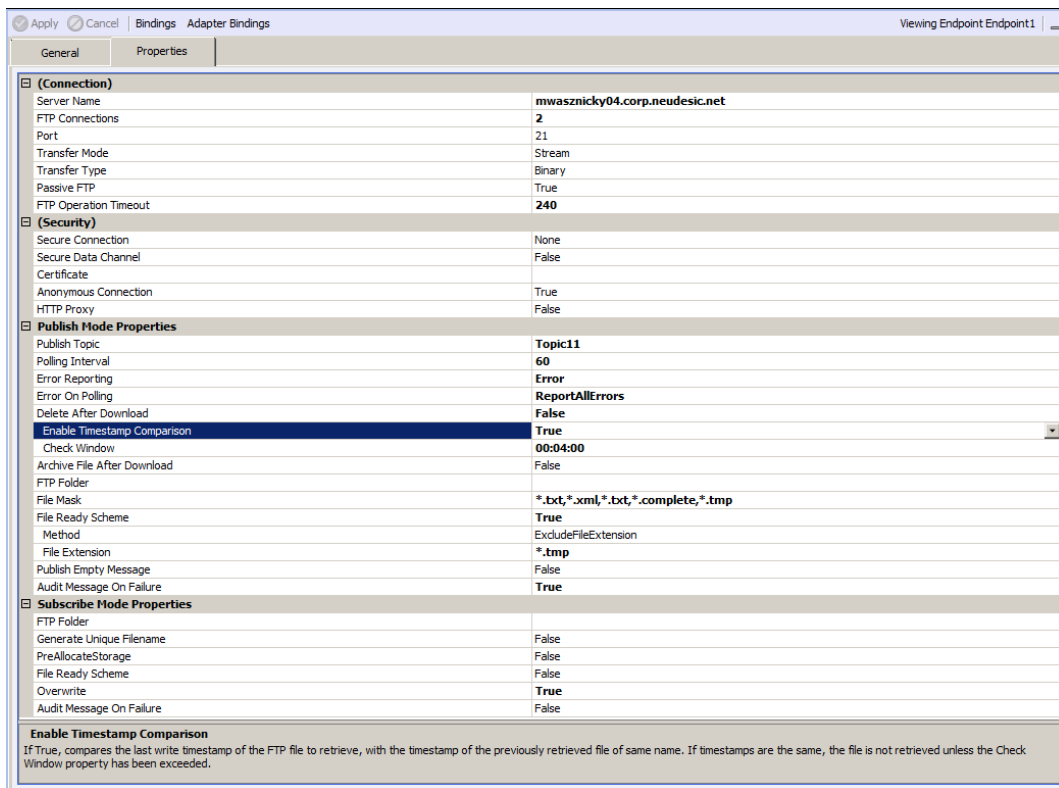
### FTP/SFTP Adapter Timestamp Comparison – new architecture

We initially shipped the FTP and SFTP adapters in Neuron 3.0 as well as 3.0.2. They shipped packed full of useful features that seem to be a "must" on most projects. However, one of our customers ran into a unique constraint while using the Timestamp Comparison feature. The Timestamp Comparison feature is used when organizations are not able to delete the files being retrieved, yet need to ensure they are

not downloaded twice. This could be due to permissions, but also due to the fact that other entities need to download the same files. This kind of capability needs to survive shut downs or server failures. Generally when this feature is enabled, a “Check Window” time limit is entered. For example, if the Check Window as 3 days, then once a file is retrieved it would not be retrieved again until either the Check Window expired or the Last Write timestamp on the file had been changed on the server, indicating a modification was done.

In the case of the customer in question, they expected upwards of over 100,000 files at any time existing in the FTP folders that Neuron would have to monitor with the Timestamp Comparison feature. Although the feature worked well with a dozen or even hundreds of files in the folder, it did not hold up when trying to execute the persistence and logic comparisons required to implement the feature.

In short, we rewrote the entire feature. Once the work was done, we tested it against FTP folders with over 150,000 files. Results were great! Memory consumption and process utilization remained low and steady. More importantly, performance increased by anywhere between 2 and 3X!



The picture above illustrates all the properties that can be configured for a Neuron ESB FTP adapter endpoint, depending on if the adapter is monitoring an FTP site and publishing files to the bus, or is receiving messages from the bus and uploading them to an FTP server. The Neuron ESB SFTP adapter has a similar set of features.

## Microsoft Azure Service Bus Adapter – Polling, JSON and Encoding support

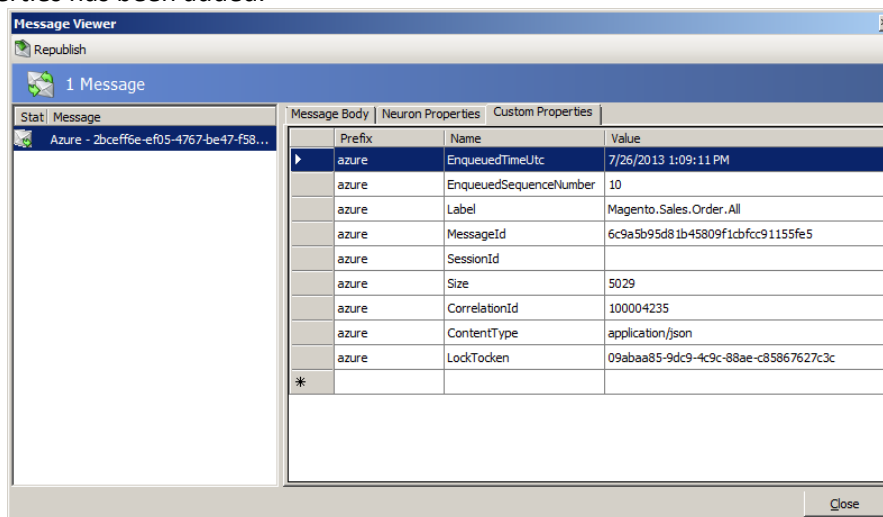
In this release we added a polling option in addition to the existing wait duration property for the Microsoft Azure Service Bus adapter that can be configured when receiving messages from Azure to publish to Neuron ESB.

The wait duration can be configured specifically to determine how much time the adapter should “wait” on an empty queue or topic before a message arrives. For example, if the wait duration was 10 seconds, the adapter would wait and listen on the configured Queue or Topic for either 10 seconds or until a message arrived to be received, whichever happens first. Consequently, if there were plenty of messages in the queue than the wait duration would be functionally ignored and the queue would be drained in retrievals that equaled the configured Batch Size.

With the introduction of the Polling option, the adapter can be configured to pause between the retrieval of messages.

Although all messages of any type would always be retrieved, we were not specifically determining the encoding of the message bodies retrieved from Azure. Unless the message body content type was text or xml, the body itself would be stored as a byte stream within the Neuron ESB Message. This prevented users from viewing and editing message types that should be human readable and editable, like JSON messages. Now, the encoding is determined on all messages and stored using that encoding.

Lastly, support for Azure custom brokered message properties as well as many of the Azure brokered message properties has been added.



Azure custom properties can be retrieved on incoming messages and can be added or modified and consequently applied to outgoing messages using the “azure.custom” message property prefix. For example, if a user wanted to either add a custom property to an azure outbound message the following could be run within a Code Step in a Business Process associated with the subscribing outbound Azure adapter:

```
Code Editor
void OnExecute (PipelineContext<Neuron.Esb.ESBMessage> context)
{
    1 context.Data.SetProperty("azure.custom", "myNewProp", "my crazy value");
}
Compile Errors
Save Cancel
```

That property would be propagated to the Azure Brokered Message and would be visible within the Azure Service Bus management portal.

