Neuron ESB 3.5 introduces Long Running Workflow capabilities!

Neuron ESB 3.5 introduces several new features and enhancements, most significant of which is long running, fault tolerant Workflow. Neuron ESB 3.5 now ships with Workflow capabilities that allow companies to design fault tolerant, business resilient workflows to automate critical processes that may span hours, days, weeks or months and cross inter- or intra-company domains. Neuron ESB's Workflow offering is built upon Microsoft .NET Workflow Foundation 4.5, overlaying it with tools, infrastructure, a hosting environment and services necessary to deliver enterprise-level performance and scalability on the Microsoft .NET platform.

Previous versions of Neuron ESB offered business process capabilities through a graphical, user-friendly Business Process designer and runtime environment. The Business Process engine targeted real-time requirements where performance, agility and time to market were driving factors. This was often used in low latency environments such as request/response type of messaging to provide either simple VETO or, more complex Scatter–Gather and Service Composition/Orchestration Patterns. Service Composition and Orchestration is commonly used to expose a discrete set of services within an organization as higher level business services.

For example, a Business Process used to execute purchase orders may "orchestrate" the execution of several existing services in the organization and/or cloud to retrieve the information needed or update necessary systems. The results of which may need to be evaluated, enriched and/or aggregated and returned as the final response. Some of these activities may be executed asynchronously or even in parallel. Collectively, these activities and services represent a higher level Order Processing Service, where innovation is created by "composing" existing services into new business capabilities.

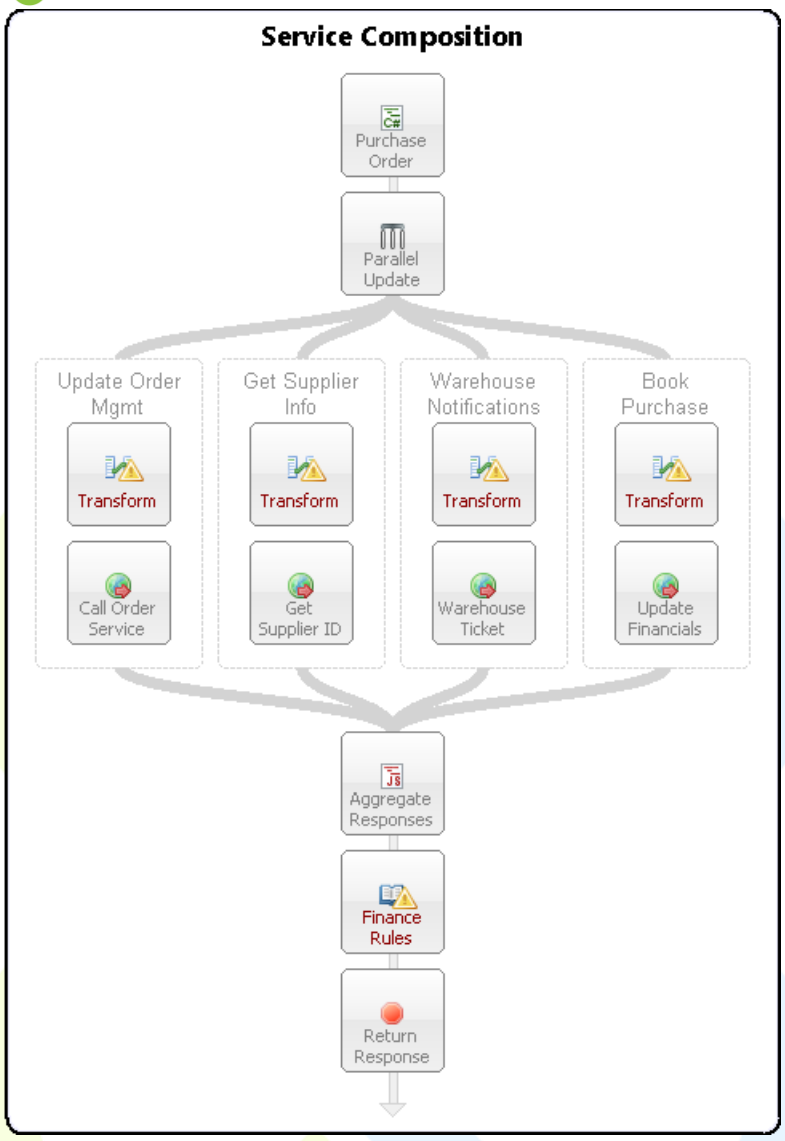

Figure 1 Neuron ESB Service Composition/Orchestration – Example of a Service Composition Business Process within the Neuron ESB Explorer

In addition to Service Composition/Orchestration, many organizations use the Neuron ESB Business Process engine to build fairly complex business processing scenarios. However, where the Business Process engine excelled in the areas of performance, functionality and ease of use, it lacked certain features such as real time activity tracking, fault tolerance, correlation of long running messages as well as “out of the box” compensation (commonly referred to as “saga” or “long running transactions”).

Neuron ESB 3.5 Workflow adds all these features and more, allowing businesses to automate and manage processes that span cloud, partner, system and organizational boundaries. When critical failures occur in the process or the underlying hardware, workflows can resume where they left off in the Neuron ESB hosting environment. Neuron ESB 3.5 provides a clustered hosting environment (called “Availability Groups”), that load balances the execution of workflows across multiple servers in dedicated/isolated host processes. This same clustered hosting environment allows failed workflows to

automatically rollover onto available servers and start where they left off, providing both resiliency and reliability for mission critical functions.

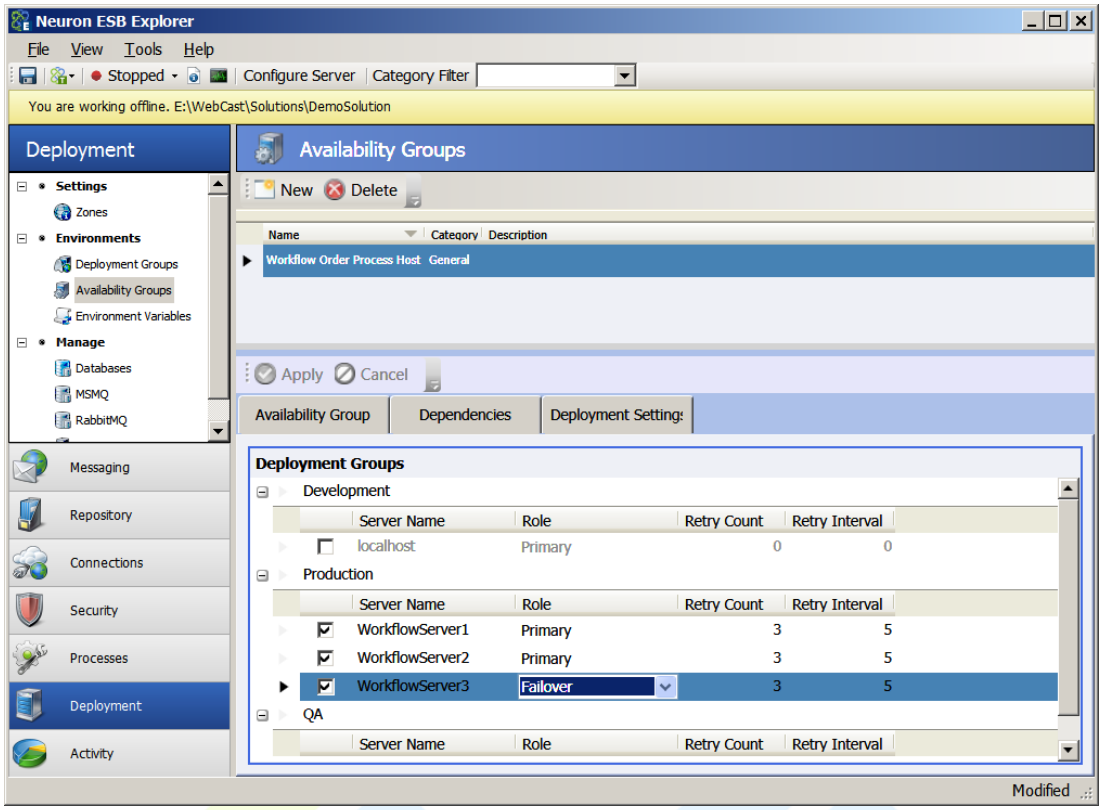

Figure 2 Neuron ESB Availability Groups – Availability Groups provide isolated high availability clustered hosting for Neuron ESB Workflows. Servers can be defined for load balancing Workflow execution as well as dedicated failover.

In addition to the new long running, fault tolerant Workflow capabilities, Neuron ESB 3.5 provides a number of new features and enhancements that make using, managing and interacting with Service and Adapter endpoints easier and faster. Some of these include:

- Composition using Adapter and Service Endpoints
- WS-Discovery enabled runtime, parties and endpoints
- REST and WMI enabled Endpoint Health Monitoring
- NetSuite and Dynamics CRM 2013 Online
- Neuron ESB Explorer User Experience (UX) improvements

All the changes included in the 3.5 release can be found in the Neuron ESB Change Log which gets installed with Neuron ESB. Users can download the latest Neuron ESB release from the Neuron ESB web site download page.

Neuron ESB Workflow Environment

Neuron ESB 3.5 Workflow brings long-running transactions and complex business processes to the enterprise service bus. 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 3.5 provides a complete Workflow hosting environment for running workflows as part of, or independent of, your ESB messaging solution.

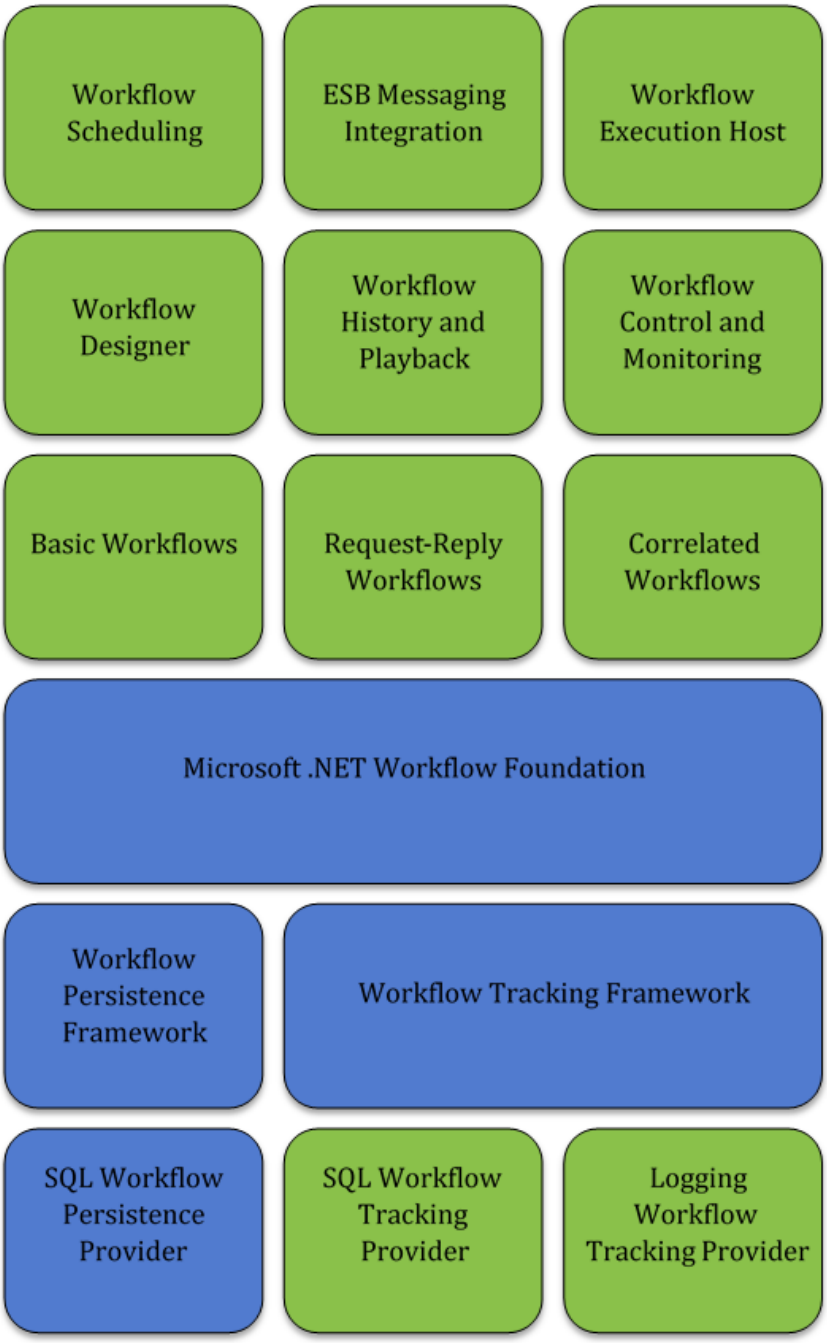

Figure 3 Neuron ESB Workflow – Green boxes are Neuron ESB provided Infrastructure, tools and runtime services

Features

Figure 3 illustrates the features of Neuron ESB 3.5 Workflow. The blue highlighted elements are the core features of the Microsoft .NET Workflow Foundation (WF) that come as part of the .NET Framework.

The green highlighted elements are the additional features that Neuron ESB provides on top of WF for use in enterprise environments.

Neuron ESB's 3.5 Workflow is built on WF that was originally introduced in .NET 4.0 and improved upon in .NET 4.5. 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
- Workflow Types
- Workflow Execution Environment
- ESB Message Integration
- Workflow Tracking and Playback
- Workflow Control and Monitoring
- Workflow Samples

Workflow Designer

Neuron ESB hosts its own Workflow Designer within the Neuron ESB Explorer. The Workflow Designer hosted in the Neuron ESB Explorer is the same designer that developers use inside of Microsoft Visual Studio to design and build workflows for .NET applications. With Neuron ESB, developers do not need to leave the Neuron ESB Explorer environment or have Visual Studio installed in order to build and edit workflows. Neuron ESB maintains compatibility with Visual Studio workflows allowing workflows to be imported and exported between both environments.

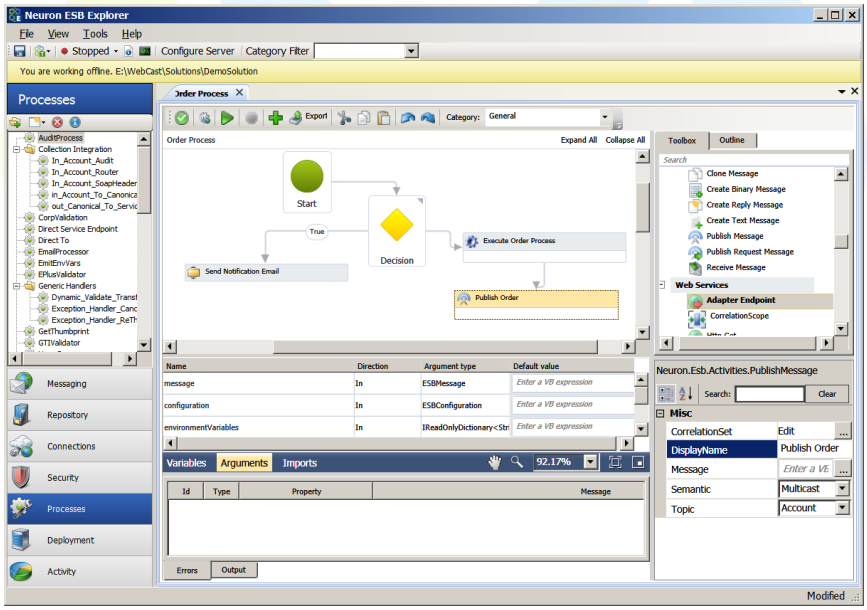

Figure 4 Neuron ESB Workflow Designer – Located within the Neuron ESB Explorer. The toolbox to the right of the designer contains all the out-of-the-box workflow Activities including Neuron specific activities. Over70 activities are included.

Using the Neuron ESB Workflow Designer, users can create new Workflows or import existing Workflows previously created in Visual Studio by selecting either “Create Workflow” or “Import Workflow” from the Processes toolbar.

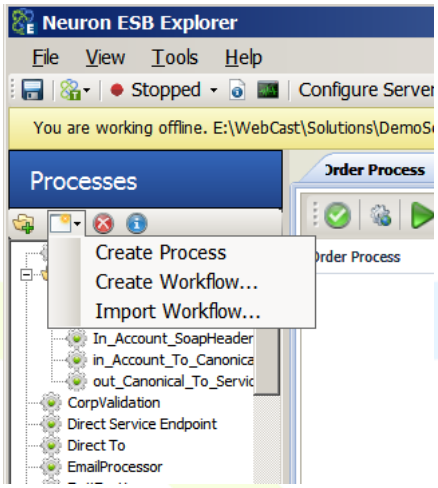

Figure 5 Neuron ESB Processes Menu – Users can create or import existing workflows using the Menu items located on the “New” toolbar button.

Workflow Activity Toolbox

The Workflow Activity Toolbox is located to the right of the Workflow Designer and contains 79 Workflow Activities. Although many are the standard Workflow Activities that ship as part of WF, others are Neuron ESB specific that enable interaction with Neuron ESB Messaging, Adapters and Service Endpoints or provide some additional level of interaction with the Neuron ESB Messaging system.

Neuron ESB also supports WF custom developed activities as well as external assembly references which can be added through the Neuron ESB Workflow Designer toolbar.

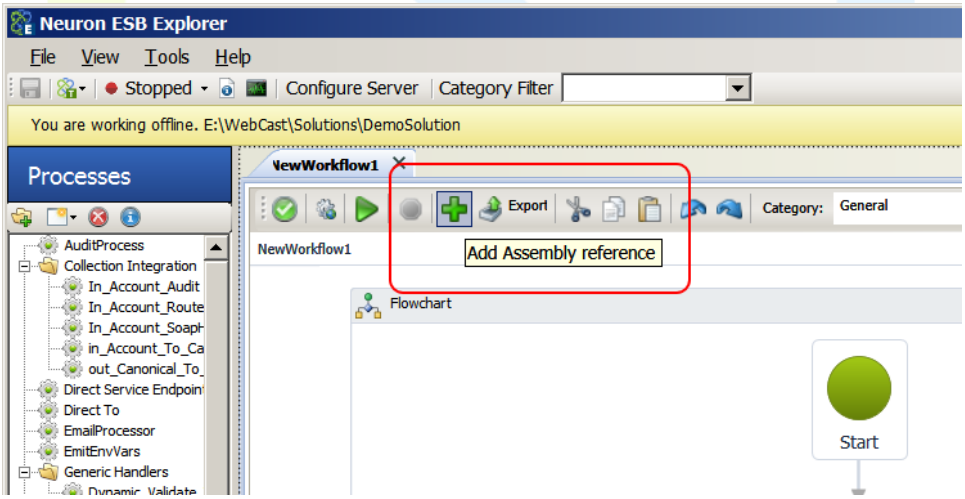

Figure 6 Neuron ESB Add Assembly – Users can add references to external .NET assemblies through the “Add Assembly reference” toolbar button

Custom activities can be added to the Toolbox by copying activity assemblies and dependent assemblies to the Workflows folder under the Neuron ESB instance installation folder and restarting Neuron Explorer (ex: C:\Program Files\Neudesic\Neuron ESB v3\DEFAULT\Workflows). When custom activities are added, they will show up in the Toolbox where they can then be dragged onto the surface of the Workflow Designer. The out-of-the-box Neuron ESB specific Workflow Activities are listed in the table below.