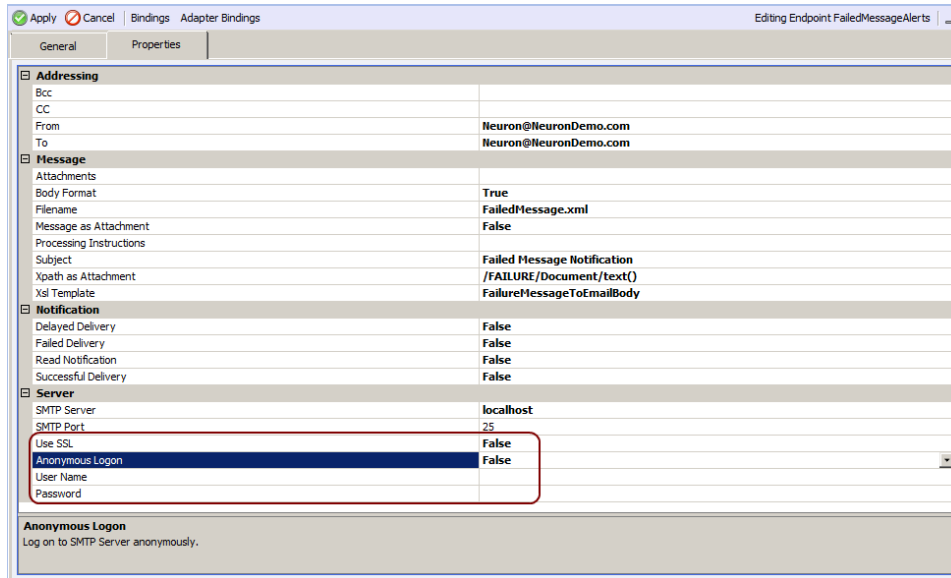
Many of the standard Azure Brokered Message header properties are also supported, some of which can be modified or applied to outgoing messages as well. Specifically:

- *CorrelationId*
- *MessageId*
- *ContentType*
- *Label*

These properties are associated with the “azure” message property prefix (rather than “azure.custom”).

### **SMTP Adapter now supports authentication and SSL**

Interestingly enough, Neuron has always shipped with an SMTP adapter for a while. However by default, it only supported anonymous login to SMTP servers. In this release, we finally added support for both SSL and user supplied credentials. The SMTP adapter itself is fairly full featured. It supports various message body and xslt options that can be directly configured in the property grid as shown in the screen shot below:



## Dynamics CRM Adapter – Customer ID lookup support

The Neuron ESB Dynamics CRM Subscription adapter is useful for updating entity fields within Dynamics CRM. However, there was a case where Lookups for Customer ID type fields would not work. For example, if a user wanted to save a contact and associate that contact with a specific account dynamically, it would fail. Now this type of use case will be successful. Here's an example of a Save command executed through the adapter:

```
<NeuronCrmAdapterMessage>
  <CrmCommands Transactional="false">
    <Lookups>
      <Lookup id="customerid" entity="account" lookupfield="accountid">
        <searchfield name="accountnumber" value="NEUCALUS01" operator="equal" />
      </Lookup>
    </Lookups>
    <SaveCrmRecord allowCreate="true" id="contact1" entity="contact" identityfield="contactid">
      <property name="emailaddress1" value="first.last@domain.com" typename="String" />
      <property name="birthdate" value="04/11/1982 12:00:00 AM" typename="CrmDateTime" />
      <property name="firstname" value="Jane" typename="String" />
      <property name="lastname" value="Smith" typename="String" />
      <property name="donotemail" value="false" typename="CrmBoolean" />
      <property name="parentcustomerid" lookupid="customerid" reference="account" typename="Customer" />
      <property name="creditlimit" value="10000" typename="CrmMoney" />
      <property name="department" value="Neudesic Sales" typename="String" />
    </SaveCrmRecord>
  </CrmCommands>
</NeuronCrmAdapterMessage>
```

## Business Processes, the glue that binds...

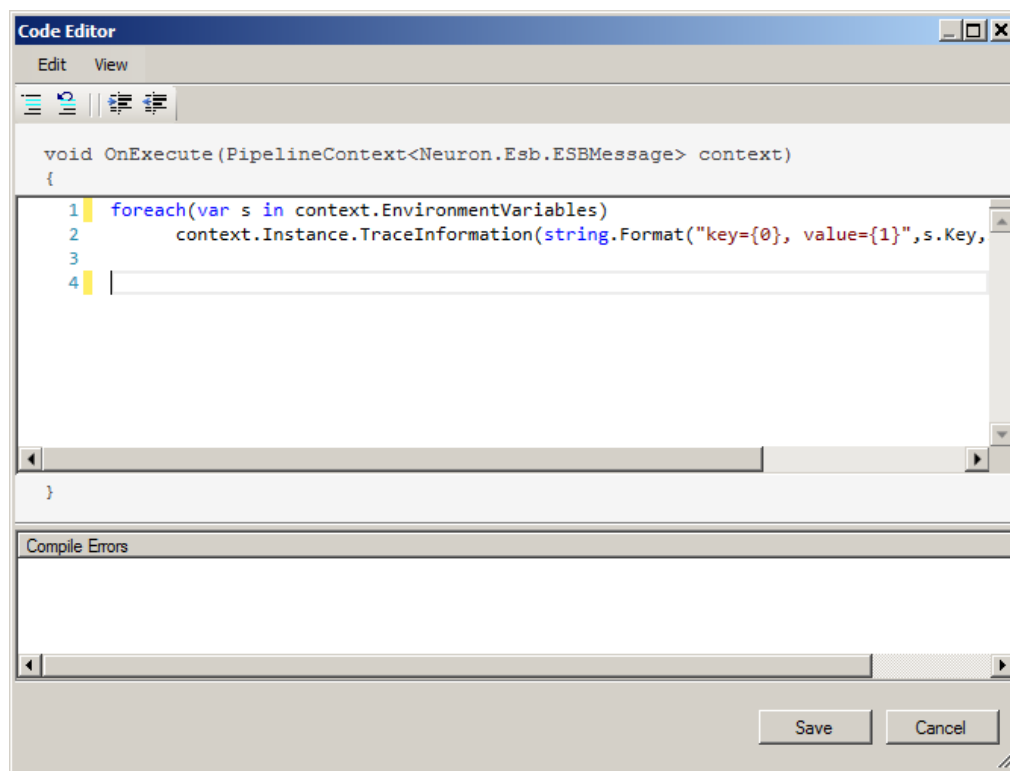
I think we delivered some pretty good things to make building and testing business processes easier within the Neuron ESB Explorer. We always try to deliver “productivity” out of the box. If we can find an opportunity to reduce the amount of code someone has to write in our designer, we’ll try to take the opportunity to do so.

## Code Process Step – Lets add some useful functions ☺

The Code Process Step has to be one of the most useful process steps in the designer. With it, the developer can do nearly anything they want with the Microsoft .NET C# language. It’s a full code editor with all the .NET 4 language enhancements and intellisense. Users can turn off or on line numbering,

add references to existing assemblies or inherit referenced assemblies applied at the process level, zoom in and out, etc. However, we're always trying to make this more useful so that developers can be more productive.

In this release, we've added a toolbar that allows users to do some fairly common things when writing code, specifically comment or uncomment as well as indent/unindent the lines of code they're working with.



Although this seems minor, we're far from done. In future releases expect more enhancements. Some of the things we're looking at doing are:

- Adding namespace support – relieving the need to fully qualify your objects
- Adding a debugger – so break points can be set and developers can walk through a process, step by step as well as through the code within a Code step
- Jscript support
- Formatting support
- Code Snippets
- Modeless Code Editor window – allowing multiple code editors to be opened at once

So stayed tuned!

### **SQL based Process Steps – Command and Connection timeouts**

We've finally added the basics i.e. Command and Connection timeouts that users can configure on the Table Query, Xml Query and Store Process Steps. Generally, we've never had a problem with the

defaults, but there are those situations which we've encountered where users simply need the ability to modify these. Something long overdue 😊. The upside, the ODBC process step already had these properties built into it!

### New JSON Process Step!

Although Neuron already shipped with the Newtonsoft JSON.NET libraries, it required Neuron ESB developers to learn the API to use it. With this release we're introducing a specific JSON Process Step that should make JSON serialization/deserialization a walk thru the park for most users.

The JSON Process step supports the following conversions:

- JSON to XML
- XML to JSON
- .NET Class to JSON
- JSON to .NET Class
- XML to .NET Class
- .NET Class to XML

Using the JSON Process step is fairly straight forward. It will auto detect the incoming message format i.e. if it's XML, binary or text (JSON). The user then configures the following properties of the JSON process step within its respective property grid:

Property Name	Description
Convert To	Either XML, JSON or Binary can be selected
Class Type	The fully qualified name of a .NET class to serialize/deserialize to. The assembly must be in the GAC or the Neuron installation folder. There is an assembly and class picker User Interface that allows users to browse for the assembly and select the class. The Class selected must be serializable. This is required when converting from Binary or to Binary.
DateTime Format	This property is visible ONLY when Convert To is set to JSON. Optional. It allows users to specify a custom date time format string, rather than use the default ISO 8601 format.
Include Object Type Names	This property is visible ONLY when Convert To is set to JSON. Optional. If True, include the .NET type name when serializing into a JSON object structure
XML Root Name	This property is visible ONLY when Convert To is set to XML. Optional. Xml Root name to apply if converting JSON (without a root object) to XML. If a root already exists, this root name will be removed from final XML
Include Xml Declaration	This property is visible ONLY when Convert To is set to XML. Optional. When converting to XML, this will determine if the XML Declaration is written.

For example, using this process step I can easily convert the following JSON :

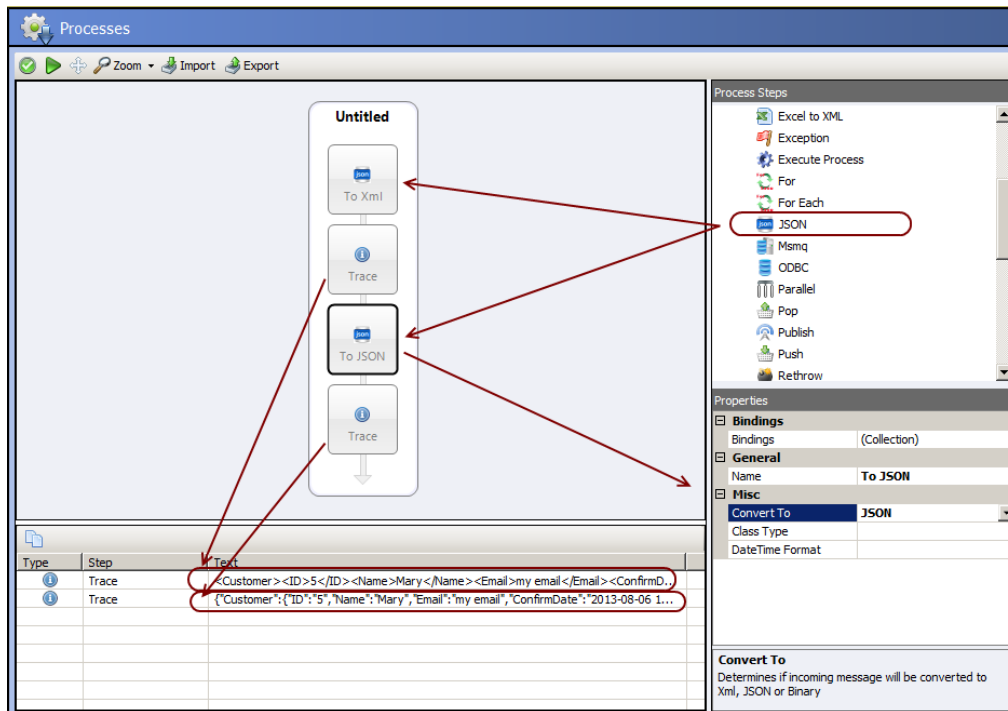
```
{"Customer":{"ID":"5","Name":"Mary","Email":"my email","ConfirmDate":"2013-08-06 10:04:42"}}
```

To Xml:

```
<Customer>
  <ID>5</ID>
  <Name>Mary</Name>
  <Email>my email</Email>
```

```
<ConfirmDate>2013-08-06 10:04:42</ConfirmDate>  
</Customer>
```

And back again. Setting the properties for this is relatively simple as shown in the picture below:

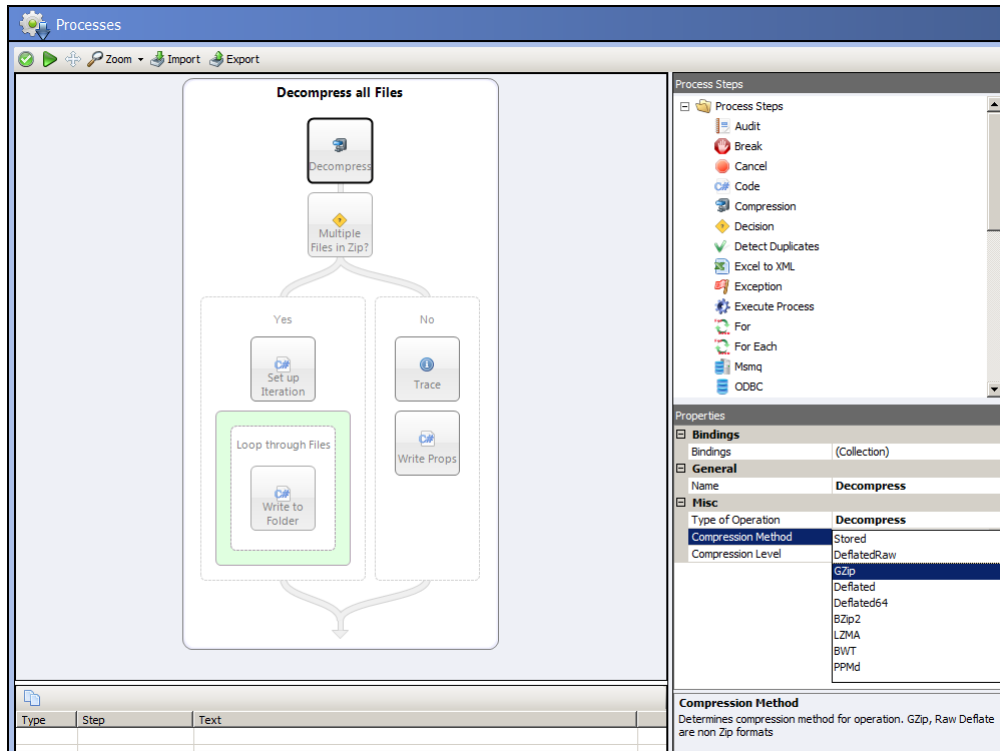


### Zip/Unzip Process Step – Compression with GZIP!

In 3.0.2 we introduced the Zip/Unzip Process Step. A nice easy way to compress and decompress data on the fly! However, we found some customers wanted GZip support, while others required RAW Deflate support.