 Comment Out	The Comment Out activity is useful when you need to disable the execution of one or more activities, but do not want to delete the activities from the workflow. The Comment Out activity is a container activity that can hold a single activity or multiple activities contained in a Sequence activity. At execution time, the Comment Out activity will prevent the activities that it contains from executing.
 C#	The C# activity executes a C# code fragment. The code fragment is compiled at runtime into a dynamic assembly. The C# code activity can reference classes in the .NET Framework or reference third-party libraries. The C# code activity can also interact with any variables of the workflow that are accessible to the activity. This is identical to the C# Process Step and supports full IntelliSense, and opens into a Visual Studio style editor.
 C# Class	The C# Class activity gives the developer more freedom than the C# activity and allows the developer to define a custom class that will be executed. The C# Class activity has the ability to get or set the values of variables in the activity's scope. The main advantage of the C# Class activity over the C# activity is that the developer can define methods and make larger code fragments more readable than the C# activity. This is identical to the C# Class Process Step and supports full IntelliSense, and opens into a Visual Studio style editor.
 JavaScript	The JavaScript activity allows the user to execute dynamic code written in JavaScript. The JavaScript activity uses the Google v8 JavaScript engine to execute workflow-specific custom logic. This is identical to the JavaScript Process Step and opens into a Visual Studio style editor.
 Visual Basic .NET	The Visual Basic .NET activity executes a Visual Basic .NET code fragment. The code fragment is compiled at runtime into a dynamic assembly. The Visual Basic .NET code activity can reference classes in the .NET Framework or reference third-party libraries. The Visual Basic .NET code activity can also interact with any variables of the workflow that are accessible to the activity. This is identical to the Visual Basic .NET Process Step and supports full IntelliSense, and opens into a Visual Studio style editor.
 Execute Process	The Execute Process activity allows developers to reuse existing message processing that has been built using Neuron ESB's Business Process designer. With the Execute Process activity, a workflow developer can choose an existing Business Process to execute and can send a message into the process and capture the message output by the process.
 Deserialize Data Contract	The Deserialize Data Contract activity will accept an XML message and will use the .NET DataContractSerializer class to deserialize the XML into an object. The Deserialize Data Contract activity is a generic workflow activity and the workflow developer will be prompted at design time to choose the class type for the deserialized object.

 Serialize Data Contract	The Serialize Data Contract activity will take a DataContract object and will serialize the object to XML using the .NET DataContractSerializer class.
 Clone Message	The Clone Message activity will accept an ESBMessage object and will create an identical copy of the ESBMessage. This activity can be useful when publishing the message to other topics or when it is necessary to change the body or headers of the message during workflow processing.
 Create Binary Message	Create Binary Message will create an ESBMessage object containing a binary body. The Create Binary Message activity accepts the body as an array of bytes and will output an ESBMessage object.
 Create Reply Message	The Create Reply Message activity will accept an ESBMessage object and will generate another ESBMessage that can be used to return a reply for the original message. The Create Reply Message activity is useful when building Request/Reply workflows
 Create Text Message	The Create Text Message activity will create an ESBMessage object containing a text body. The Create Text Message activity accepts a body as a string and will output an ESBMessage object.
 Publish Message	The Publish Message activity will let a workflow publish a message to a Neuron ESB topic. The Publish Message activity can specify correlation settings that the workflow runtime will use to find and route reply messages back to the workflow. If a Correlation Set is used, the Semantic property must be set to Multicast. When testing at design time the solution must be loaded into the local Neuron ESB Runtime. The runtime must be started. Lastly, a valid Source ID and Topic must be entered into the Edit Message Dialog when initiating the test.
 Receive Message	The Receive Message activity is used in correlated workflows to receive messages that have been routed to the workflow based on configured correlation settings. When running correlated workflows, messages will be queued in the Neuron ESB database for the workflow instance. The Receive Message activity will retrieve the next message from the queue and will return the message to the workflow for processing.
 HTTP GET	The HTTP GET activity will execute an HTTP GET request against an HTTP service such as a web service or website. The HTTP GET activity will return the body of the data that was received from the remote web service.
 HTTP POST	The HTTP POST activity will execute an HTTP POST request against a web service or website. The HTTP POST activity will allow the workflow to specify the body of the request to be sent to the remote web service.
 Write to Event Log	The Write to Event Log activity allows the workflow to report a message to the Windows Event Log.
 Invoke PowerShell	The Invoke PowerShell activities are used to execute PowerShell scripts inside of a workflow. There are two PowerShell activities. The second activity is capable of returning a result from the PowerShell script to the workflow. Each requires the installation of PowerShell 3.0 or greater.
 Is Match?	The Is Match activity evaluates a string value using a regular expression to determine whether the string is a match. The Is Match activity will return a Boolean result.

 Matches	The Matches activity will execute a regular expression against a string value and will return the string fragments that match the specified regular expression.
 Replace	The Replace activity will execute a regular expression against a string value and will replace the matched text with an alternative string or value.
 Database Query	The Database Query activity will execute a query against a SQL database and will return the result of the query.
 Query DataSet	The Query DataSet activity will execute a query against a SQL database and will return the results of the query in a .NET DataSet object for processing.
 Query Scalar	The Query Scalar activity will execute a query against a SQL database and will return the single result of the query.
 Database Update	The Database Update activity will execute a SQL update command against the database. The Database Update command will typically be used to perform INSERT, UPDATE, or DELETE SQL statements.
 Get Workflow Instance ID	The Get Workflow Instance ID activity is useful when the workflow needs to know the identifier for the executing workflow. Each instance of a workflow has a unique UUID value that is generated automatically when the workflow instance is created. The Get Workflow Instance ID activity will return the identifier for the current executing workflow. This activity can be useful for logging or other diagnostic uses.
 Deserialize From JSON	The Deserialize From JSON activity will accept a JSON string and will deserialize the JSON into an object. Deserialize From JSON uses the JSON.NET library to deserialize the JSON object into a .NET object.
 Serialize To JSON	The Serialize To JSON activity will take a .NET object and will generate the JSON representation of the object. The Serialize To JSON activity uses the JSON.NET library to serialize the object to JSON.
 Publish Request Message	The Publish Request Message activity will publish a message as a request to a Neuron ESB topic and will wait for the reply to be returned. While waiting for a reply, the workflow may become idle and unload to allow another workflow to execute. When the reply is received, the workflow will be reloaded from the database and will continue execution. When testing at design time the solution must be loaded into the local Neuron ESB Runtime. The runtime must be started. Lastly, a valid Source ID and Topic must be entered into the Edit Message Dialog when initiating the test.
 Adapter Endpoint	The Adapter Endpoint activity allows a workflow to call a configured Neuron ESB adapter endpoint directly without the need of publishing a message to an ESB topic. The Adapter Endpoint activity will allow the workflow to send a message to or receive a message from a configured Neuron ESB adapter endpoint.
 Service Connector	The Service Connector activity allows a workflow to send a message to a configured Neuron ESB service connector directly without the need of publishing a message to an ESB. The Service Connector activity is useful when interacting with web services where a service connector is already configured. When testing at design time the solution must be loaded into the local Neuron ESB Runtime. The runtime must be started. Lastly, a valid Source ID and Topic must be entered into the Edit Message Dialog when initiating the test.

 Audit Message	The Audit Message activity functions identically to the Audit Message Process Step. Allows messages to be selectively audited into the Neuron ESB message history or failed message reports. When testing at design time the solution must be loaded into the local Neuron ESB Runtime. The runtime must be started. Lastly, a valid Source ID and Topic must be entered into the Edit Message Dialog when initiating the test.
 Split Xml Message	The Split Xml Message activity allows users to split incoming batch files into their individual records, storing those records into a List<ESBMessage> collection that can be operated on. Users can configure the Split Xml Message with an XPATH statement to determine the boundary of individual records within the batch file.
 Join Xml Messages	The Join Xml Messages activity compliments the Split Xml Message. It takes the collection of messages (i.e. List<ESBMessage>) and outputs them as a single aggregated message. Users can define the root node as well as namespace for the outgoing batch file.
 Validate Xml	The Validate Xml Activity can be used to validate an Xml message against a set of Xml schemas. This activity is identical in both functionality and property configuration to the Validate – Schemas Process Step. If validation is not successful, an exception is thrown indicating the reason why validation failed.
 Transform Xml	The Transform Xml Activity can be used to apply an Xslt/Xsl transform to an XML message. Additionally parameterized Xslt/Xsl is supported. Parameters can be useful when the same value must be repeated many times within the document. This activity is identical in both functionality and property configuration to the Transform - Xslt Process Step. If the transform is not successful, an exception is thrown indicating the reason why transformation failed.

Workflow Simulation

Neuron ESB's Workflow Designer supports similar testing/simulation features as the Neuron ESB Process Designer. Using the Neuron ESB Workflow Designer, users have the ability to simulate workflow execution for testing at development time. Neuron ESB Explorer's test runner allows developers to pass messages to the workflow and to monitor the workflow's execution in real-time. Several of the workflow activities also support the simulated runtime environment. For example, the *Receive Message* activity will allow developers to test receiving messages during a workflow's execution. There are other Workflow Activities that support a combination of design time and run time testing. Specifically, the Publish Message, Publish Request Message, Audit Message, Adapter Endpoint and Service Endpoint Workflow Activities can all be tested at design time while interacting directly with the local Neuron ESB runtime solution. To test these, the local Neuron ESB runtime service needs to be loaded and started with the current solution and a valid Source ID and Topic must be provided in the Edit Test Message dialog.

Simulation/Testing is started by pressing the Test Workflow button on the Workflow Designer's toolbar as shown in Figure 6. This will prompt the user for a message via the Edit Test Message dialog (the same dialog used in the Business Process Designer). When the user presses the OK button, the simulation will begin. Each step that executes will be highlighted in Yellow, with any outputs (WriteLine workflow activity or anything that writes to System Console) being written to the Output window located at the

bottom of the Workflow Designer. Any errors that occur will be written to the Errors window located at the bottom of the Workflow Designer.

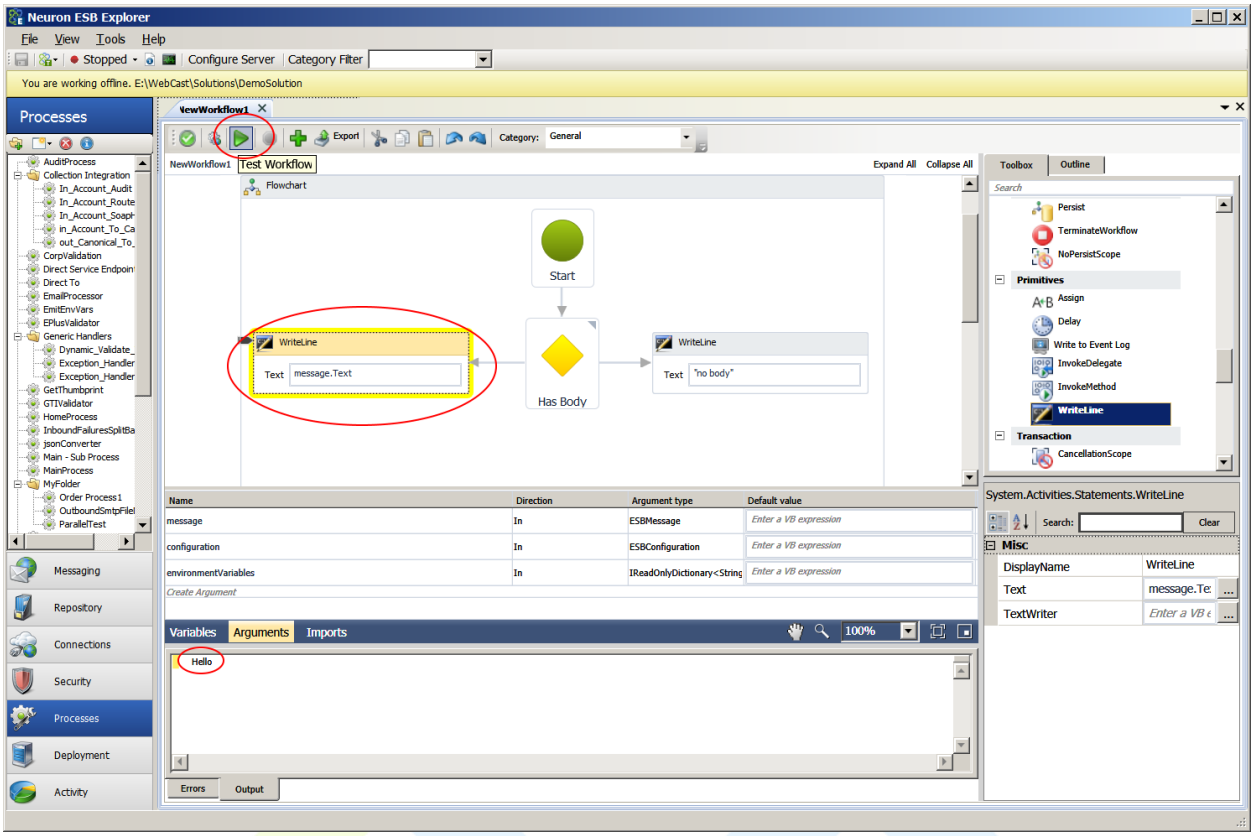

Figure 7 Neuron ESB Workflow Simulation – Workflow Simulation by pressing "Test Workflow" button on toolbar. Each shape executed becomes highlighted in Yellow at time it's executed. Outputs are written to the Output window.

Workflow Types

When users select the "Create Workflow…" option from the Neuron ESB Processes menu, they are presented with a prompt that allows them to choose from 3 basic types of Workflows i.e. Normal, Request/Reply or Correlated Workflow shown in figure 7.

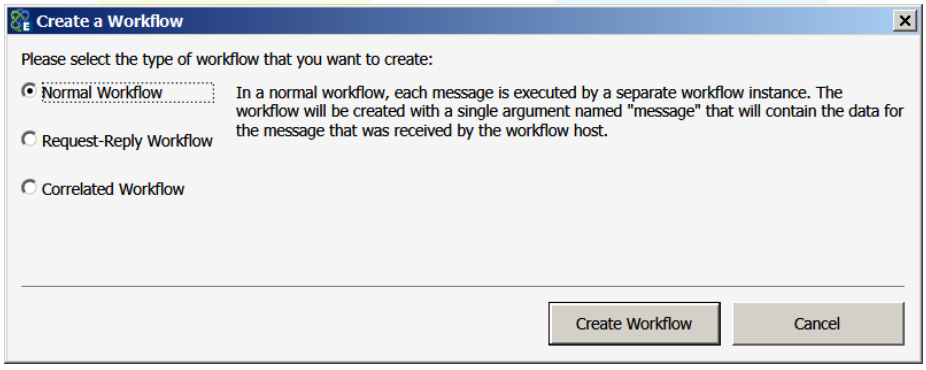

Figure 8 Neuron ESB Workflow Types – Users are prompted to select between Normal, Request-Reply or Correlated Workflow to create.

Normal Workflows are used primarily where a response is not expected to be returned from the Workflow. Request – Reply Workflows are what they infer, where a Request message starts a Workflow and that Workflow instance sends back a response to the client/system that initiated the original Request. Correlated Workflows are a special type that defines a unique set of messages to be processed by a single instance of a Workflow.

Normal Workflow

Normal Workflow types are most commonly used to create Workflows that subscribe to messages and execute an instance of a Workflow for each message received. When a user selects to create a *Normal Workflow*, a Workflow definition is created within the Workflow Designer. Once the Workflow is completed, it's must be saved and associated with a Workflow Endpoint. A Workflow Endpoint is used to associate the Workflow definition with a Neuron ESB Subscriber, Topic and Availability Group. Workflows essentially "subscribe" to messages published to the bus.

When a Normal Workflow is created within the Workflow Designer, the Neuron.Esb namespace as well as 3 arguments specific to Neuron ESB Messaging are added to the Workflow definition, allowing any activity within the Workflow to interact directly with Neuron ESB Messaging or Configuration.

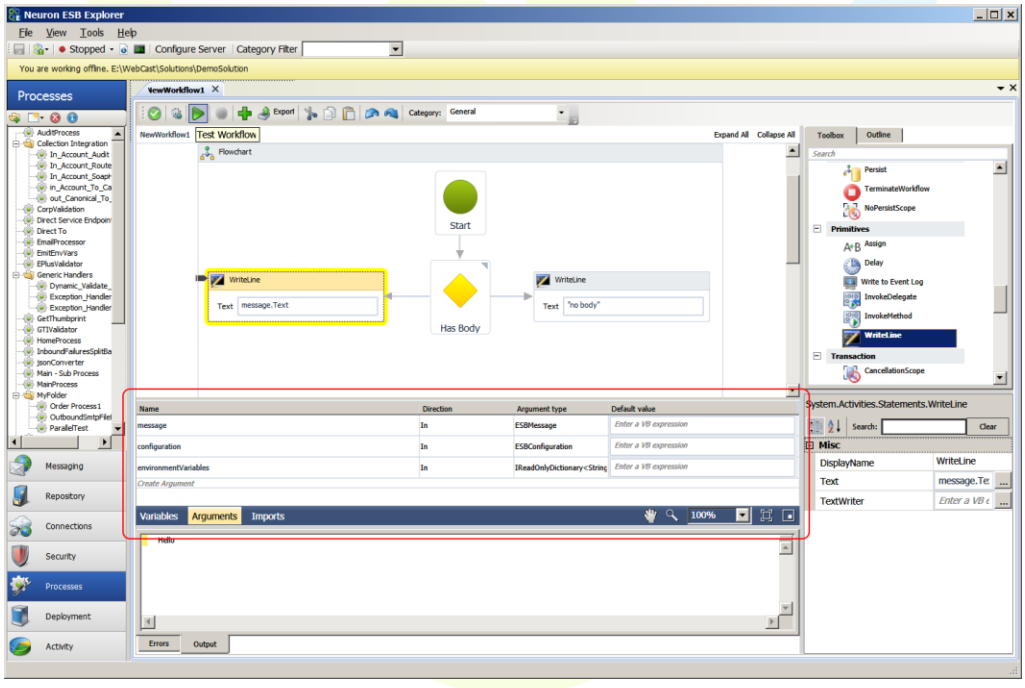

Figure 9 Neuron ESB Arguments Window – Encircled in red, the Arguments window contains the message, configuration and environmentVariables arguments.