In 3.0.3 we decided to just rename Process Step to “Compression” and add RAW Deflate (which can be used with MSFT’s library or PHP’s zlib) and GZip as supported compression methods. Previously, we never exposed the compression methods as a configurable property in 3.0.2. However, since we’re introducing GZip and RAW Deflate, we now make all the compression methods configurable, as well as the level of compression (normal, low, high). All other selectable compression methods are ZIP based.





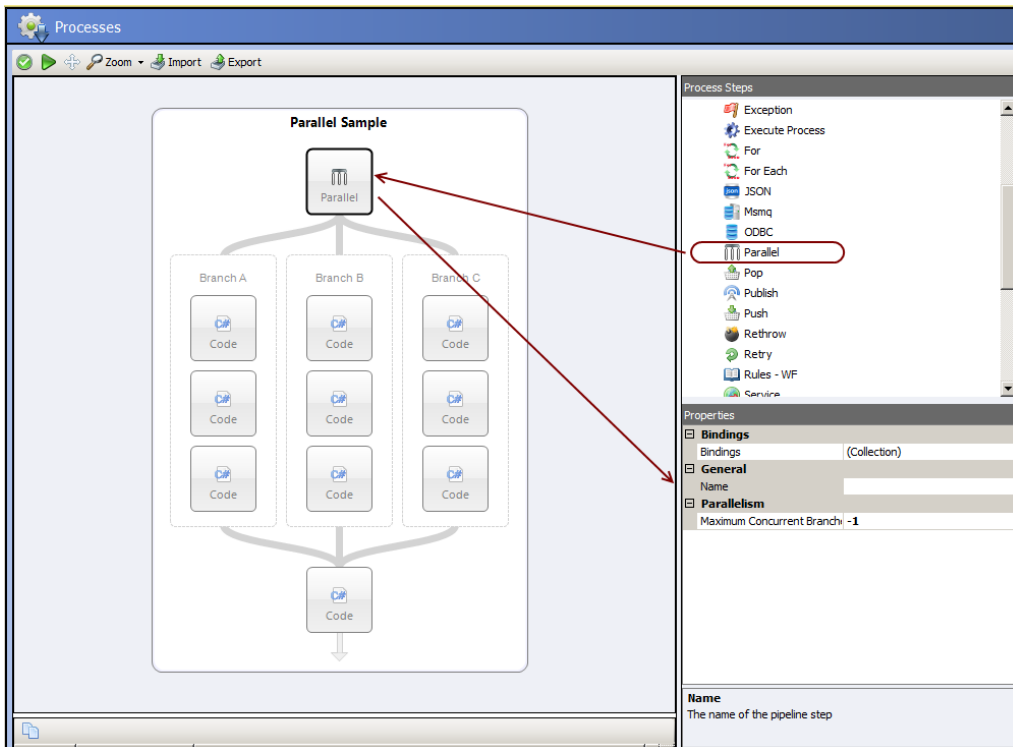
### Service Process Step – Dynamic Service Url Support

NEW – Service Process Step – Users can now dynamically configure the service URL at runtime by setting the context property i.e. `context.Data.SetProperty("Addressing", "To", http://localhost/someService ).`

I think the “lack” of this functionality definitely was a limiting factor for adoption. Now the Service Step can be used directly in scatter gather patterns where generally the URL and Action are changed per back end service calls. The Action property could always be dynamically set ....so a nice compliment!

### New Parallel Processing Step!

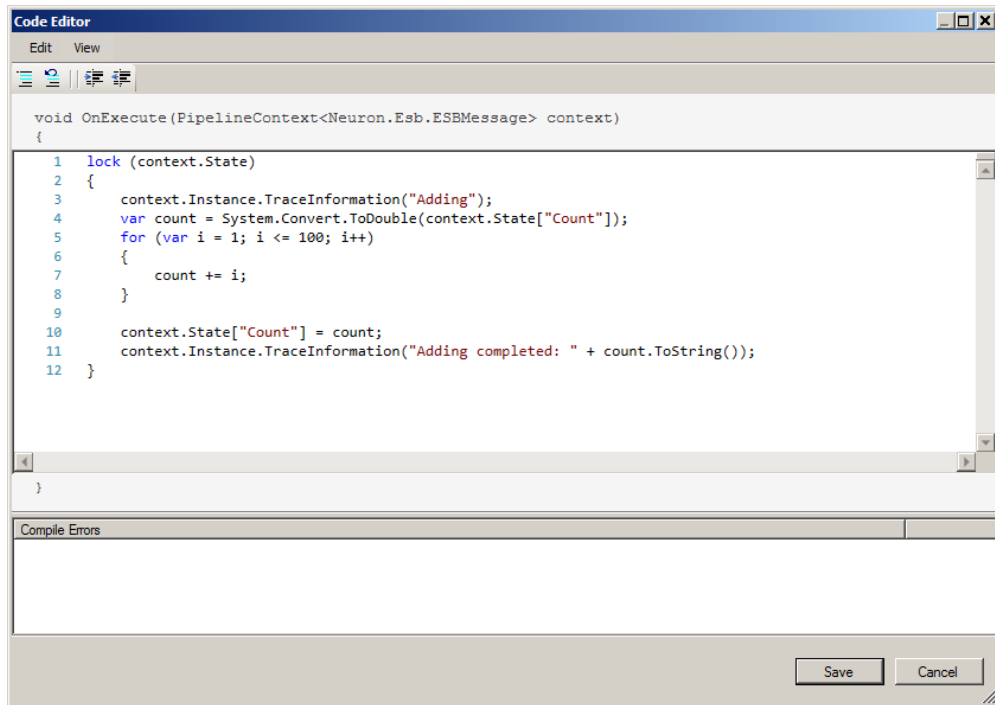
What can I say? I’m pretty excited about the inclusion of the new Parallel Processing Process Step in this release. Although Neuron always had some unattractive hacks for doing it, this new Process Step makes parallel processing incredible simple! This new step allows actions to be executed in Parallel within a Business Process. Multiple parallel branches can be added; each branch by default will run in its own thread (unlike some other products out there☺). The total number of concurrent threads (i.e. branches) is configurable within the property grid.



To make this even more useful, a new State object, off of the context object has been added to manage concurrent access to state across all running instances of the same business process and within each branch of a parallel step. This State object is thread safe and can be accessed and updated concurrently between threads. Setting the property is as easy as:

```
context.State["Name"] = "Joe";
```

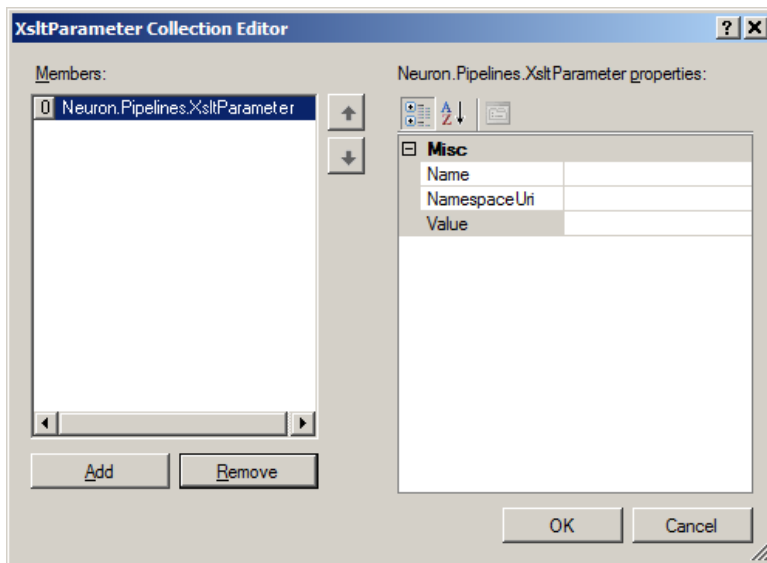
Alternatively a lock can be used to control longer running access against a property in the State object as shown below:



```
void OnExecute(PipelineContext<Neuron.Esb.ESBMessage> context)
{
    1 lock (context.State)
    2 {
    3     context.Instance.TraceInformation("Adding");
    4     var count = System.Convert.ToDouble(context.State["Count"]);
    5     for (var i = 1; i <= 100; i++)
    6     {
    7         count += i;
    8     }
    9
    10    context.State["Count"] = count;
    11    context.Instance.TraceInformation("Adding completed: " + count.ToString());
    12 }
}
```

### Xslt Transform and Dynamic Parameters

Here's another great feature a long time coming. Being able to dynamically set the values of Xslt Parameters at runtime! Yes, this can now be easily done by using custom message properties, environment variables, or XPath expressions in in the parameters collection for a configured Xslt Transform Process Step:



To use a custom property, enter the following into the value field for the parameter:

`{property:<prefix>.<name>}`

For example, if I have defined the custom property “Person.Name” with value “Michael Jones”, then using “{property:Person.Name}” will insert the value “Michael” into the parameter value when the XSLT is evaluated.