As shown in Figure 9, these arguments can be used directly within any activity, including all the Neuron ESB Code Activities such as the C#, C# Class, JavaScript and Visual Basic.NET Activities.

The *message* argument represents the original Neuron ESB Message that the Workflow will be initiated by at runtime. All of its content and properties are accessible both during design time testing as well as runtime as shown in Figure 10.

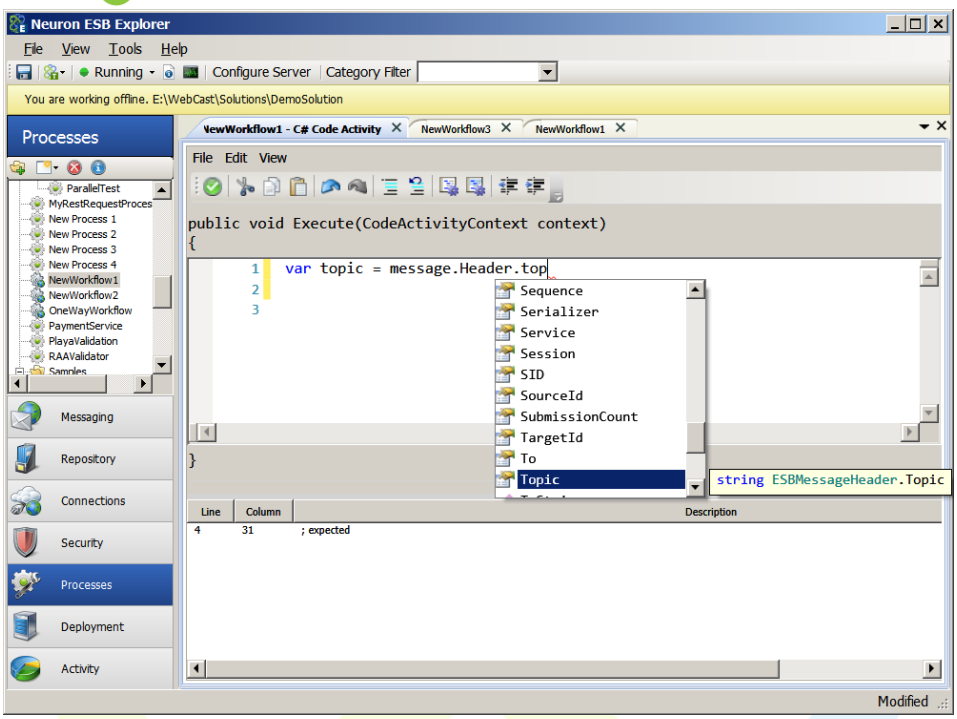

Figure 10 Neuron ESB message Argument – The Neuron ESB Message (message) Argument accessed within a C# Code Workflow Activity. This same argument can be used within any Workflow Activity.

The *environmentVariables* argument can be used to retrieve values specific to the runtime environment that the Workflow instance is running in (i.e. Production, Staging, QA, etc.). Neuron ESB Environment Variables are defined and managed within the Neuron ESB Explorer and located under *Deployment->Environments->Environment Variables* section and can be used to configure any Business Process step, Database connection, Adapter and Service Endpoints. Many developers will create application specific Environment Variables, retrieve them at runtime and use their values to drive the business logic within their custom Business Processes or Workflows.

Lastly, the *configuration* argument provides access to the entire Neuron ESB solution configuration. Almost all elements of a Neuron ESB solution can be accessed through this object. This can be useful to retrieve XSLT or XML documents, encryption keys, certificates and any other entity contained within the Neuron ESB Solution.

Request-Reply Workflow

Request-Reply workflows are easy to build using the Neuron ESB Workflow Designer by selecting *Request-Response Workflow* in the prompt displayed in figure 11.

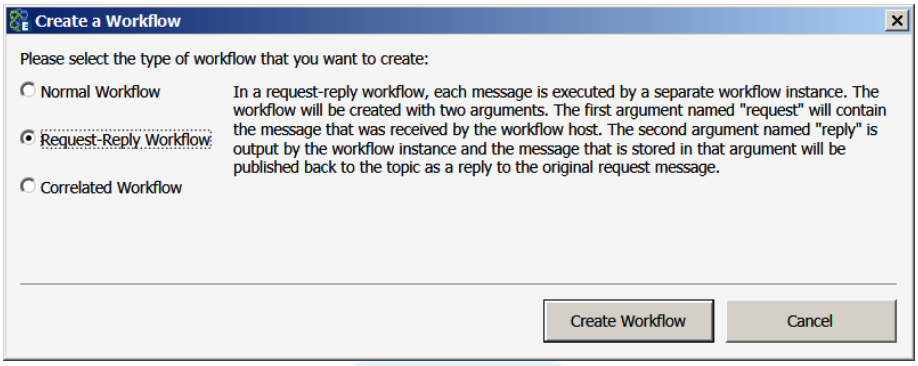

Figure 11 Neuron ESB Workflow Types – Users are prompted to select between Normal, Request-Reply or Correlated Workflow to create. Request-Reply is displayed

By following the pre-defined pattern of accepting a request as an input argument and outputting a message when the workflow terminates, the Neuron ESB workflow engine will automatically return the output message as a reply to the Neuron ESB Party that originally sent the request message. This could be a remotely hosted Neuron ESB Party, an Adapter or even a Neuron ESB Client Connector. The Neuron ESB workflow engine was designed to fit into the messaging patterns employed by Neuron ESB users.

Request-Reply Workflow arguments differ slightly from Normal Workflows. The inbound Neuron ESB Message argument is named *request* (named *message* in Normal Workflows), while the outbound Neuron ESB Message argument is named *reply*. Use the *Create Reply Message* activity to set the *reply* argument and additional code to set its properties. The *reply* argument contains the final Neuron ESB Message that is returned to the original calling Neuron ESB Party as shown in figure 12.

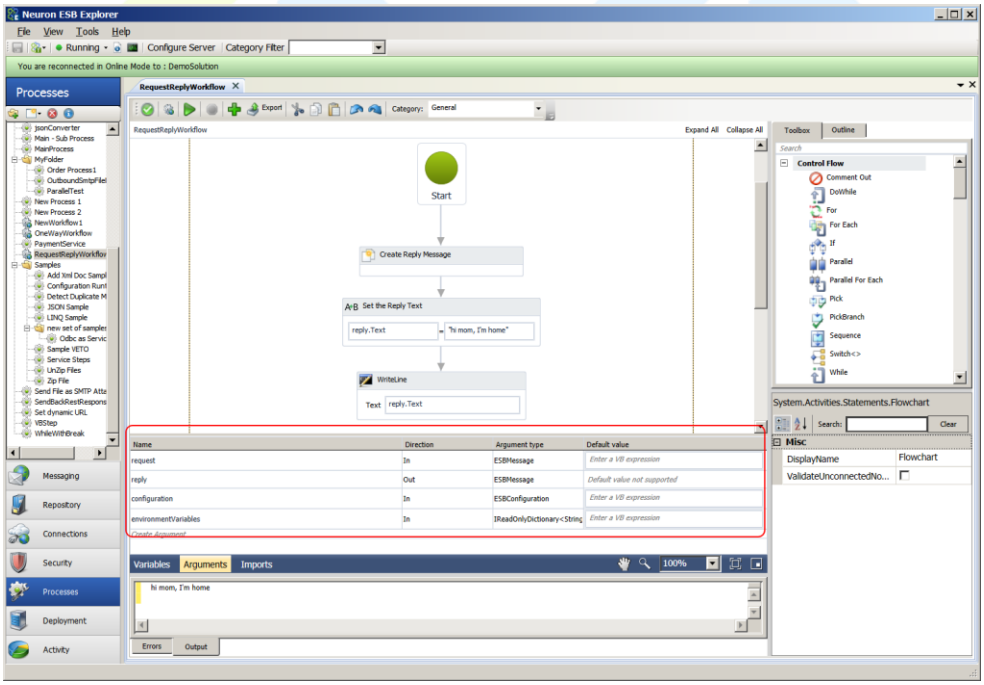

Figure 12 Neuron ESB Arguments Window – Encircled in red, the Arguments window contains the request, reply, configuration and environmentVariables arguments.

Correlated Workflows

The last type of Workflow that can be created is a *Correlated Workflow* as shown in figure 13.

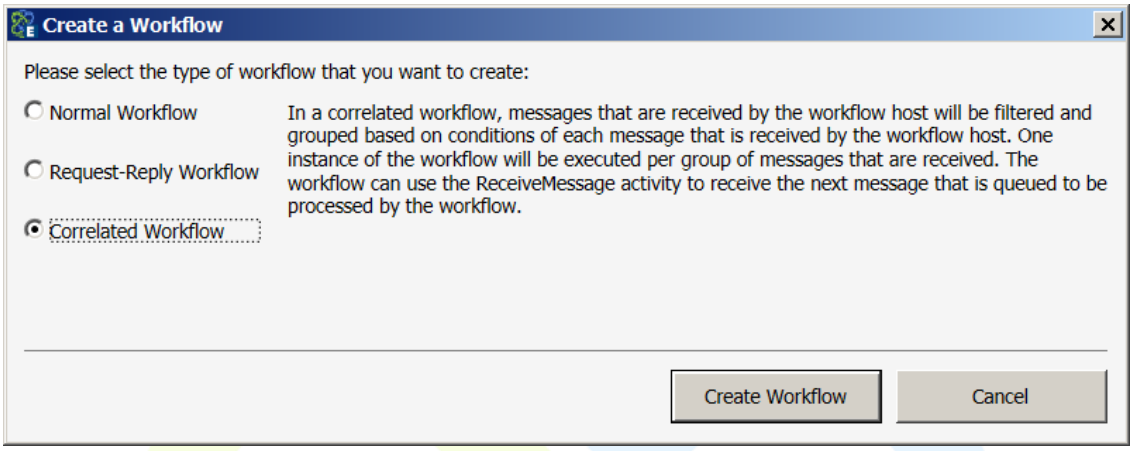

Figure 13 Neuron ESB Workflow Types – Users are prompted to select between Normal, Request-Reply or Correlated Workflow to create. Correlated Workflow is displayed

Correlated Workflows are a special type of Workflow that employs *Custom Correlation* at the workflow endpoint level. Custom Correlation is user-defined criteria that controls what set of messages a single instance of a Workflow will process. This set of messages will be routed to a specific instance of a Workflow (i.e. Singleton pattern), whether that Workflow is running or currently dehydrated in the database. Custom Correlation determines the "*uniqueness*" of a set of messages. Although many instances of a Workflow may still execute, each instance could be processing a set of messages received from the bus, rather than just one, as in the case of Normal Workflows. The first message that launches the instance of the workflow is used to initialize the values of the Correlation Set.

For example, if all messages from a single publisher needed to be processed by the same workflow, a user can choose to correlate messages based off of the session identifier and source identifier. This will result in all messages sent from the same party on the same connection to be processed by the same workflow instance.

Another example would be where a large file was previously split into individual records and published to the bus. A Workflow could be used to aggregate all of the related messages (the individual records) into a new outgoing file. In this case, any combination of message content, custom header, SOAP or HTTP header or even regex and XPATH expressions against the body of the message could be used to define the correlation set for a Workflow instance.

A *Receive Message* Workflow Activity is used within a Correlated Workflow to receive all messages that match the Correlation Set definition on the Workflow Endpoint. Usually this is placed within a While or similar loop within the Correlated Workflow. Hence the *Receive Message* Workflow Activity "follows" the Correlation Set that was initialized when the Workflow Instance was first started (Figure 14).

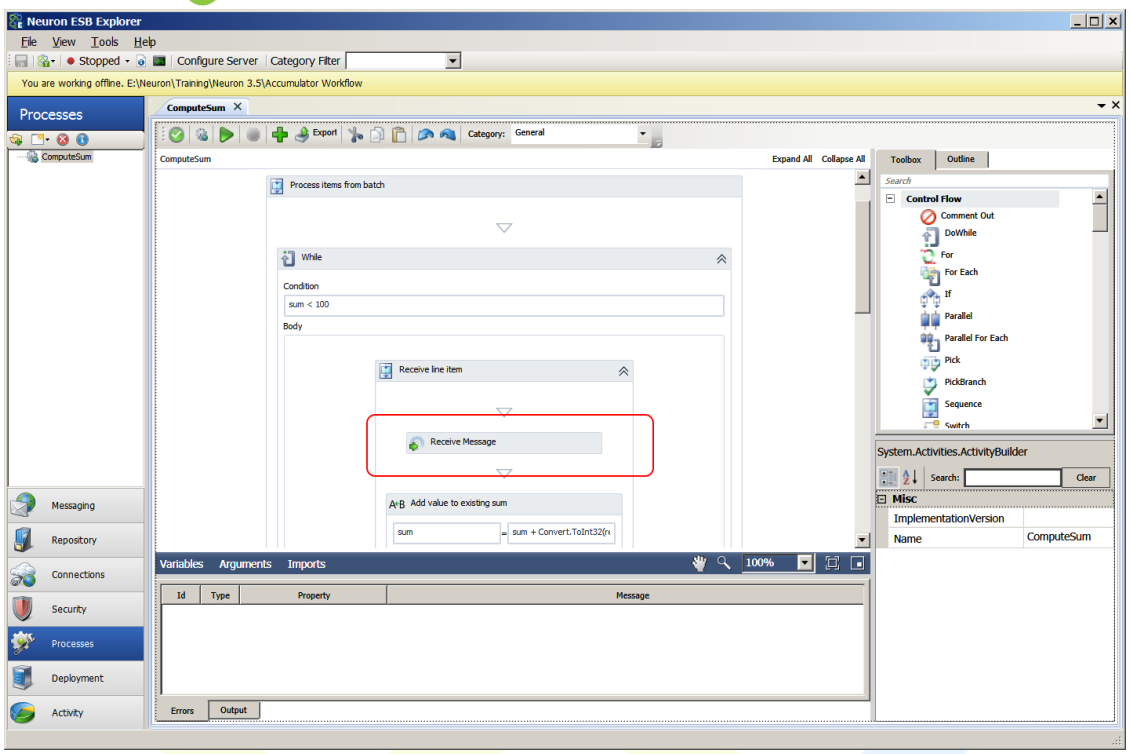

Figure 14 Neuron ESB Sample Correlated Workflow– Encircled in RED is the "Receive Message" Activity. In the sample it is located within a while loop to continually receive messages that match the values of the Correlation Set initialized by the first message that started the Workflow instance. The Receive Message activity will continue to read messages from the internal Workflow queue while the sum is less than 100. Once that value is hit, the current instance will complete and a new instance will be created if there are messages pending in the queue to be processed.

When a user creates a Correlated Workflow and assigns it to a Workflow Endpoint, the Correlation Set tab of the Workflow Endpoint is enabled. This allows the user to define the necessary properties that control what messages are processed together by a single instance of a Workflow as shown in the figure below.

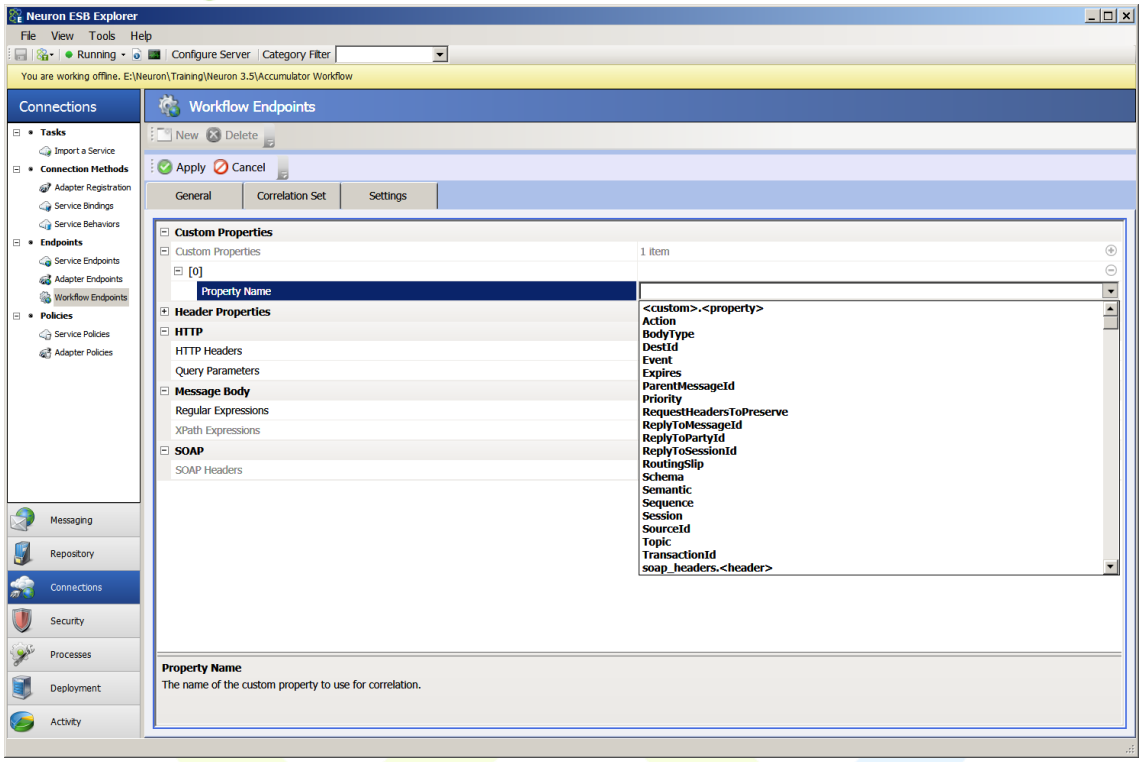

Figure 15 Neuron ESB Workflow Endpoint Correlation Set tab– Users can define the criteria that will determine what set of messages will be processed by each instance of a Workflow. Users can define any combination of either custom properties, Neuron ESB Header properties, SOAP headers, HTTP Headers or Regex or XPATH expressions against the body of each message.

Correlation is not limited to Correlated Workflows. Correlation sets can also be defined within any type of Workflow using the *Publish Message* Workflow Activity followed by the *Receive Message* Workflow Activity.

Workflow Execution Engine

Neuron ESB introduces Workflow Endpoints and Availability Groups for hosting workflows. A Workflow Endpoint is a first-class citizen in the Neuron ESB ecosystem, similar to Service Endpoints or Adapter Endpoints. A Workflow Endpoint is associated with a specific Workflow Definition created with the Neuron ESB Workflow Designer. It acts in a subscriber role and is configured to receive messages from a Topic by a specific Subscribing Party. Lastly, it's assigned to a specific Availability Group which serves as the runtime host (Neuron ESB workflow engine) for managing the execution of the Workflow Definition associated with the Workflow Endpoint. Users can create any number of Availability Groups which in turn can be assigned to any number of Workflow Endpoints.