For an environment variable:

```
{env:<name>}
```

If I have the environment variable “MachineName” set to “MJONES01”, then using “{env:MachineName}” will result in the parameter having the value “MJONES01”.

Finally, for an XPath expression:

```
{xpath:<xpath-expression>}
```

If my source XML looks like this:

```
<person>
  <firstname>Michael</firstname>
  <lastname>Jones</lastname>
</person>
```

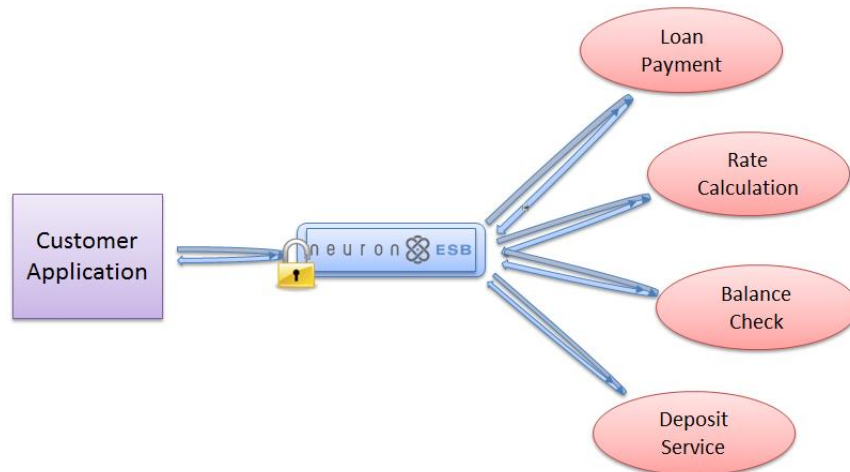
Using “{xpath:/person/firstname}” will use the value “Michael” for the parameter value.

## Service Broker for the Microsoft Platform

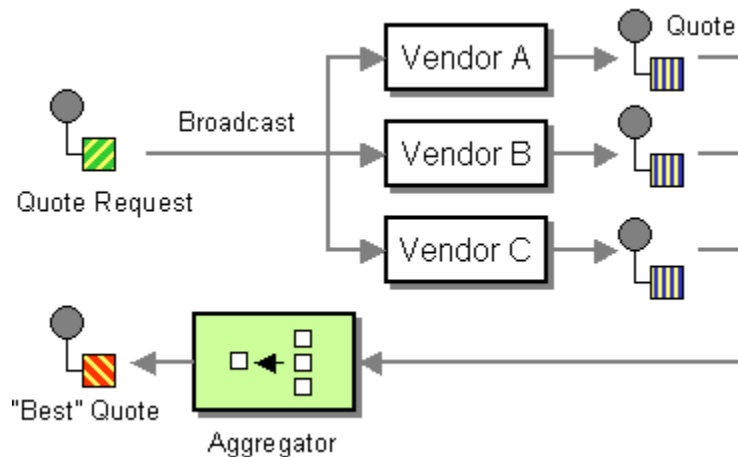
Neuron ESB exposes a number of rich capabilities including Business Process Design, Enterprise Application Integration. In conjunction with providing a powerful integration platform for the .NET Developer, Neuron ESB 3.0 provides a web service platform which facilitates critical functions for organizations interested in successfully adopting a SOA strategy:

- Service Intermediary
- Service Router
- Service Versioning
- Service Provider
- Service Patterns

By serving as a broker, Neuron ESB can abstract, process, mediate, and dynamically route incoming service requests to existing services throughout an organization. Neuron ESB can shield customer facing applications from changes that occur in an organization’s infrastructure, applications or services.



Neuron ESB provides a powerful runtime environment for hosting either SOAP or REST based services as well as a real-time Business Process runtime for developing complex service pattern such as Scatter-Gather as depicted below:



Neuron ESB 3.0 ships with a sample that demonstrates in detail the Scatter Gather Pattern and how to implement it within Neuron ESB. This pattern in fact can now be greatly improved using the new Split Process step couple with the updated Service Process Step!

However, we're constantly trying to improve and enhance what we do, and how we do it. The Neuron 3.0.3 release is no exception.

### SSL Support for REST endpoints and other goodies

The Neuron ESB 3.0.3 finally includes SSL support for REST full endpoints. This means when you use the REST binding on either a client or service connector, SSL certificates can be selected and the binding's security properties can be properly set!

We also included augmented support for handling REST errors returned to the client. When a service endpoint returned a 4xx Status Code, Neuron would return a 500 status code to the client and wrap the actual status code and description in headers, for example:

```
HTTP/1.1 500 The remote server reported an error.  
Content-Length: 0  
Server: Microsoft-HTTPAPI/2.0  
StatusCode: 401  
StatusDescription: Authentication failed. Invalid token.  
Date: Fri, 26 Jul 2013 13:01:58 GMT
```

In the Neuron 3.0.3 release the actual status code and description flow through unwrapped.

Lastly, when using a REST service connector, it looks like encoded values were being re-encoded, resulting in incorrect URLs being sent to OData web services. Users will find that this works now!

## Topic based Publish/Subscribe

We made some great modifications to our Topic Transport list. Neuron ESB provides a hierarchical, Topic-based publish and subscribe model to mediate the routing of messages between Parties (Publishers and Subscribers), Adapters, and Service Endpoints. Neuron's Topic model is composed of a sub topic hierarchy that can more intuitively reflect either an organization's structure or business requirements. However, Neuron ESB is unique in the industry in that it allows the business to determine the Quality of Service (QoS) attributes at the Topic level, and that many Topics of various QoS attributes can exist side by side. By providing this level of flexibility, organizations do not have to worry about changing their specific business requirements to meet the limitations imposed by other competing products. Some of the critical QoS attributes include:

- Throttling
- Encryption
- Auditing
- Transport
- Transactions
- Durability
- Compression

Transport is a critical QoS selection since it affects many aspects that businesses may require including transaction and durability support as well as ordered messaging, guaranteed delivery, once only delivery, scale out, latency and performance. In previous versions of Neuron ESB, several configurable Transports for Topics were supported including:

- TCP
- MSMQ
- PEER
- BIZTALK
- NAMED PIPES
- AMQP

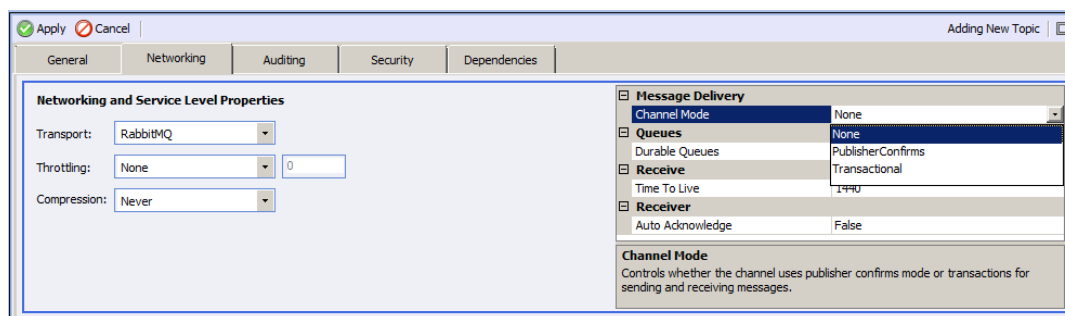
With the Release of Neuron 3.0.3, we made some modifications to the list and the level of control we expose to our users!