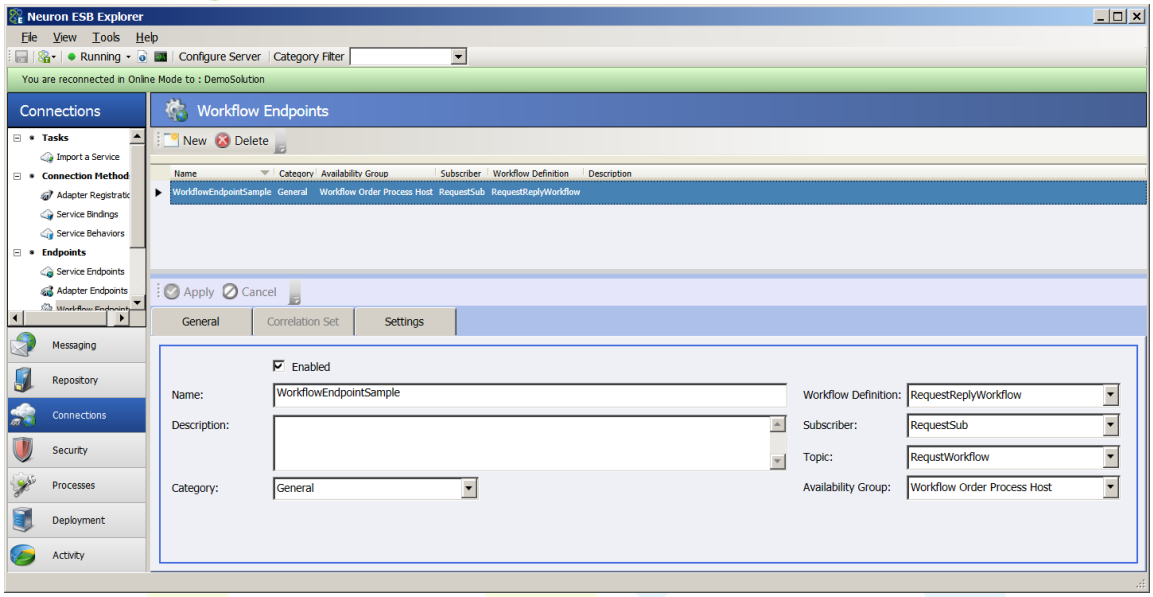

Figure 16 Neuron ESB Workflow Endpoints – The General Tab of Workflow Endpoint allows users to configure the Subscriber ID, Topic, Workflow Definition and Availability Group.

Neuron ESB's Workflow Endpoints also employ configurable limits on workflow execution in order to not overload host servers. The Neuron ESB workflow engine utilizes a persistent queue built inside of SQL Server to schedule the execution of workflow instances. The persistent queue allows workflows to be interrupted if it is necessary to restart the ESB Service or the host service, and the workflow engine can restart work processing when the Workflow Endpoint is restarted.

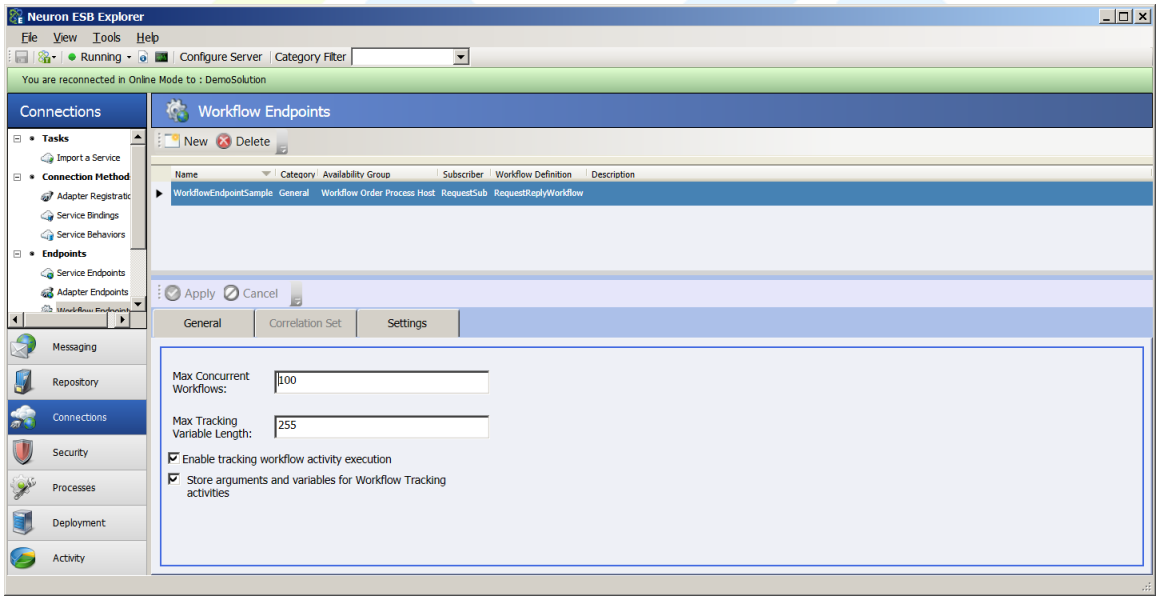

Figure 17 Neuron ESB Workflow Endpoints – The Settings Tab of Workflow Endpoint allows users to configure the Maximum number of concurrent Workflows that should be allowed to run at any one time in the specified hosting environment (i.e. Availability Group). It allows users to set limits on the amount of event tracking data that is collected at runtime.

While Workflow Endpoints contain the configuration specific to setting up the environment for executing a Workflow, Availability Groups use the configuration to provide the actual runtime hosting environment for the Workflow. Availability Groups introduce a new form of isolation and reliability inside of Neuron ESB's Runtime Windows service. Availability Groups are isolated into their own process space and execute as child processes of the Neuron ESB Runtime Windows service. If a fatal error occurs that causes the Availability Group to fail, the Neuron ESB Server can restart the Availability Group on the same server, or on a different server when Neuron ESB is being run in a clustered configuration (see Figure 2).

Through the Deployment Settings tab of an Availability Group, users can configure on what servers, in which specific Deployment Groups they want an Availability Group host to run on. When in a clustered configuration, the Neuron ESB Workflow Engine will schedule execution of Workflows to the selected Primary servers within the clustered configuration, monitoring CPU, Memory and the configurable Max Concurrent Workflows property on the Settings tab of the Workflow Endpoint. Workflow execution will be evenly spread across all configured servers depending on their resource limits.

Deploying Workflows

Neuron ESB 3.5 makes deploying Workflows just as easy as deploying most other entities within Neuron. Once a Workflow Definition has been created and saved (i.e. click the *Apply* button followed by the *Save* button), an Availability Group must be created to serve as the runtime host for the Workflow. If one already exist, it may be practical to reuse it. Once the Availability Group has either been created or identified, a Topic and Subscriber must be created. Lastly, a Workflow Endpoint must be configured for the Workflow Definition using the Availability Group, Subscriber and Topic. Deploying a Workflow involves these 5 entities:

- Workflow Definition
- Availability Group
- Topic
- Subscriber
- Workflow Endpoint

In Neuron ESB 3.5, each Workflow entity created within the Neuron ESB Explorer (i.e. Availability Group, Workflow Endpoint, Workflow Definition) are saved like all other entities; as individual XML files within dedicated folders on disk. The same deployment methods are used with these as are used with other Neuron ESB entities such as Topics and Endpoints. This also means that when changes are saved, or these entities are deployed to another server, the Neuron ESB Runtime and Workflow host will automatically detect the additions, deletions or changes and load them for execution.

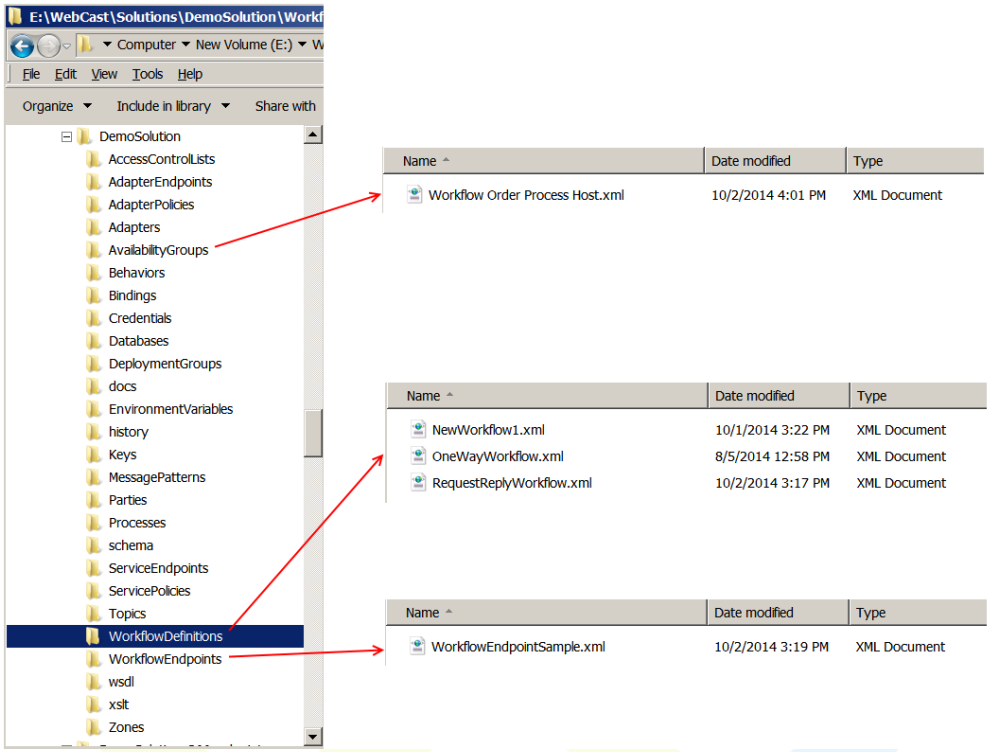

Figure 18 Neuron ESB Workflow Entity Folders – Neuron ESB Workflow Entities are saved as individual XML files in the Neuron ESB Solution directory.

ESB Message Integration

The Neuron Workflow engine and Designer provide a number of integration points into the Neuron ESB Messaging system, some of which are:

- Failed Message Reporting
- Publishing to Topics
- Service Endpoints
- Adapter Endpoints
- Auditing Messages

Failed Message Reporting

Neuron ESB's workflow engine integrates directly with Neuron ESB's auditing service to report on messages that failed while being processed by workflows. This furthers Neuron ESB's goal not to lose messages during transport and processing of messages. Messages that fail during workflow execution can be viewed using Neuron ESB Explorer's Failed Messages report. Additionally, all the messages of a failed or in flight workflow can be inspected through the new Neuron ESB Workflow Tracking console.

To support integration with Failed Message Reporting, additional properties were added to the Failed Message report, specifically the Workflow Name, Workflow Instance ID and Workflow Endpoint Name. This allows users to correlate failed messages back against the specific instances of a Workflow that generated them.

Publishing to Topics

Just as in the Neuron ESB Business Process Designer, users can directly publish messages to Neuron ESB Topics using the *Publish Message* and *Publish Request Message* Workflow Activities.

The *Publish Request Message* Workflow Activity allows users to make request/response type calls over the bus by publishing the request message to a Topic (must be Topic other than that configured in the Workflow Endpoint). Neuron ESB will route the message to a subscribing party which could either be an Adapter, Service Connector or a remotely hosted Party (endpoint). That party will process the request and publish the response back to the bus at which point Neuron ESB will return the response back to the Workflow instance that initiated the request. This can be used for bus driven, decoupled, request/response calls.

Alternatively, the *Publish Message* Workflow Activity can be used to publish messages to a Topic using any of the other Neuron ESB message semantics such as Multicast (fire and forgot), Reply, Direct or Routed. Once published, the Neuron ESB Messaging system will route the message to all eligible subscribers.

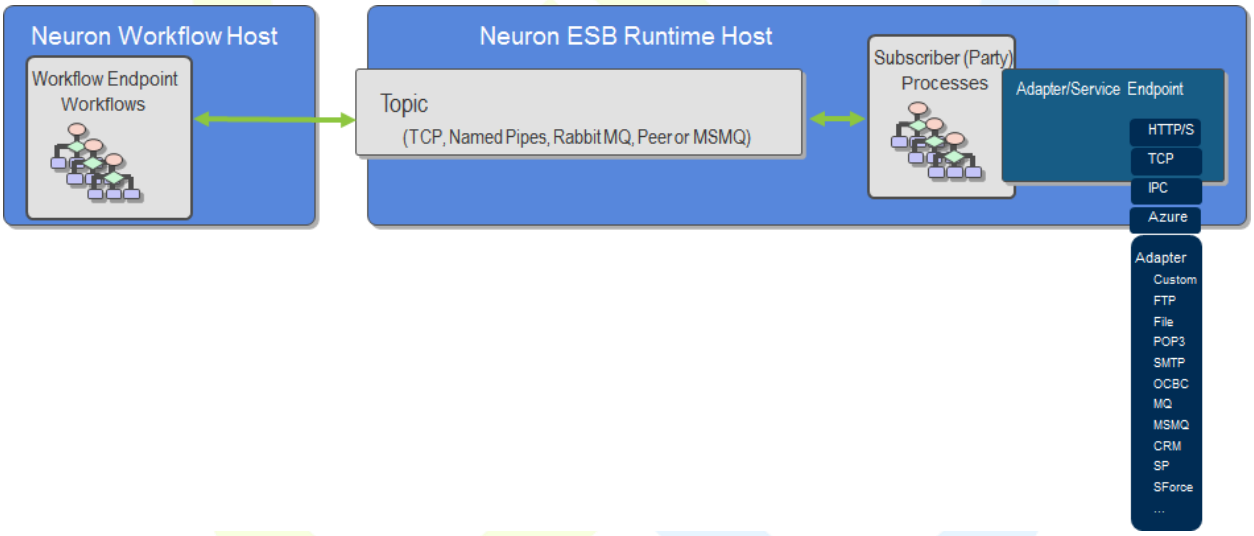

Figure 19 Publish (Request) Message flow– Either the Publish Message or Publish Request Message workflow activity allows for direct integration to the Neuron ESB pub/sub messaging system. These activities can publish directly to any Topic.

The *Publish Message* Workflow Activity can also be used for Correlated request/reply scenarios. Users can define a Correlation Set (a property on the Workflow Activity) and set the semantic to Multicast. Once this is done, the outgoing message that is published to the Topic from the Workflow will be used to initialize the values of the Correlation Set. The initialized values for the Correlation Set are persisted to user defined correlation variable. To receive the expected response message, a *Receive Message* Workflow Activity must be added to the Workflow following the *Publish Message* Workflow Activity and be configured with the correlation variable.

For example, imagine a common partner scenario where a request is sent to a vendor. However, the response may not be received immediately. In fact, it could be hours or days before the response is sent

back from the Vendor. In that time, the Workflow would have been unloaded from memory and persisted to the database. The runtime environment may have also been restarted or even moved. Once the response message is received, it's the job of the Neuron ESB runtime environment to route the response back to the correct "instance" of the Workflow, i.e. the Workflow instance that sent out the original request. In many scenarios like this there will not be a unique identifier within the response message to correlate on. In those cases users can define on the outgoing request message a Correlation Set. *Correlation Set* is a property exposed by the *Publish Message* Workflow Activity. When activated it will display the Correlation Set dialog as shown in the figure below. This is identical to the user interface of the Correlation Set tab of the Workflow Endpoint.

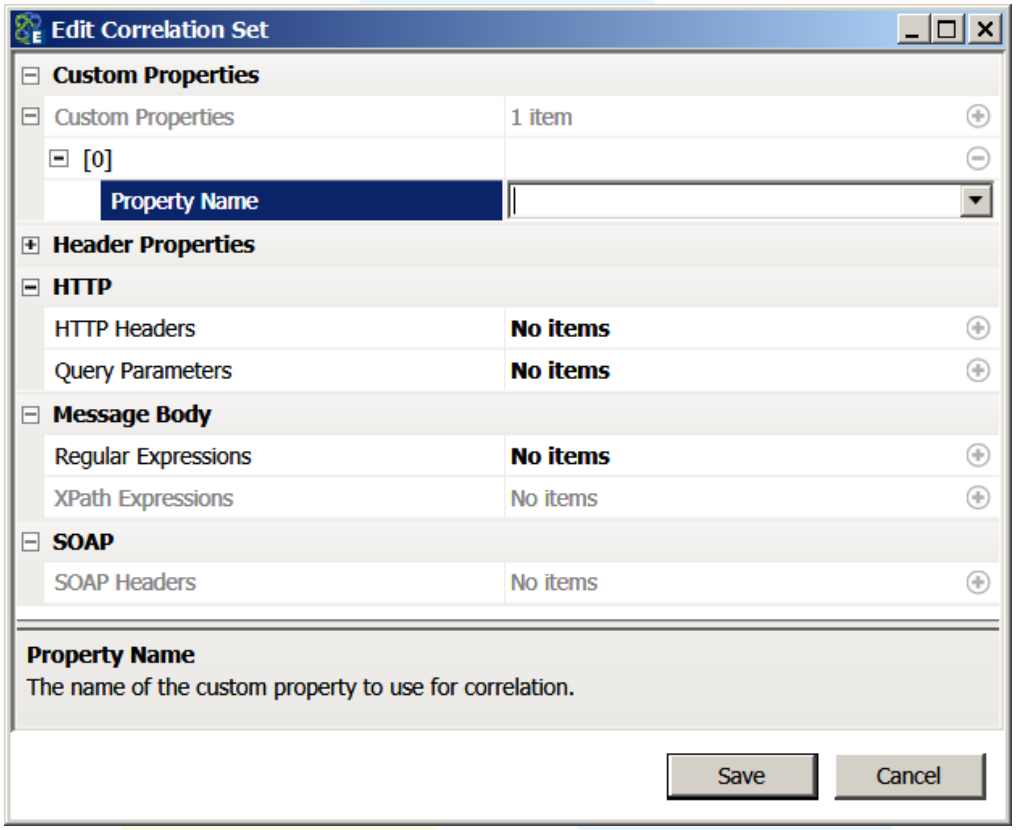

Figure 20 Neuron ESB Publish Message Correlation Set dialog– Users can define the criteria that will determine what set of messages will be processed by each instance of a Workflow. Users can define any combination of either custom properties, Neuron ESB Header properties, SOAP headers, HTTP Headers or Regex or XPATH expressions against the body of each message.

In the case of a purchase order, a user may define a Correlation Set as a combination of customer number, vendor id and purchase order number. When a response message is published to the bus to the expected topic, those property values will be retrieved and evaluated and if a match is found, the response message will be routed to the *Receive Message* Workflow Activity that directly follows the *Publish Message* Workflow Activity on the correct instance of the Workflow. The *Receive Message* Workflow Activity must be used either with a Correlated Workflow, or it must follow a *Publish Message* Workflow Activity.

Service Endpoints

Neuron ESB includes a Service Broker that enables organizations to deploy Neuron ESB as a Service Gateway, providing mediation, security, hosting and a number of other services. Service Connectors are essentially registrations within Neuron ESB that point to existing services hosted within an organization, partner or cloud domain. In previous versions of Neuron ESB, the only way to route a request message to a Service Connector (externally hosted service registered with Neuron ESB) was by publishing the request to a Topic, whose underlying publishing service would then route to the Service Connector. This meant that every web service request received by Neuron had to go through the pub/sub infrastructure if an externally hosted service had to be called. Although the pub/sub Topic Transport of choice would usually be an in-memory option, this still led to additional overhead and configuration at runtime.

Neuron ESB 3.5 provides both a *Service Endpoint* Workflow Activity and Process Step that can be used with either the new Neuron ESB Workflow Designer or the existing Business Process Designer. Either one allows a user to directly call any Service Endpoint registered with Neuron, without the need to publish a request to a Topic, eliminating all pub/sub overhead.

This allows users to create either a Workflow or Business Process that defines an end-to-end solution without the pub/sub abstraction in the middle.

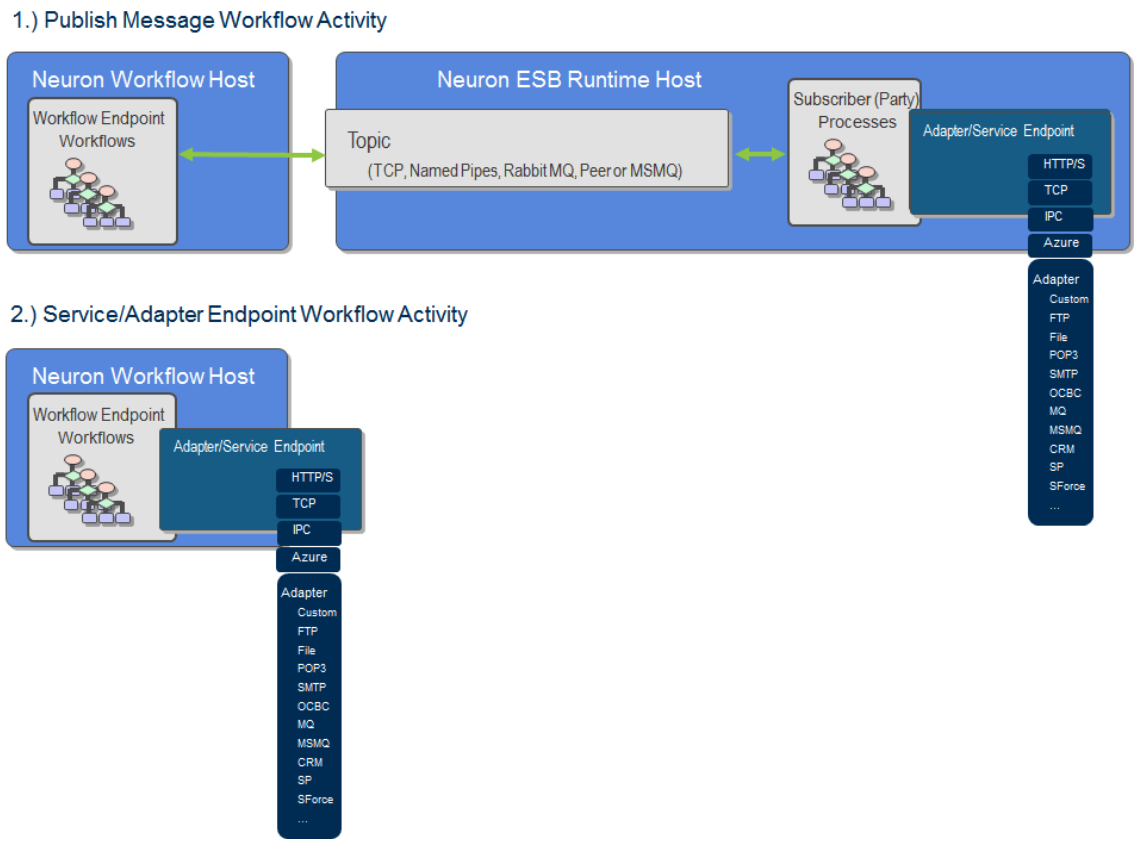

Figure 21 Service Endpoint/Adapter Endpoint Workflow Activities compared with Publish Workflow Activities – When using the Service or Adapter Endpoint Workflow Activities, the entire Neuron ESB Messaging infrastructure is circumvented, allowing registered endpoints to be called directly.

Adapter Endpoints

A core feature of Application and Data Integration servers is their ability to bridge 3rd party applications, databases, technologies, protocols or transports. Neuron ESB provides this through either its library of built in adapters and by allowing users to build their own adapters using the Neuron ESB Adapter Framework. In many ways, Adapters provide capabilities similar to those found in Neuron ESB's Service Broker specifically:

- Bridging external endpoints
- Functioning as subscribers

Just as with Service Connectors, Adapter Endpoints would normally need to be called through the Neuron ESB Messaging system where a message is published to a Topic and then routed to eligible subscribers, one of which could be an Adapter Endpoint.

Neuron ESB 3.5 provides both an *Adapter Endpoint* Workflow Activity and a Process Step that can be used with either the new Neuron ESB Workflow Designer or the existing Business Process Designer. Either one allows a user to directly call any Adapter Endpoint registered with Neuron, without the need to publish a message to a Topic, eliminating all pub/sub overhead (Figure 21).

This allows users to create either a Workflow or Business Process that defines an end-to-end solution without the pub/sub abstraction in the middle. These activities and process steps should always be used with any request/response type of messaging since there will never be more than one subscribing endpoint fulfilling the request. For fire-and-forget messaging (i.e. multicast/datagram), unless there is a need to decouple using the topic-based pub/sub engine as in the case where the publishing process should not know who the subscribing endpoints/parties are, then using these activities and process steps would be a preferred approach.

Auditing Messages

Neuron ESB provides message tracking capabilities at several levels, commonly referred to as Message Auditing. When used, messages either published to or subscribed to through the Neuron ESB Messaging system are placed in the Neuron Audit tables (database) where they can be viewed, searched for, edited and resubmitted back to the bus within the Message History report. The first level most commonly used is configured at the Topic level as shown in the Figure below.

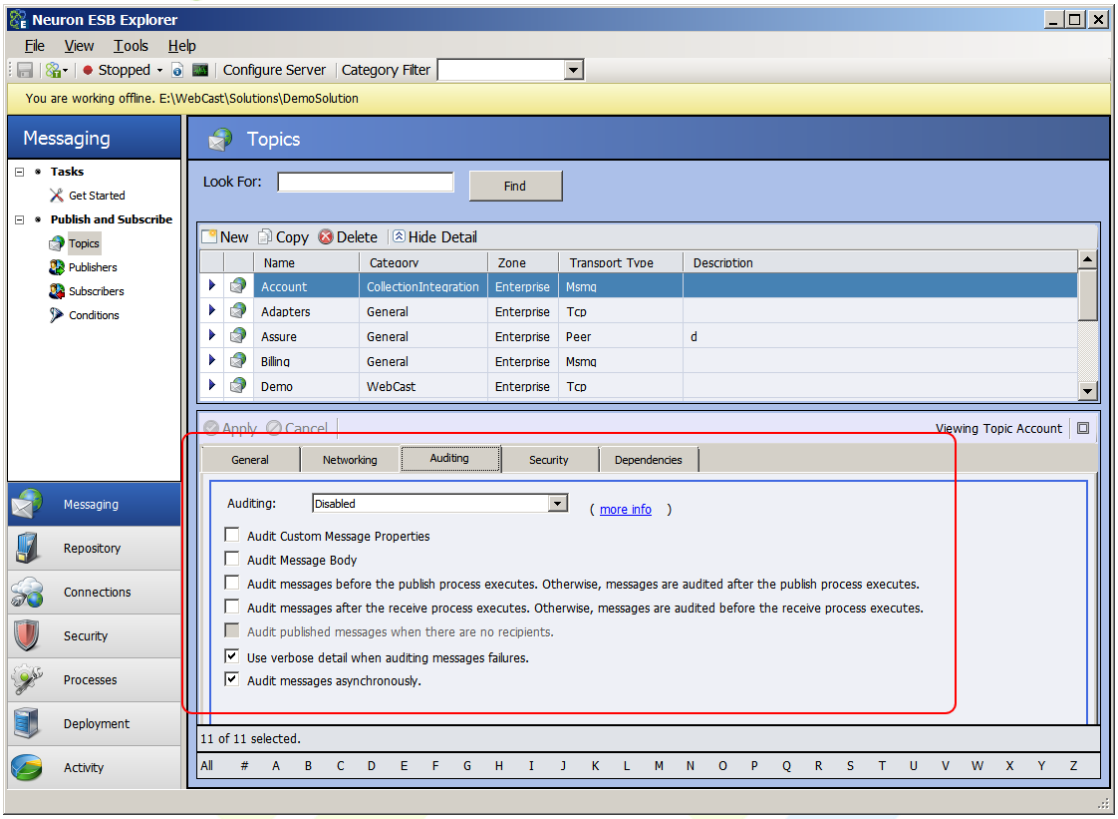

Figure 22 Neuron ESB Auditing Configuration – Global auditing for the Neuron ESB Messaging system can be configured at the Topic level through the Neuron ESB Explorer.

By default, this provides blanket auditing. In other words, all messages that are published to or subscribed to from the Topic will be stored in the Neuron Audit tables. The properties on the Auditing tab of the Topic control the time and granularity of the messaging auditing feature. For example, users can choose to "audit" either before or after a process executes or determine whether or not either the message data or custom properties will get tracked, along with the exception stack. We call this level of Auditing "Global" since it tracks everything that flows through a Topic. In many cases though, users require a more granular level of control over message Auditing. Specifically, the ability to Audit a message at a point in a process where it matters most. In some cases it may be desirable to audit just a fragment of the message rather than the entire message. For those more common scenarios Neuron ESB provides a configurable Neuron Audit Process Step that users can add to an existing Neuron ESB Business Process to strategically Audit a message as part of a defined process.

The Neuron ESB Audit Process Step can also be used to capture an exception in a Business Process and store the affected message, its context and the exception information are placed in the Neuron Audit tables (database) where they can be viewed, searched for, edited and resubmitted back to the bus within the Failed Message report. This similar feature is also exposed by Service and Adapter endpoint policies, whereas if a delivery failure occurs the policy can be configured to capture the message, context and exception information and place all into the Neuron Audit tables, managed through the Failed Message report.

In Neuron ESB 3.5, the Neuron ESB Audit Process Step has been completely ported to a Workflow Activity, where it can be used in exactly the same way as it is used today in a Neuron ESB Business Process. Users can drag the *Audit Message* Workflow Activity onto the Workflow designer, configure it to capture a specific message, set the properties. Now that message will be viewable, searchable and editable within either the Message History or Failed Message reports (depending on context) as shown in the figure below.

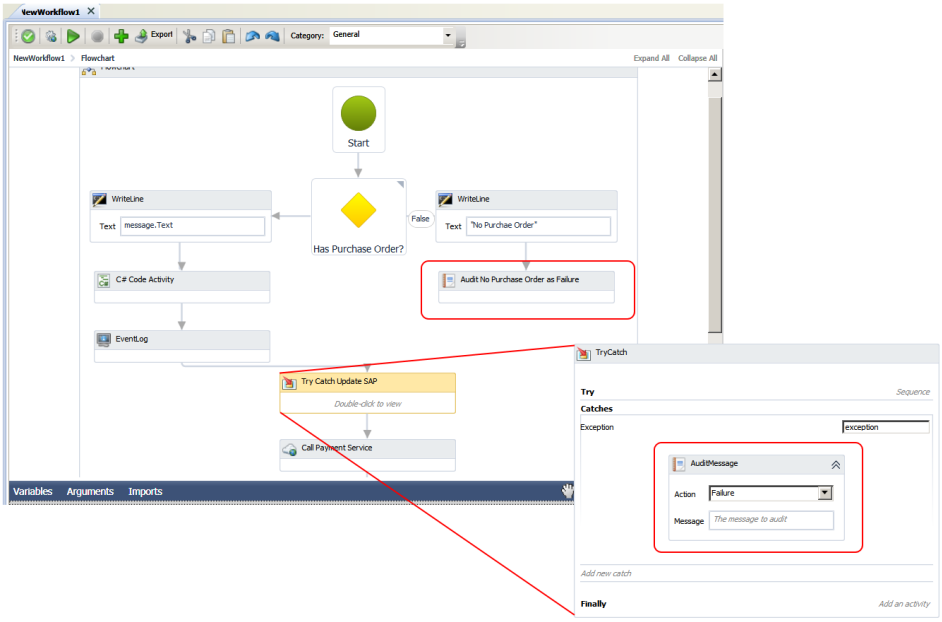

Figure 23 Neuron ESB Auditing within a Workflow – Audit Workflow Activity can be used anywhere within a Workflow to capture the state of the message. It can also be used within a Try/Catch Activity to capture the exception, context and message, storing this information in the Neuron Audit tables where it can be viewed in the Failed Message report.

Workflow Tracking and Playback

Neuron ESB's workflow engine implements proprietary tracking providers that are used to collect and report execution history for the workflow. The primary tracking provider stores workflow execution tracking events and data into a Microsoft SQL Server database. Once collected in the database, Neuron ESB Explorer provides the monitoring interface that developers or administrators can use to observe the status of each executing workflow, and if desired, play back the workflow's execution. Using Neuron ESB Workflow Tracking, users can observe the execution history and state transitions of the workflow. Users can also 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 also able to explore error conditions and view exception messages for errors that occurred during the workflow.

Neuron ESB Workflow Tracking (seen in the Figure below) is the central user interface for querying and viewing the tracking information for all workflow instances; those that are in process as well as those that have completed.

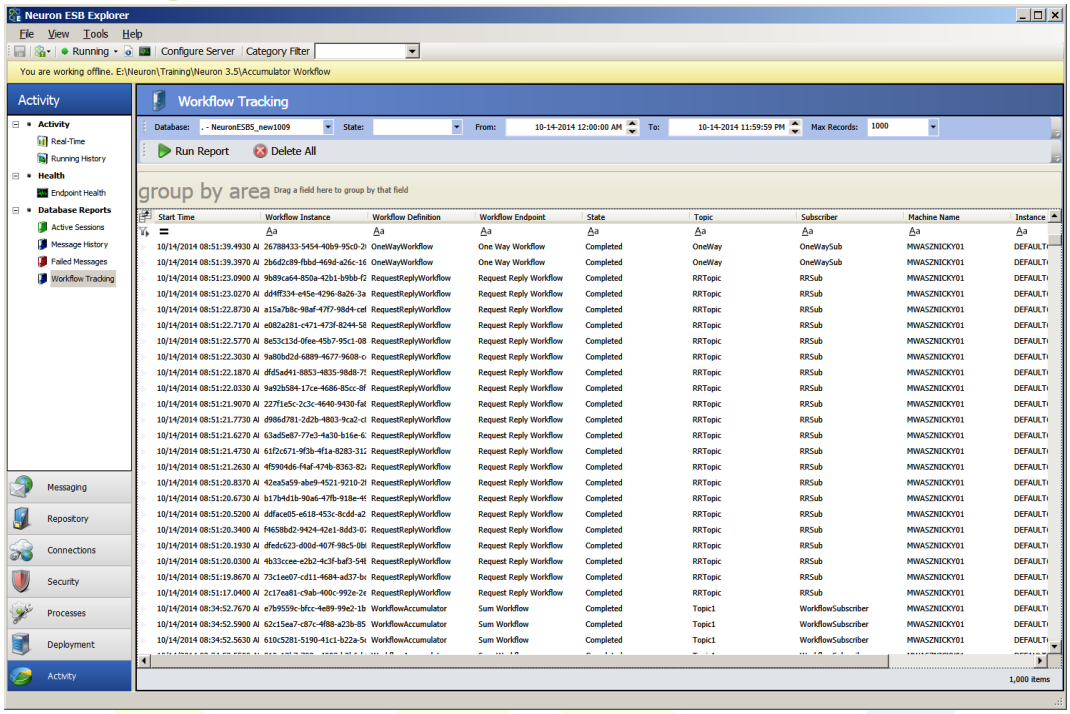

Figure 24 Neuron ESB Workflow Tracking – Provides users the ability to see the state of any Workflow Instance during execution or when completed. Users can group, sort, filter, delete, search and choose which columns to include in the main view.

The Workflow Tracking interface allows users to group, sort, filter, search and delete all information related to the execution of Workflows. Each row in the table represents a unique instance of a Workflow. Double-clicking on any row will bring up the detailed tracking profile of the Workflow Instance. Users can examine the state of the messages, variables and arguments at each stage of the Workflow Instance by navigating through the Tracking events (Tracking Tab), or highlighting a Workflow Activity in the Workflow instance diagram displayed in the main pane. Users can open any ESB Message variable into the Edit Message dialog where it can be edited and resubmitted back to the Neuron ESB Messaging system.