## AMQP renamed to Rabbit MQ and updated!

In this release, the most noticeable change users will see is that the AMQP transport is renamed to RabbitMQ. We decided to rename this because we had to build specific support for Rabbit MQ to provide the monitoring and management experience that customers expect. It just didn't make sense to keep the old name. However, that's not all we did.

First, we upgraded out support to Rabbit MQ version 3.1.3, from 2.8.5. This upgrade allowed us to provide a few useful and necessary features that were missing from 2.8.5. For example, configuration support has been added to provide Message Level Time to Live (TTL) support. With this, comes our implementation of automatic Dead Letter Queue handling, similar to what we offer today with the MSMQ transport. What does this mean? It means users can configure when a message will expire once it's published to the bus. If the message is not picked up by a subscriber in the set amount of time, it will automatically be moved to the Neuron ESB Audit Failed Message table. Slick!

We also took the liberty to add configuration options so that users can choose between RabbitMQ's transaction and publisher confirms reliability options at the Topic level (Channel Mode property). This also allowed us to move routing for Topics and Sub Topics to the Exchange level.



I guess a common question will be why would anyone use Neuron ESB's Rabbit MQ transport over MSMQ? I listed below some of the features of each:

### Rabbit MQ

- The underlying Neuron ESB infrastructure queues to support the Topic/Parties are automatically created, regardless of where the Rabbit MQ server is installed. Very nice feature!
- If using Rabbit MQ based Topics, Rabbit MQ is NOT required on remote machines hosting the Neuron ESB Client API. Another nice feature!
- Rabbit MQ does not support System.Transactions or the MSDTC. As far as Topics go, this could be a blessing in disguise 😊. Rabbit MQ has its own implementation of Transactions which is supported through Neuron ESB
- Rabbit MQ does not require Windows Clustering. Instead Queues are "Mirrored" to another Rabbit MQ server. This means it's much easier to setup High Availability.
- Supports Ordered Message Delivery
- Neuron ESB Provides support for viewing message statistics for the underlying Neuron ESB infrastructure queues for Topics/Parties. Users can also delete existing pending messages. However, users can not view/edit or redirect these pending messages (unlike MSMQ)

## MSMQ

- The underlying Neuron ESB infrastructure queues to support the Topic/Parties must be manually created, using either the scripts or commands provided by the Neuron ESB Explorer
- If using MSMQ based Topics, MSMQ IS required on remote machines hosting the Neuron ESB Client API
- For High Availability, MSMQ requires Windows Clustering
- Supports Ordered Message Delivery
- MSMQ supports MSDTC type transactions. This can cause complexity when configuring Neuron ESB Policies for retries
- Neuron ESB Provides full support for managing, viewing, editing, deleting and redirecting pending messages that exist in the underlying Neuron ESB infrastructure queues for Topics/Parties

I think there are going to be many scenarios where MSMQ is used today that Rabbit MQ will end up being a better choice for. Moving forward, our goal will be to build out even more message management support for Rabbit MQ.

## BizTalk Transport dropped

Well, I had mixed feelings about this. I originally worked at Microsoft for 6 years on the BizTalk team. I personally thought the BizTalk channel we developed was slick. Once installed on the BizTalk Server farm, users of Neuron wouldn't have to know anything about BizTalk at all. But at the end of the day, no one ever used it. It certainly contrasted with the rest of the Transports we offered when looking at performance and function. While the other transports we offered performed very well, BizTalk...well, it's BizTalk. Which means it could never perform as well as the others or even come close. Just way too many constraints and bottlenecks built into the BizTalk architecture. Also, the added infrastructure requirements just didn't make it practical as either a viable transport option or as an interop option. If users need to interop Neuron with an existing BizTalk application it was just as easy to it with our existing Service Broker or Adapters.

Removing this should also end any confusion; Neuron ESB has NEVER been built on, OR dependent on BizTalk Server for any of its capabilities!

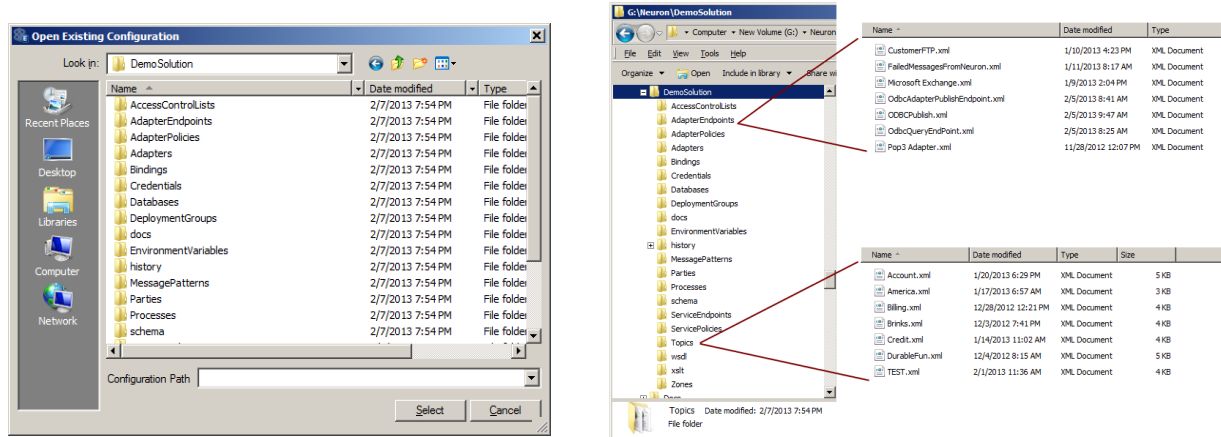
Goodbye BizTalk Server, may you rest in peace ☺