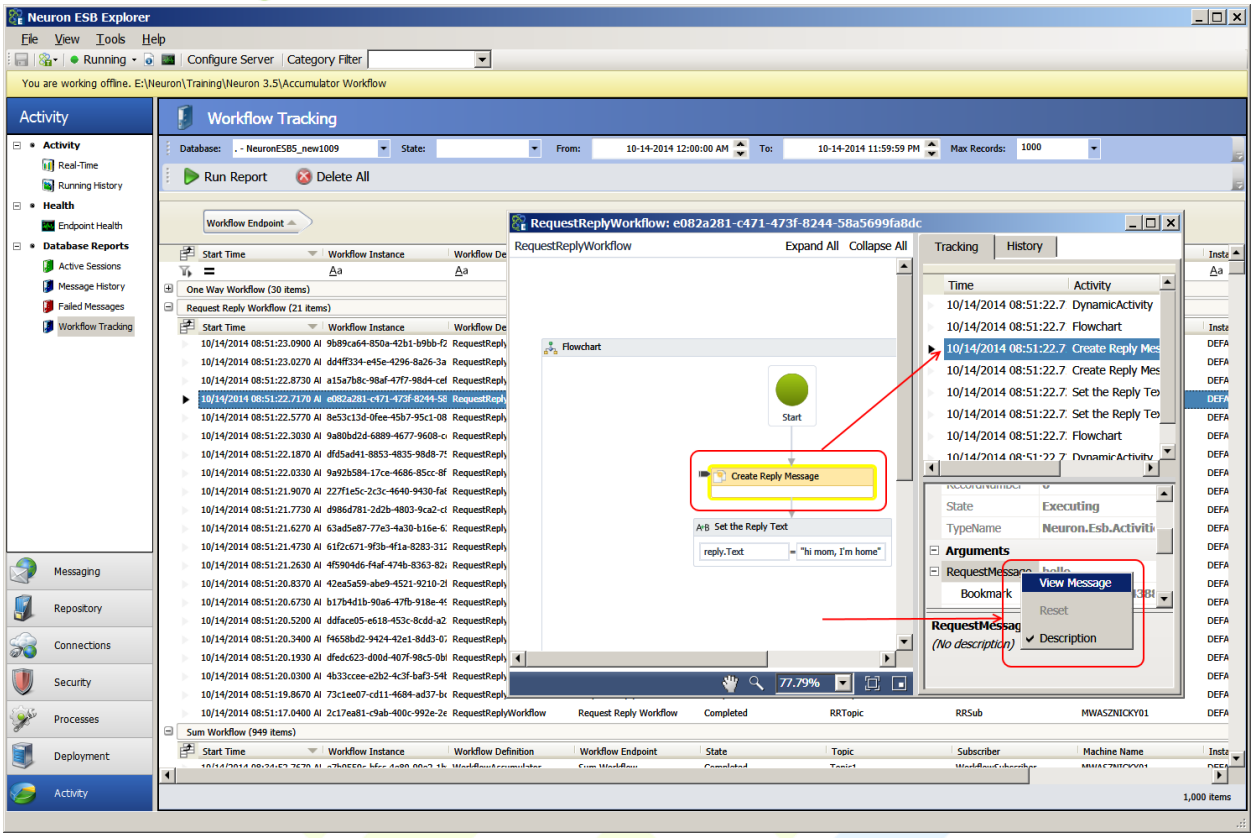

Figure 25 Neuron ESB Workflow Tracking Detail – Users can view the detailed record of any Workflow Instance. This will provide a graphical view of the Workflow Instance, its Tracking information including all state information. Users can navigate the Tracking events to replay the execution of the Workflow Instance.

Lastly, all the rows in the current view can be exported to Excel by right clicking the grid and selecting "Export to Excel..." from the context menu.

Workflow Control and Monitoring

Neuron ESB 3.5 provides Workflow control and monitoring through the new Workflow Tracking, Neuron ESB Endpoint Health and WMI Performance Counters. Workflow Tracking provides users with control commands such as Start, Suspend, Cancel, and Abort or Delete that can be used against any selected Workflow instance or group of Workflow Instances. Control commands for Workflow instances are context sensitive, depending on their current state. For example, a Workflow Instance that is in an unloaded or suspended state can be Started, Cancelled or Aborted. An Aborted, Completed or Cancelled Workflow Instance can be deleted while a Started one can be suspended. Control commands are accessed through the right click context menu of the Workflow Tracking grid (as shown in the figure below).

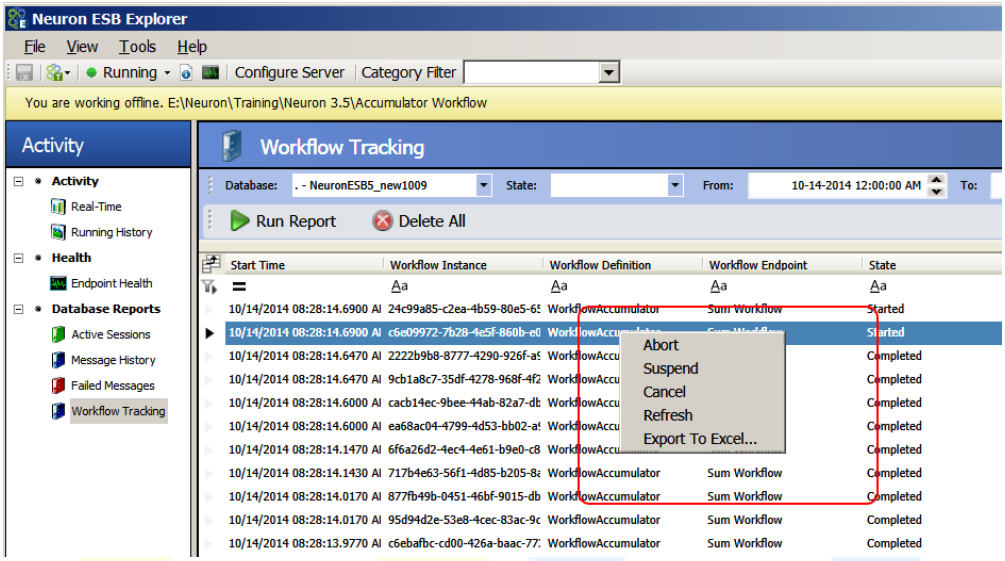

Figure 25 Neuron ESB Workflow Tracking Control Commands – Accessible through the right click context menu. Users can control the execution state of any Workflow instance.

While Workflow Tracking provides detailed information for each Workflow instance, both executing and completed, a summary view of real time Workflow execution activity can be viewed through the Neuron ESB Endpoint Health interface.

Neuron ESB Endpoint Health has been refactored to accommodate real time activity monitoring for the Workflow hosting environments (Availability Groups) as well as the Workflow Endpoints that they host. The new user interface provides users the ability to group and sort, and now lists all machines in the selected deployment group. The Rate measurement has also been changed. In previous versions of Neuron ESB, the Rate column was calculated based on the last time the Neuron ESB runtime was started. However, this meant that the value reflected could become less useful over time if there was not a steady continuous stream of processing. The Rate is now calculated based on the refresh window (default is 15 seconds which can be changed in Zone settings). Effectively we calculate a new rate every 15 seconds based on what was processed in that window.

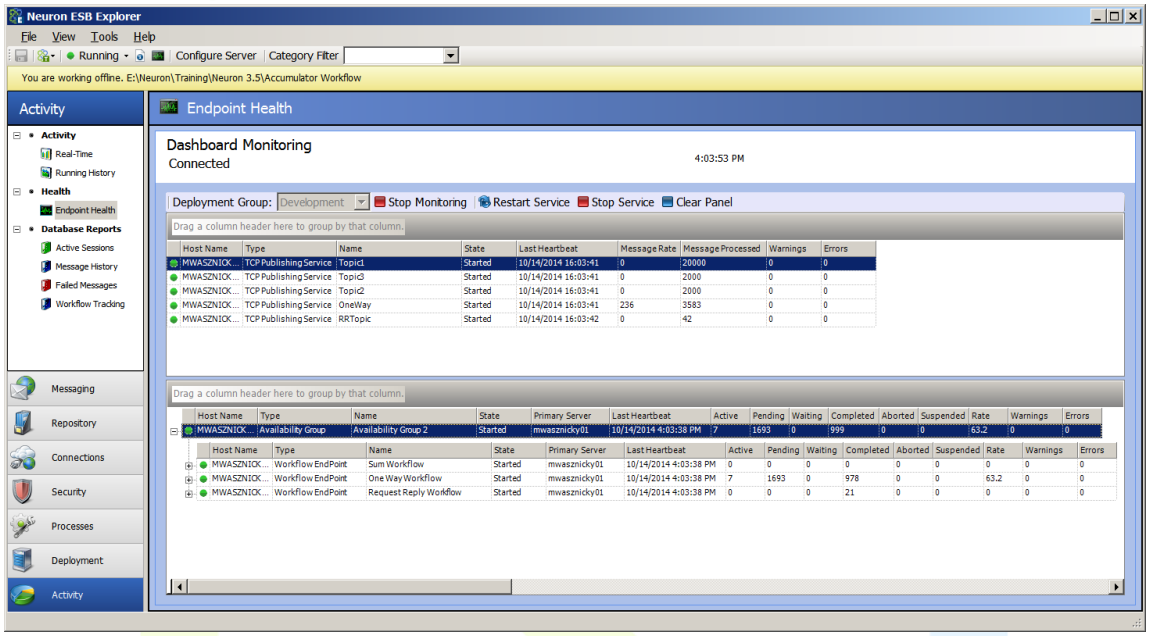

Figure 26 Neuron ESB Endpoint Health – Real time monitoring of both Neuron ESB Messaging and Workflow. Availability Groups (child process) and their Workflow Endpoints can be stopped and started from here. Real time activity of Workflows can be monitored.

The Neuron ESB Endpoint Health interface has a scalable horizontal divider, separating Neuron ESB Messaging entities such as Topics, Adapter and Service Endpoints from Workflow entities such as Availability Groups and their associated Workflow Endpoints. A context menu is exposed at the entity level that allows users to restart any selected entity.

Lastly, two new WMI Performance Counter groups are introduced with Workflow; “Neuron AvailabilityGroups” and “Neuron Workflow Endpoints”. Although “Neuron AvailabilityGroups” only exposes counters to track Errors and Warnings, “Neuron Workflow Endpoints” 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.

Neuron ESB WMI Performance Counters Installation

The WMI Counters for Workflow are created by the Setup.exe installer. This feature is represented on the Feature Install page of the installation wizard by the "ESB Service Management Objects" and is disabled (unchecked) by default. Neuron ESB Workflow, specifically the ability to monitor Workflow activity within Endpoint Health, requires that this feature be selected and installed.

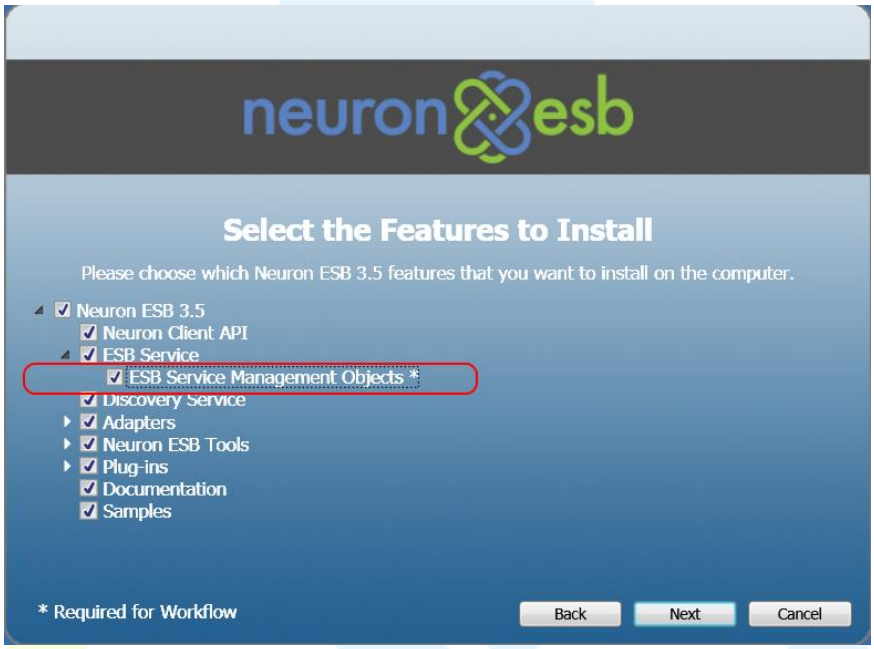

Figure 27 Neuron ESB Setup.exe – Feature selection page of the Neuron ESB installation program. ESB Service Management Objects feature controls the installation of all Neuron ESB WMI Performance counters.

Workflow Samples

Neuron ESB 3.5 ships several new Workflow samples accessible through the Neuron ESB Explorer's Sample Browser, shown in the figure below:

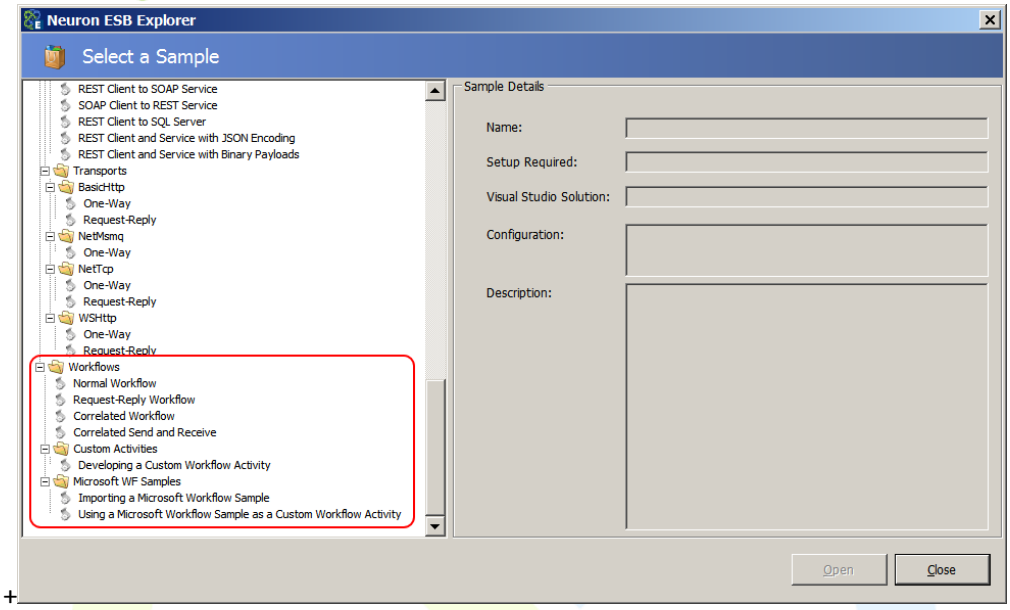

Figure 28 Neuron ESB Sample Browser – Accessible through the Neuron ESB Explorer's View->Samples menu. All new Workflow samples are encircled in red.

There are many Workflow Foundation (WF) samples available on the Microsoft Developer Network. Since Neuron ESB 3.5 hosts its own Workflow Foundation runtime, many samples from Microsoft can be used in Neuron ESB with little or no modifications.

Sample workflows can be imported into Neuron ESB Explorer. This has the advantage of being able to reuse existing workflows and allows users to modify the workflow in the Neuron ESB Workflow Designer.

Note: The Microsoft Windows Workflow (WF) Samples can be downloaded and installed from the download link found on the Windows Workflow Samples page.

The new Neuron ESB Workflow samples that are included in 3.5:

Developing a Custom Workflow Activity

This sample demonstrates how to create a custom workflow activity that generates lorem ipsum text. To test, compile the solution, copy the assembly to the \Program Files\Neuron ESB v3\DEFAULT\Workflows folder and restart the Neuron Explorer. Includes: Visual Studio solution and documentation.

Importing a Microsoft Workflow Sample

This sample demonstrates how to import the *Switch* activity that ships with the Microsoft samples into Neuron ESB Explorer. This sample contains only documentation and requires that you install the Windows Workflow (WF) Samples from the Microsoft site.

Microsoft Workflow Sample as a Custom Workflow Activity

This sample demonstrates how to use the *FlowChartWithFaultHandling* activity that ships with the Microsoft samples into Neuron ESB Explorer. This sample contains only documentation and requires that you install the Windows Workflow (WF) Samples from the Microsoft site.

Normal Workflow

This sample shows a simple Normal workflow that logs the text of the incoming ESBMessage as well as the Workflow Instance ID. A Normal workflow is executed once for each message sent to the topic associated with the Workflow Endpoint.

Request-Reply Workflow

This sample shows a Request-Reply workflow that accepts a promotion code on the text property of the incoming ESBMessage and replies with a new ESBMessage where the text property contains the discount corresponding to the promotion code. A Request-Reply workflow is executed once for each message sent to the topic associated with the Workflow Endpoint.

Correlated Workflow

This sample shows a Correlated Workflow that joins order records after a Process has split them into separate messages. A Correlated-type workflow is executed once for a unique set of messages determined by the correlation set properties configured on the Workflow Endpoint. This is sometimes referred to as a Singleton pattern. In this sample, orders are assigned to batches and the batch ID is used to initialize the correlation set. All orders in batch 'B001' are handled by one instance of the workflow while all orders in batch 'B002' are handled by another workflow instance.

Correlated Send and Receive

This sample shows an order message sent to a normal-type workflow that then publishes it to another topic for further processing. When that processing completes the order is sent back to the existing instance of the workflow that published it. Routing to the original workflow is done by setting the correlation set on the Publish Message Activity to the order ID.

Composition using Adapter and Service Endpoints

As referenced earlier in this document, Neuron ESB 3.5 provides an Adapter and Service Endpoint Process Step that can be used in the Neuron ESB Business Process designer for creating real-time service composition solutions by aggregating existing services and application endpoints into more innovative business capabilities that can be accessed throughout an organization.

The advantage of the Adapter and Service Endpoint Process Steps is that they eliminate much of the overhead traditionally seen when bus or other messaging architectures are incorporated in service composition solutions where request/response type of message patterns are predominately employed.

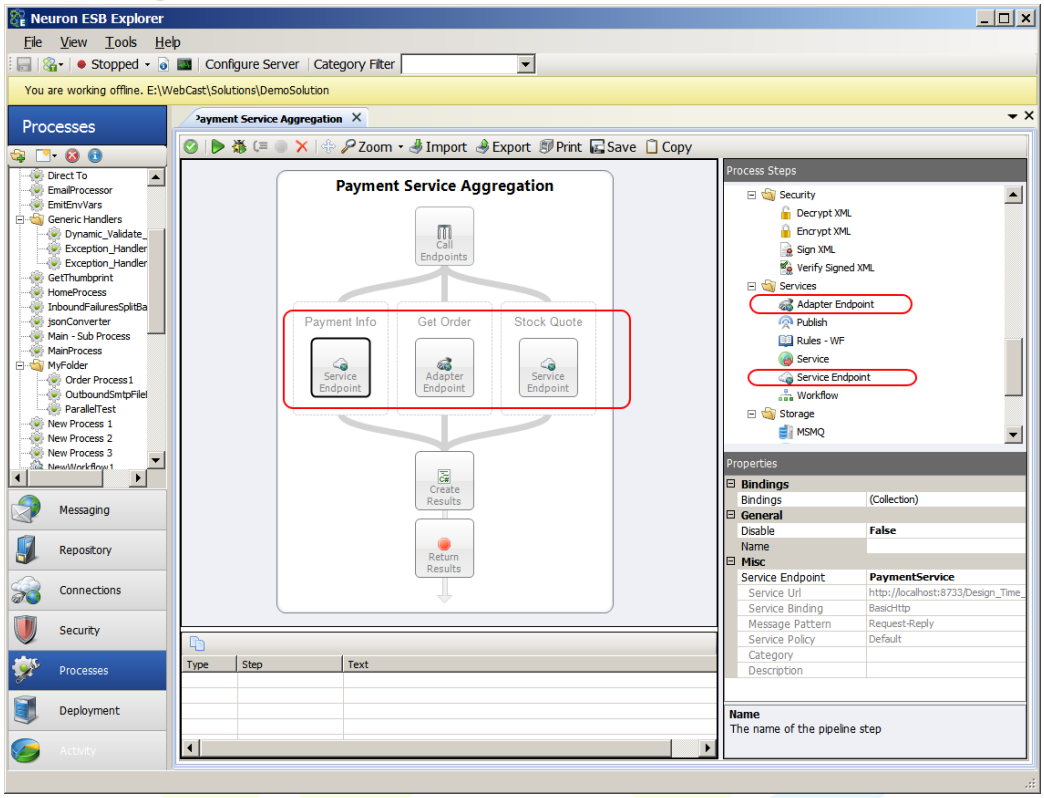

Figure 29 Neuron ESB Service Composition – Service composition example in the Business Designer using the Parallel process step in conjunction with the new Adapter and Service Endpoint process steps encircled in Red.