## Neuron ESB Configuration Encryption

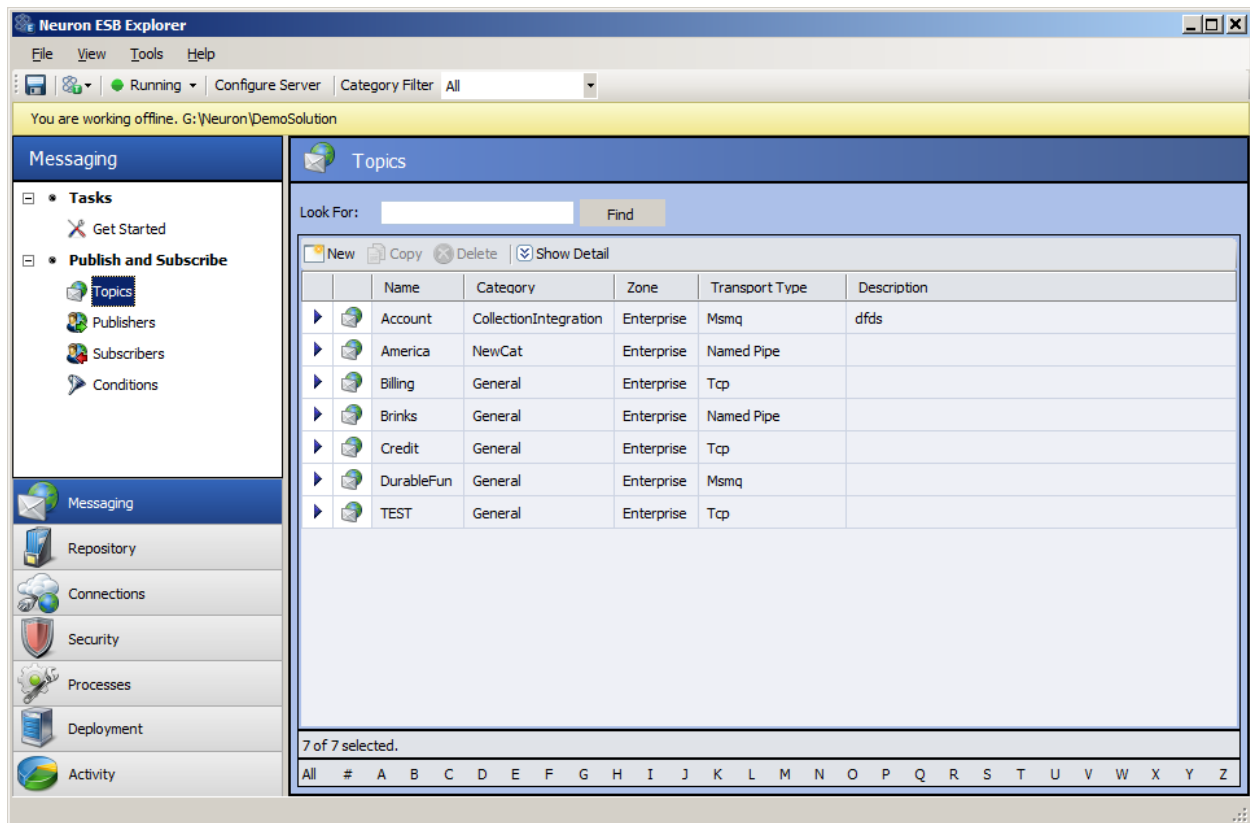
The Neuron ESB 3.0 configuration storage format has evolved from the single file format (i.e. \*.ESB file available in Neuron ESB 2.x) to a directory structure consisting of folders representing each entity type. Within each folder, an XML formatted file represents a specific entity such as a Topic, Endpoint, Business Process, etc. The Neuron ESB Explorer is designed to work and store all modifications to the directory structure.



After launching the Neuron ESB Explorer, users can Connect, Open, or Create a new Neuron ESB 3.0 configuration. When Open is selected, users are prompted to select the root folder of an existing Neuron ESB 3.0 configuration directory as depicted below:



Once the Neuron ESB 3.0 configuration is opened, each XML file within the entity folder is loaded and managed within the Neuron ESB Explorer. There is a one-to-one mapping of XML file to managed entity. The figure below displays the Topics represented by XML files within the Topics sub folder of the DemoSolution directory (pictured above).



In Neuron ESB configurations we were diligent to always encrypt/decrypt the passwords for Credentials or Access Control Lists that were configured in the Security section of the Neuron ESB Explorer. This ensured that those passwords involved were never stored in plain text within the entity's related XML file. However, in the Neuron ESB 3.0.3 release we've modified this to be far more inclusive and complete.

In the Neuron 3.0.3 we're not just encrypting passwords, but other properties on adapters as well as associated entities. Here's a table that shows the differences between Neuron 3.0 and 3.0.3. Everything in in Bold/Italics/Blue is new to 3.0.3

Entity	Encryption Level
Administrators	The entire object is encrypted
Credentials	Only passwords are encrypted
Access Control Lists	The entire object is encrypted
Service Endpoints	Only passwords are encrypted
Subscribers (if accounts are used to restrict)	The entire Account object is encrypted
<b>Adapter Endpoints</b>	<b><i>All properties that end with "Password", "Key", "Passphrase" or "ConnectionString" are encrypted</i></b>
<b>Databases</b>	<b><i>The Connection String is encrypted</i></b>
<b>Business Process Steps</b>	<b><i>All properties that end with "Password", "Key", "Passphrase" or "ConnectionString" are encrypted</i></b>

This means that any custom Process Step or Adapter that users develop that expose any property (regardless of case) that end with the "ConnectionString", "Key", "Passphrase" or "Password" keywords will be automatically stored as encrypted data within their respective XML files.

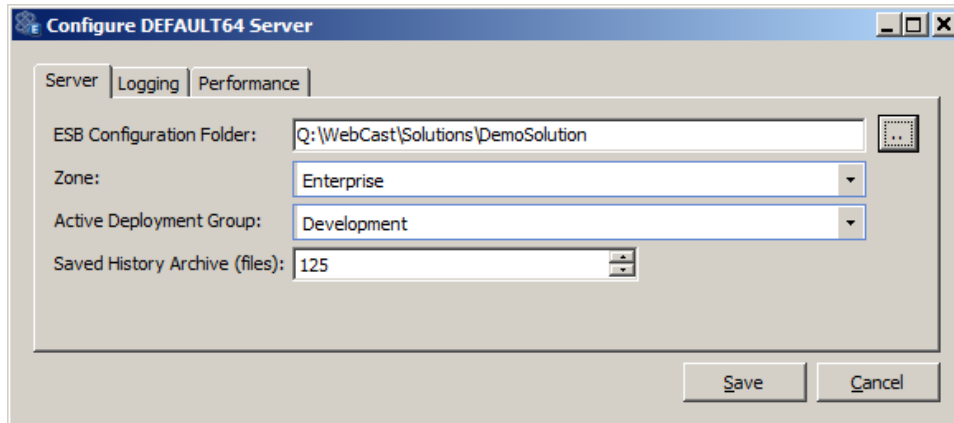
The Neuron ESB Explorer as well as the Neuron ESB Runtime is responsible for encrypting/decrypting the information at either design time or runtime.

## Neuron ESB Explorer - enhancing the User Experience!

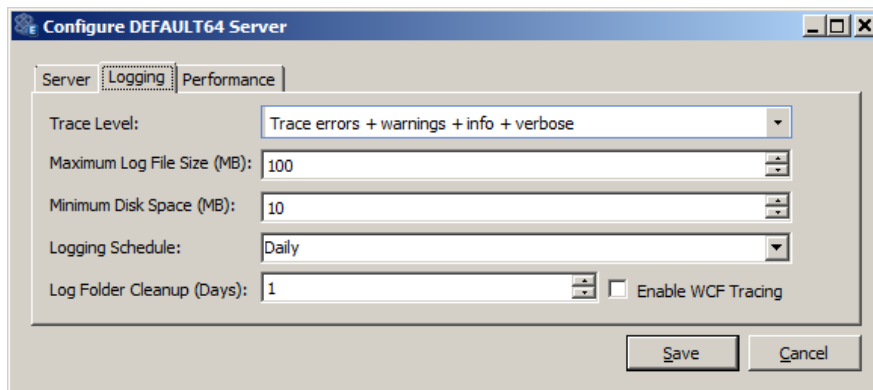
Considering the Neuron ESB Explorer is the primary User Experience for most people, we make a concerted effort to improve the user experience when the opportunity presents itself. This time around we made some minor changes; some to help prevent users from making mistakes; others to help with everyday house cleaning.

For example, we've added support for configuring automatic deletion of older Neuron ESB Saved Configuration archive files within the History folder of Neuron ESB Solutions. During development, every time a configuration is saved, Neuron automatically backs up the existing configuration in the History folder before committing the saved changes. This allows users to roll back their changes to any point in time. A pretty handy feature! However, hit "Save" frequently and before you know you it, there could be hundreds or more backed up configurations in this History folder.