The Adapter and Service Endpoint Process Steps allow users to call any configured Adapter Endpoint or Service Endpoint (i.e. Service Connector) directly, without the need to publish the message to a Topic. This works with either Request/Response or Multicast/Datagram types of messaging patterns.

Using these new process steps does not preclude the ability for users to dynamically specify which service to call at runtime. For example, the Service Endpoint Process Step allows users to dynamically set the name of the Service Connector to call at runtime by simply setting the Service property of the ESB Message Header, using either the Set Property Process Step or any of the language Process Steps such as the C# Process Step ex:

```
context.Data.Header.Service = "Payment Service";
```

WS-Discovery enabled runtime, parties and endpoints

Neuron ESB 3.5 now supports Web Services Dynamic Discovery (WS-Discovery) for its own Discovery service, Neuron ESB runtime instances, Client Connectors (Neuron ESB hosted services) and remotely hosted Neuron ESB Parties (i.e. client API). WS-Discovery is a powerful management standard and technical specification that defines a multicast discovery protocol to locate services on a local network. It operates over TCP and UDP port 3702 and uses IP multicast address 239.255.255.250. As the name suggests, the actual communication between nodes is done using web services standards, notably SOAP-over-UDP.

By default, the WS-Discovery protocol is enabled for the Discovery service, Neuron ESB runtime instances, Client Connectors (Neuron ESB hosted services), whereas remotely hosted Neuron ESB Parties must be specifically enabled through an app.config setting (see tables below).

Neuron ESB Discovery API – App.config			
	Key	Value	Description
	WSDiscoveryManagedPort	9021	Represents the TCP port used for Managed probe requests using the Neuron ESB Discovery API
	WSDiscoveryEnabled	true/false (default true)	Determines if a job will run to clean up stale endpoint registrations
	WSDiscoveryApplicationScope	default	Defines scope of all queries.

Client API – App.config (affects remotely hosted Neuron ESB Parties)			
	Key	Value	Description
	WSDiscoveryRemoteEndpoint	true/false (default false)	Determines whether the remote party can accept and respond to probe requests. Must be true if WSDiscoveryEnabled is set to true
	WSDiscoveryEnabled	true/false (default true)	Determines whether or not the remotely hosted API will broadcast WS-Discovery “hello”/”goodbye” messages
	WSDiscoveryApplicationScope	default	Defines scope of all queries.
	WSDiscoveryAnnouncementInterval	00:02:00	Only effective if WSDiscoveryEnabled = true. Determines broadcast interval

Neuron ESB Runtime Instance – EsbService.exe.config (affects configuration service and client connectors)			
	Key	Value	Description
	WSDiscoveryBroadcast	true/false (default true)	Determines whether or not the remotely hosted API will broadcast WS-Discovery “hello”/”goodbye” messages
	WSDiscoveryEnabled	true/false (default true)	Determines whether the services can accept and respond to probe requests.
	WSDiscoveryApplicationScope	default	Defines scope of all queries.
	WSDiscoveryAnnouncementInterval	00:00:10	Only effective if WSDiscoveryEnabled = true. Determines broadcast interval

Neuron ESB Discovery Service – DiscoveryService.exe.config			
	Key	Value	Description
	WSDiscoveryManagedPort	9021	Represents the TCP port used for Managed probe requests using the Neuron ESB Discovery API
	WSDiscoveryEnabled	true/false (default true)	Determines whether the discovery service will broadcasts and receive unicast probe requests
	WSDiscoveryApplicationScope	default	Defines scope of all queries.
	WSDiscoveryAnnouncementInterval	00:00:10	Only effective if WSDiscoveryEnabled = true. Determines broadcast interval
	WSDiscoveryManagedIpAddress	“”	Allows user to provide an IP address that will be broadcast to discovery clients to be used as unicast address for probes. A way to override the

			default use of netbios names.
	WSDiscoveryManagedUseHostName	true/false (default false)	Determines whether the DNS host name or local host name will be used as the address for WS-Discovery proxy. Default is local host name.

Neuron ESB implements Ad Hoc and Managed Mode WS-Discovery and provides an API that allows users to query (probe) for endpoint information as well as monitor WS-Discovery broadcast events. Managed mode probe requests are sent locally to the Neuron ESB Discovery Service which also serves as a WS-Discovery proxy for Neuron ESB enabled services and endpoints. Managed mode requests are made over TCP port 9021 by default. This port can be changed by modifying the "***WSDiscoveryManagedPort***" setting in both the application (that is using the Neuron ESB Discovery API) app.config file as well as the Neuron ESB Discovery Service app.config (i.e. DiscoveryService.exe.config).

WS-Discovery broadcasts and probe results have additional metadata associated with them, specific to the endpoint. This metadata is exposed as typed information and can be retrieved using the Neuron ESB Discovery API. The Neuron ESB Discovery API namespace is *Neuron.Esb.Discovery* and is represented by the *DiscoveryClient* class. The Neuron ESB Discovery API sample below demonstrates several ways to interact with Neuron ESB services using the WS-Discovery protocol. This same API is built into the Neuron ESB Test Client and the Neuron ESB Explorer, allowing users to auto discover Neuron ESB runtime instances on the network in which to connect to.

```
private static void OfflineDiscovery(object sender, AnnouncementEventArgs e)
{
}
private static void OnlineDiscovery(object sender, AnnouncementEventArgs e)
{
}
static void Main(string[] args)
{
    using (var discovery = new Neuron.Esb.Discovery.DiscoveryClient())
    {

        /// these events capture all online/offline WS-Discovery broadcasts made by all
        /// Neuron ESB Runtime services such as Configuration and Peer Resolver.
        discovery.OfflineNeuronWsDiscovery += discovery_OfflineNeuronWsDiscovery;
        discovery.OnlineNeuronWsDiscovery += discovery_OnlineNeuronWsDiscovery;

        // this sets all probes to use Managed mode versus Ad Hoc. Managed mode uses unicast probe
        // requests over TCP
        discovery.DiscoveryMode = Neuron.Esb.Discovery.DiscoveryModeEnum.Managed;

        /// FindNeuronEndpoints() is a synchronous call that returns a
        /// List<System.ServiceModel.Discovery.EndpointDiscoveryMetadata>
        /// Each EndpointDiscoveryMetadata object contains an Extensions element containing Neuron
        /// ESB metadata specific for each type of Neuron ESB endpoint
        var resultsConfig = discovery.FindNeuronEndpoints(NeuronDisoveryTypeEnum.ServiceEndpoint,
              3);
        foreach (var endpoint in resultsConfig)
```

```
            Console.WriteLine("Endpoint address found :" + endpoint.Address);

        resultsConfig = discovery.FindNeuronEndpoints(NeuronDisoveryTypeEnum.ConfigurationService,
                    3);
        foreach (var endpoint in resultsConfig)
            Console.WriteLine("Endpoint address found :" + endpoint.Address);

        /// these events capture the results of the Asynchronous FindRuntimeService call. They
        /// provide access to the
        /// EndpointDiscoveryMetadata as well as the Extensions as a strongly typed Neuron ESB
        /// object. Each type of
        /// Asynchronous call (one for Runtime Services, Service Endpoints and Remote Endpoints)
        /// has its respective 'Completed' and 'Progress' event
        discovery.FindRuntimeServiceCompletedNeuronWsDiscovery +=
                                                    discovery_FindCompletedNeuronWsDiscovery;
        discovery.FindRuntimeServiceProgressNeuronWsDiscovery +=
                                                    discovery_FindProgressNeuronWsDiscovery;

        /// FindNeuronRuntimeServiceAsync() is an asynchronous call. Its
        /// FindRuntimeServiceProgressNeuronWsDiscovery() event
        /// provides access to the found endpoints as they are found on the network, while its
        /// FindRuntimeServiceCompletedNeuronWsDiscovery()
        /// event provides access to the final entire collection of endpoints found at the finish
        /// of the duration of the call. by default, the duration is 20 seconds, but can be
        /// controlled through an argument.
        discovery.FindNeuronRuntimeServiceAsync(NeuronDisoveryTypeEnum.ConfigurationService,20);
        Console.WriteLine("Press Enter to continue past waiting for async");
    }

}
static void discovery_FindProgressNeuronWsDiscovery(object o, DiscoveryArgs e)
{
    Console.WriteLine("FOUND IN PROGRESS:" + e.Address);
}

static void discovery_FindCompletedNeuronWsDiscovery(object
                                                    o,DiscoveryFindRuntimeServiceCompletedArgs e)
{
    Console.WriteLine("COMPLETED:");
    foreach (var ep in e.DiscoveryCollection)
    {
        if(ep.Error != null)
            Console.WriteLine("ERROR RETURNED: " + ep.Error.ToString());
        else
            Console.WriteLine("     ADDRESS:" + ep.Address);
    }
}

static void discovery_OnlineNeuronWsDiscovery(object o, DiscoveryArgs e)
{
    Console.WriteLine("ON LINE:");
    Console.WriteLine("   Type: " + e.Type.ToString());
    Console.WriteLine("   Address: " + e.Address.ToString());
    Console.WriteLine("   Configuration: " + e.Configuration.ToString());
    Console.WriteLine("   Instance: " + e.InstanceName.ToString());
    Console.WriteLine("   ActiveDeploymentGroup: " + e.ActiveDeploymentGroup.ToString());

    if (e.Type == NeuronDisoveryTypeEnum.ConfigurationService)
        onlineAddress = e.Address.ToString();
}

static void discovery_OfflineNeuronWsDiscovery(object o, DiscoveryArgs e)
{
    Console.WriteLine("OFF LINE:");
    Console.WriteLine("   Type: " + e.Type.ToString());
    Console.WriteLine("   Address: " + e.Address.ToString());
    Console.WriteLine("   Configuration: " + e.Configuration.ToString());
    Console.WriteLine("   Instance: " + e.InstanceName.ToString());
```

```
            Console.WriteLine("    ActiveDeploymentGroup: " + e.ActiveDeploymentGroup.ToString());
        }
```

REST and WMI enabled Endpoint Health Monitoring

REST Interfaces

Neuron ESB 3.5 exposes much of its monitoring and server management functions through REST based interfaces. These interfaces can be used to extend and build custom monitoring and management solutions for Neuron ESB Deployments. The Neuron ESB 3.5 REST interfaces are hosted by the Neuron ESB Discovery service (DiscoveryService.exe) which is installed with the Neuron ESB Server Runtime. The default port (51002) for the REST interfaces is configurable through the Neuron ESB Discovery service's app.config. The default installation location of the Neuron ESB Discovery service is : "*C:\Program Files (x86)\Neudesic\Neuron ESB v3*".

The Neuron ESB REST interface documentation can be found at the default URL, http://localhost:51002/help.

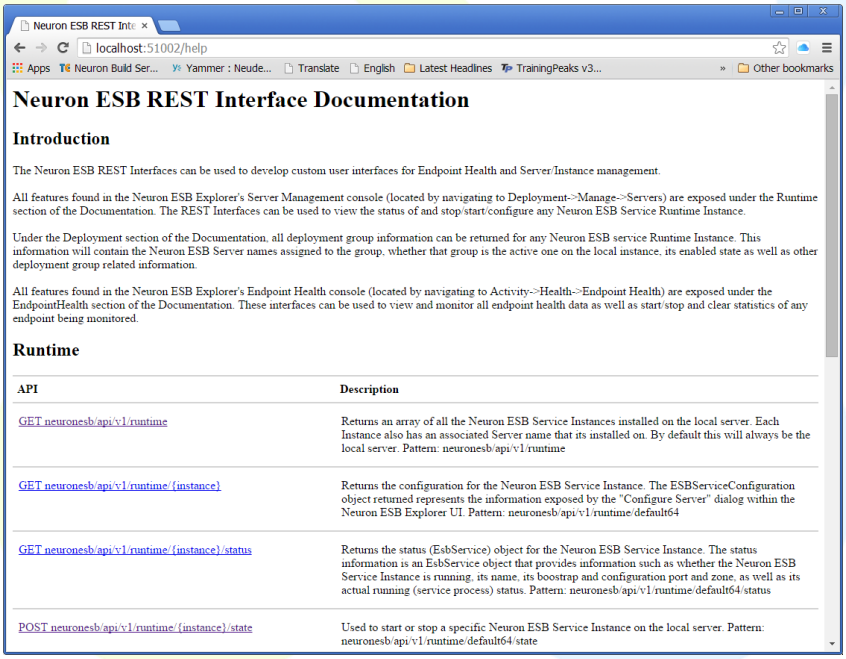

Figure 30 Neuron ESB REST Documentation Web Site – The Neuron ESB REST interface documentation is hosted on the local Neuron ESB server.

The REST interfaces are self-documenting, providing detailed instructions on how to call each specific function and are separated into the following categories:

- Runtime
- Deployment
- Endpoint Health
- Activity

Runtime

All features found in the Neuron ESB Explorer's Server Management console (located by navigating to Deployment->Manage->Servers) are exposed under the Runtime section of the Documentation. The REST Interfaces can be used to view the status of and stop/start/reconfigure any Neuron ESB Service Runtime Instance. Several interfaces are exposed including:

GET

- neuronesb/api/v1/runtime
- neuronesb/api/v1/runtime/{instance}
- neuronesb/api/v1/runtime/{instance}/status

POST

- neuronesb/api/v1/runtime/{instance}/state
- neuronesb/api/v1/runtime/{instance}/config

Using these interfaces, users can find all instances of the Neuron ESB Runtime on a specific server, retrieve their respective solution configuration, and then use the Deployment REST interface to determine exactly what servers the solution is deployed to and how it is configured across the various environments.

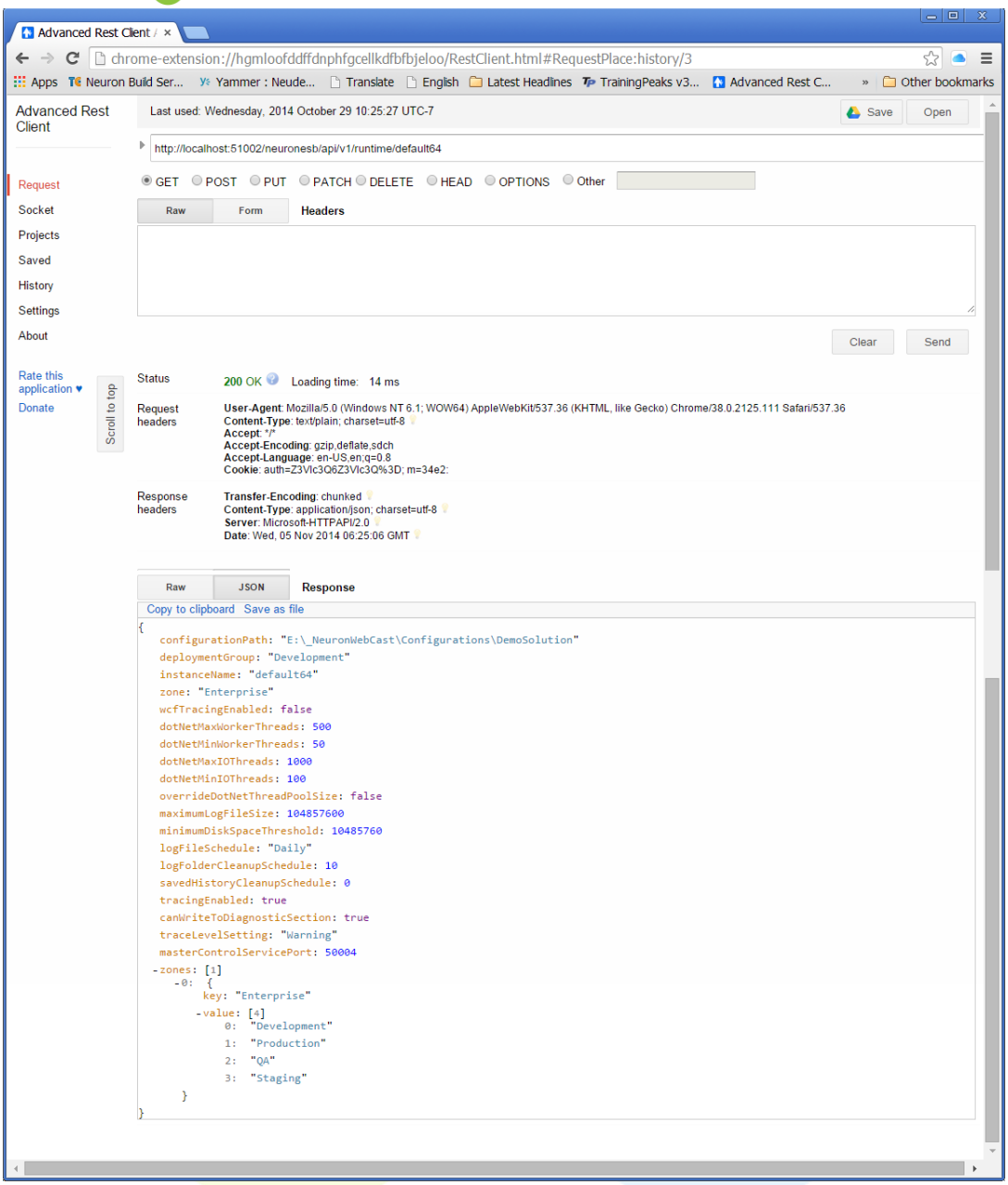

Figure 31 Neuron ESB REST Runtime – Demonstrates calling one of the Runtime REST interfaces from the Google REST client to retrieve the configuration for a specific Neuron ESB Runtime instance.

Deployment

All deployment group information can be returned for any Neuron ESB service Runtime Instance. This information will contain the Neuron ESB Server names assigned to the group, whether that group is the active one on the local instance, its enabled state as well as other deployment group related information.

A single interface is exposed:

GET

- neuronesb/api/v1/deployment/{instance}

Endpoint Health

All features found in the Neuron ESB Explorer's Endpoint Health console (located by navigating to Activity->Health->Endpoint Health) are exposed under the Endpoint Health section of the Documentation. These interfaces can be used to view and monitor all endpoint health data as well as start/stop and clear statistics of any endpoint being monitored such as a Topic, Adapter Endpoint, Service Endpoint, Availability Group and Workflow Endpoint.