Hence, the "Saved History Archive (files)" setting, defaulting to 0 (unlimited number of archived files), has been added to the Configure Server dialog which can be accessed either from the Neuron ESB Explorer toolbar or from the Neuron ESB Explorer Server management screen (i.e. Deployment -> Manage -> Servers, right click on service and select "Configure Service..." from the context menu).

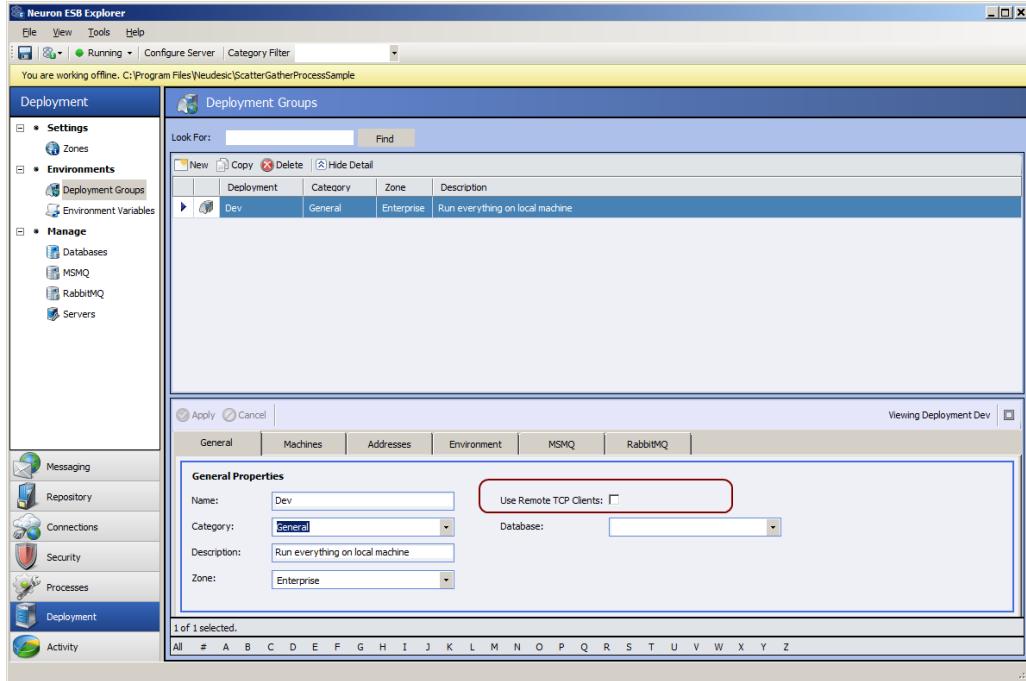


A similar setting was also added for configuring automatic deletion of older Neuron Log directories within Neuron ESB Explorer. Every time the Neuron ESB Service is restarted, a new Log Folder is created. If during development there are a lot of restarts, over time this could result in a lot of folders. If verbose logging is enabled (typical for development environments) this could also mean that there could be a considerable amount of disk space being used unnecessarily. Hence, the “Log Folder Cleanup (Days)” setting was added, defaulting to 10 days, to the Configure Server dialog which can be accessed either from the Neuron ESB Explorer toolbar or from the Neuron ESB Explorer Server management screen (i.e. Deployment -> Manage -> Servers, right click on service and select “Configure Service...” from the context menu).



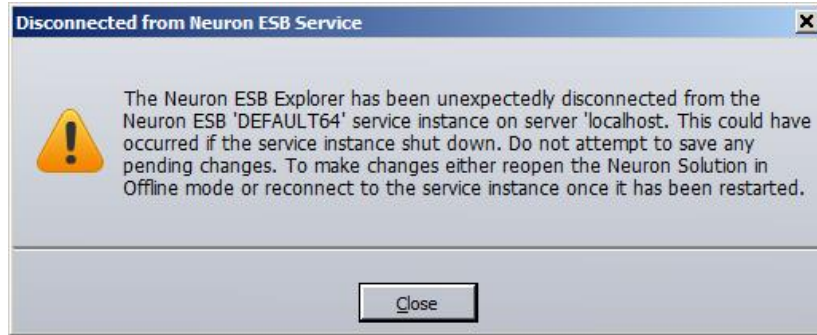
A point of user confusion we really wanted to address (personal pet peeve of Joe’s) was the “Server Topology” dropdown box located on the General tab of the Deployment Group. This could be configured for either “Server Farm” or “Single” mode...both options fairly confusing for most. In fact, this setting really didn’t have anything to do with failover or load balancing. What it intended to do is allow servers to know about the remote parties attached to any Neuron server in a group and, if a message came in, to ensure the remote attached party got the message, regardless of what server they were initially connected to. In short, this setting IS ONLY relevant if the Neuron ESB Client API is used, hosted on machines remotely located from the Neuron ESB Server AND those clients are ONLY publishing or subscribing to TCP based Topics.

Due to this we saw fit to change the “Server Topology” drop down box to a check box and relabeled to “Use Remote TCP Clients” to more accurately represent its function. This only needs to be enabled if using the Neuron ESB Client hosted in remote .NET applications which communicate via TCP based Topics. The database must also be selected.

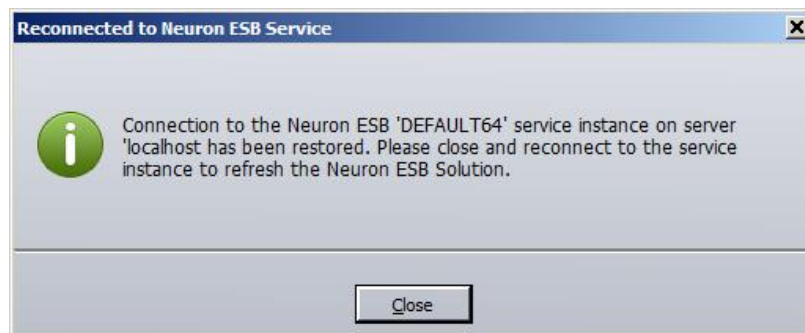


Lastly, when using the Neuron ESB Explorer users have the ability to work with a solution either in Online mode or Offline mode. When in Online mode (connected directly to the local or remote runtime, which the solution is being pulled from) users can make changes directly to the solution running in memory and changes are immediate. This is very handy when doing development since you can see immediate results. When in offline mode, changes made are written directly to the folder directory of the solution. Both are great options. However, if the Neuron ESB runtime that the Neuron ESB Explorer is connected to in Online mode is shut down either deliberately or unexpectedly, the user would not know. There was no notification wired in to alert the connected user. If the user continued to work and then tried to save the changes made, an error would be generated.

In the 3.0.3 release, we built in notifications for the Neuron ESB Explorer. Now, IF in online mode, users will be notified if the ESB Service runtime they are connected to goes offline. The Neuron ESB Explorer status bar (under toolbar) will flip from green to yellow, display a text warning and the user will be presented with a dialog similar to below.



When the service comes back Online, the user will also be notified as well. The Neuron ESB Explorer status bar (under toolbar) will flip from yellow back to green, display a text and the user will be presented with a dialog similar to below.



If ESB Service runtime shuts down while in online mode while there are pending changes to be saved, users will be prompted to continue or cancel so that changes can be saved first. If users try to save when in Online mode, but they are currently in a disconnected state, they'll be warned about the current condition.

Well that's all for now. There are more fixes and changes that you can read about in the Neuron ESB Change Log. Looking forward to posting more information about our product in the future. Stay tuned!