Two interfaces are exposed:

GET

- neuronesb/api/v1/endpointhealth/{instance}

POST

- neuronesb/api/v1/endpointhealth/{instance}/{id}/state

An example of using these together would be to first retrieve the ID of the endpoint to Stop or Start using the GET interface. Once the ID is retrieved it could be passed to the PUT interface to either Stop or Start the endpoint.

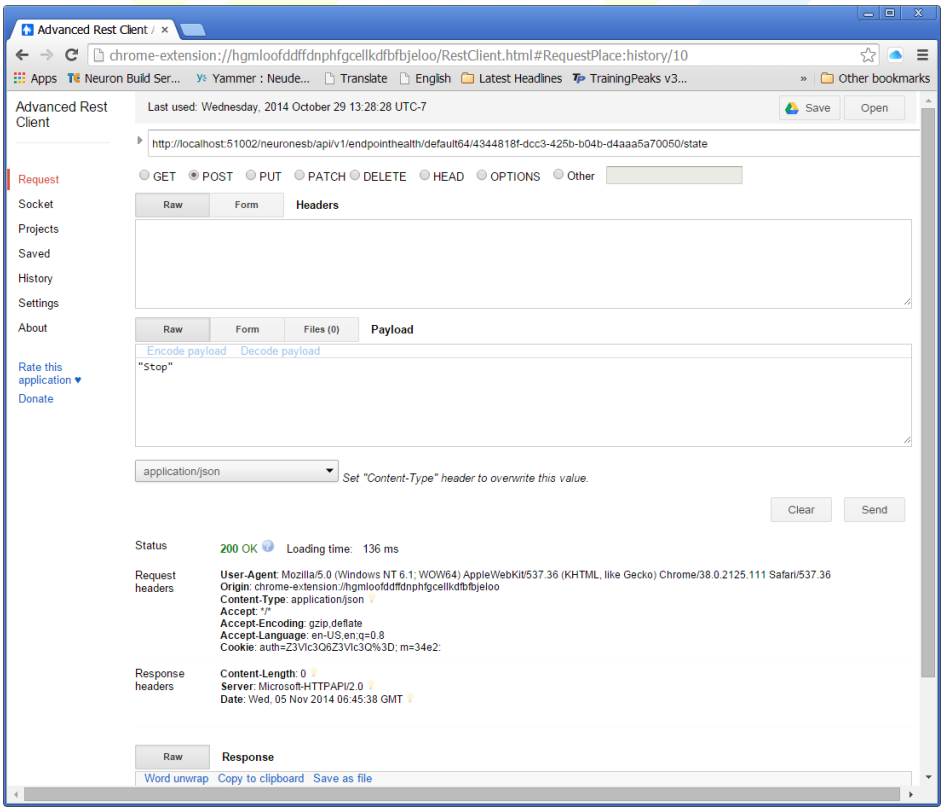

Figure 32 Neuron ESB REST Endpoint Health – Demonstrates calling one of the Runtime REST interfaces from the Google REST client to stop a specific Topic by using the ID of the Topic.

Activity

The Activity REST interface provides users the ability to query Neuron ESB and retrieve any message logged to the Failed Message Audit table by its message id.

A single interface is exposed:

GET

- neuronesb/api/v1/Activity/{instance}/{messageid}

For example, a user could use Windows Management Instrumentation (WMI) to monitor Neuron ESB for Failed Messages and, alternatively use REST to retrieve the message body if the message was larger than 1MB in size. Once the message body is retrieved, it could be published back to Neuron ESB using the client API for further processing or to generate notifications.

The first step: setup the monitoring of the WMI event:

```
static void Main()
{
    try
    {
        var managementScope = new ManagementScope("\\\\.\\root\\Neudesic_ESB_v0");
        managementScope.Connect();

        var eventQuery = new WqlEventQuery("FailedMessageEvent");

        var watcher = new ManagementEventWatcher(managementScope, eventQuery);
        watcher.EventArrived += FailedMessage;
        watcher.Start();

        Console.WriteLine("Listening for events. Press Enter to exit.");
        Console.ReadLine();

        watcher.Stop();
    }
    catch (Exception ex)
    {
        Console.Error.WriteLine(ex);
    }
}
```

The second step: capture the *FailedMessage* event:

```
static void FailedMessage(object sender, EventArrivedEventArgs e)
{

    string action = e.NewEvent["Action"] as string;
    DateTime failureDate = ManagementDateTimeConverter.ToDateTime(e.NewEvent["ActionDate"]
                           as string);
    string failureType = e.NewEvent["FailureType"] as string;
    string failureDetail = e.NewEvent["FailureDetail"] as string;
    string messageBody = e.NewEvent["Message"] as string;
    string session = e.NewEvent["Session"] as string;
    string topic = e.NewEvent["Topic"] as string;
    string party = e.NewEvent["PartyId"] as string;
    string instanceName = e.NewEvent["EsbInstanceName"] as string;
    string messageId = e.NewEvent["MessageId"] as string;
    bool largeMsg = (bool)e.NewEvent["LargeMessage"];

    Console.WriteLine(messageBody);

    var message = string.Format(
```

```
            CultureInfo.InvariantCulture,
            "Failed message on instance = {0} topic = {1} party = {2} DateTime = {3}
             Failure Type = {4} Failure Detail = {5} -- Message ID = {6}",
            instanceName,
            topic,
            party,
            failureDate.ToString(),
            failureType,
            failureDetail,
            messageId);

        Console.WriteLine(message);
    }
```

The last step: include the conditional logic to call the Neuron ESB Rest interface if the message body was too large for the event to return:

```
    static void FailedMessage(object sender, EventArrivedEventArgs e)
    {

        string action = e.NewEvent["Action"] as string;
        DateTime failureDate = ManagementDateTimeConverter.ToDateTime(e.NewEvent["ActionDate"]
                               as string);
        string failureType = e.NewEvent["FailureType"] as string;
        string failureDetail = e.NewEvent["FailureDetail"] as string;
        string messageBody = e.NewEvent["Message"] as string;
        string session = e.NewEvent["Session"] as string;
        string topic = e.NewEvent["Topic"] as string;
        string party = e.NewEvent["PartyId"] as string;
        string instanceName = e.NewEvent["EsbInstanceName"] as string;
        string messageId = e.NewEvent["MessageId"] as string;
        bool largeMsg = (bool)e.NewEvent["LargeMessage"];

        if (largeMsg)
        {
            var url = string.Format(CultureInfo.InvariantCulture,
                "http://localhost:51002/neuronesb/api/v1/activity/{0}/{1}",
                instanceName, messageId);

            HttpWebRequest webRequest = (HttpWebRequest)WebRequest.Create(url);
            webRequest.Method = "GET";
            HttpWebResponse webResponse = (HttpWebResponse)webRequest.GetResponse();
            using (Stream responseStream = webResponse.GetResponseStream())
            {
                using (StreamReader sr = new StreamReader(responseStream))
                {
                    messageBody = sr.ReadToEnd();
                }
            }
        }

        Console.WriteLine(messageBody);

        var message = string.Format(
            CultureInfo.InvariantCulture,
            "Failed message on instance = {0} topic = {1} party = {2} DateTime = {3}
             Failure Type = {4} Failure Detail = {5} -- Message ID = {6}",
            instanceName,
            topic,
            party,
            failureDate.ToString(),
            failureType,
            failureDetail,
            messageId);

        Console.WriteLine(message);
    }
```

Windows Management Instrumentation (WMI)

Neuron ESB 3.5 introduces a new WMI event that allows users to monitor state changes that occur in any Adapter or Service Endpoint (Client or Service Connector). Historically, users would monitor these state changes through the Neuron ESB Explorer's Endpoint Health. However, WMI allows users to build extended monitoring solutions without the need to reference Neuron ESB specific assemblies.

For example, using the WMI event, a solution could be developed to detect when an endpoint goes offline. When it does, a notification could be sent or, the Neuron ESB REST interface could be used to automate the restart of the endpoint.

The following states are reported by the WMI event. These are the same state events reported within Endpoint Health.

```
public enum ServiceState
{
    StateUninitialized = 0,
    StateStarting = 1,
    StateStarted = 2,
    StateStopping = 3,
    StateStopped = 4,
    StateFailed = 5,
    StatePaused = 6,
    StateDisabled = 7,
    StateOutOfService = 8,
}
```

Initializing and monitoring the event (*EndpointStateChangeEvent*) is similar to that of the Neuron ESB *FailedMessageEvent* WMI event:

```
static void Main()
{
    try
    {
        var managementScope = new ManagementScope("\\\\.\\root\\Neudesic_ESB_v0");
        managementScope.Connect();

        var eventEndpointQuery = new WqlEventQuery("EndpointStateChangeEvent");
        var watcher = new ManagementEventWatcher(managementScope, eventEndpointQuery);
        watcher.EventArrived += EndpointChanged;
        watcher.Start();

        Console.WriteLine("Listening for events. Press Enter to exit.");
        Console.ReadLine();

        watcher.Stop();
    }
    catch (Exception ex)
    {
        Console.Error.WriteLine(ex);
    }
}
```

Capturing the *EndpointChanged* event:

```
static void EndpointChanged(object sender, EventArrivedEventArgs e)
{

    string zone = e.NewEvent["Zone"] as string;
    DateTime eventDate = ManagementDateTimeConverter.ToDateTime(e.NewEvent["Datetime"] as string);
    string type = e.NewEvent["Type"] as string;
```

```
        string name = e.NewEvent["Name"] as string;
        string state = e.NewEvent["State"] as string;
        string hostName = e.NewEvent["Hostname"] as string;
        string instanceName = e.NewEvent["EsbInstanceName"] as string;
        string application = e.NewEvent["Application"] as string;
        string deploymentGroup = e.NewEvent["DeploymentGroup"] as string;
        string endpointId = e.NewEvent["Id"] as string;
        string info = e.NewEvent["Message"] as string;

        var message = string.Format(
            CultureInfo.InvariantCulture,
            "Name={0}, Endpoint Type={1}, State={2}, Instance={3}, Zone={4}, Machine={5},
             Application={6} , DeploymentGroup={7}, Info={8}, DateTime={9}",
            name,
            type,
            state,
            instanceName,
            zone,
            hostName,
            application,
            deploymentGroup,
            info,
            eventDate.ToString());

        Console.WriteLine(message);
    }
```

As seen in the example above, a number of properties are exposed by the WMI event, including Type and Message. The Type property can be one of 3 possible values:

- ClientConnector
- ServiceConnector
- AdapterEndpoint

The Message property will only be populated if a failure occurred with the endpoint. In that case, the Message property will contain the entire *System.Exception* message.

Using the Type property, the original WMI query could be modified to only monitor endpoints of a specific type using the Condition property:

```
var eventEndpointQuery = new WqlEventQuery("EndpointStateChangeEvent");
eventEndpointQuery.Condition = "Type = 'AdapterEndpoint'";
```

NetSuite and Dynamics CRM 2013 Online

NetSuite Adapter

This is a new adapter included in the 3.5 release. This subscription adapter supports one-way as well as solicit response mode. Users can perform any operation available in the SuiteTalk Web Services Platform provided by NetSuite. They can send inserts, updates or deletes into NetSuite, or perform gets, getList and search requests against NetSuite. This adapter also supports meta-data harvesting. Users can browse the operations exposed by NetSuite and elect to generate Xml Schemas and sample Xml Messages for the various operations.

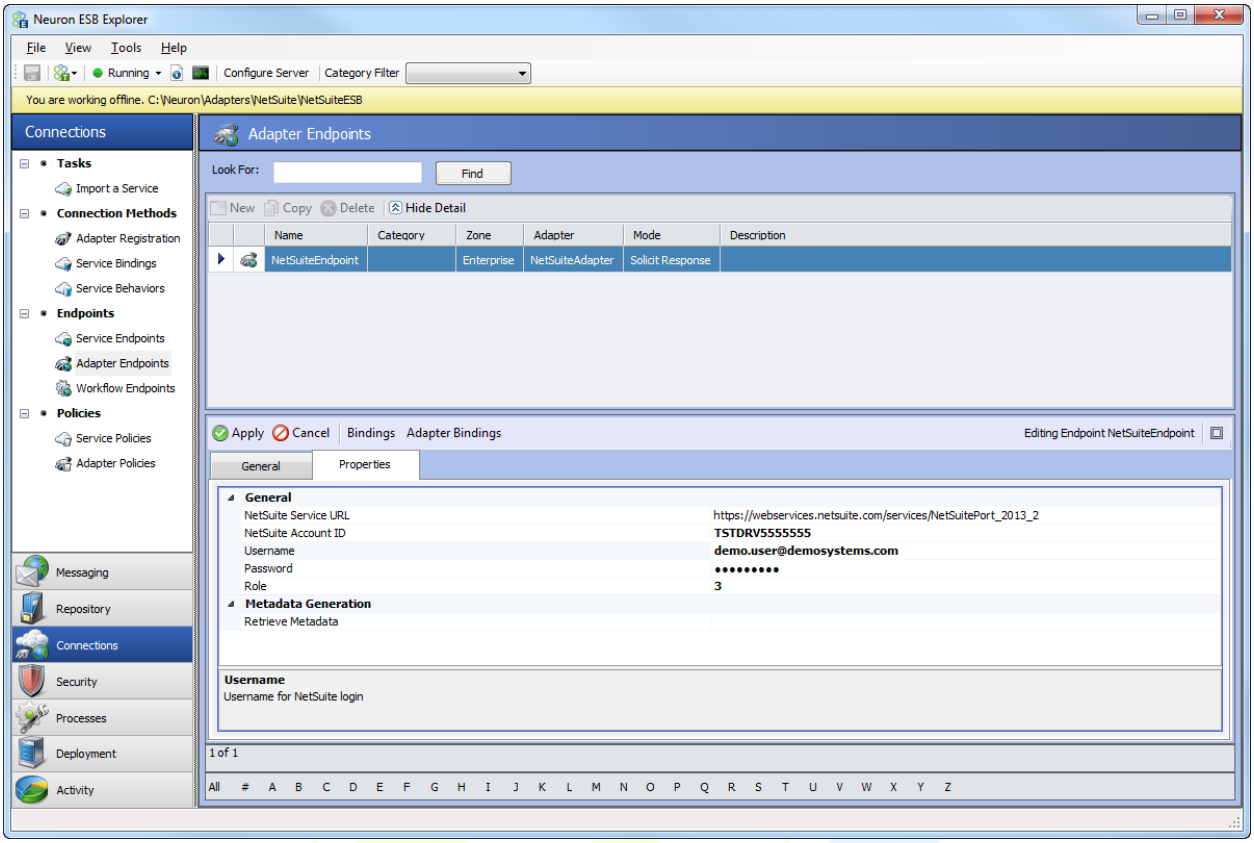

Figure 33 Neuron ESB NetSuite Adapter Endpoint Configuration – Once the NetSuite Adapter is registered, any number of Adapter Endpoints can be configured using the adapter.

The Meta data generation wizard can be accessed through the “Retrieve Metadata” property of the adapter endpoint.

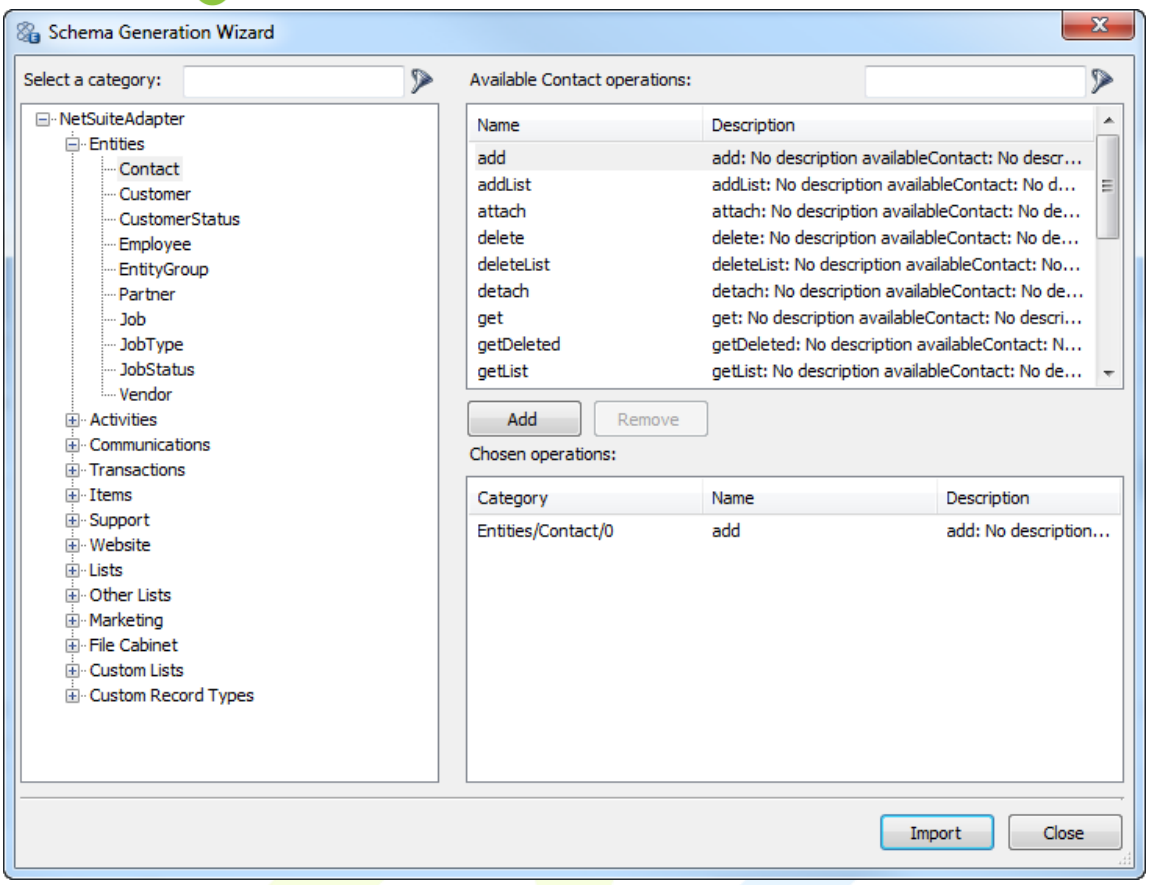

Figure 34 Neuron ESB Meta Data Browser – Users can browse all the available entities and operations exposed by the NetSuite application.

Operations can be selected by clicking the Add button. After operations are selected, the Import button will display the selected operations, allowing users to edit their properties and to optionally choose to generate sample Xml messages. The Finish button will store all the generated Xml Schemas and messages in the Neuron Explorer Repository.

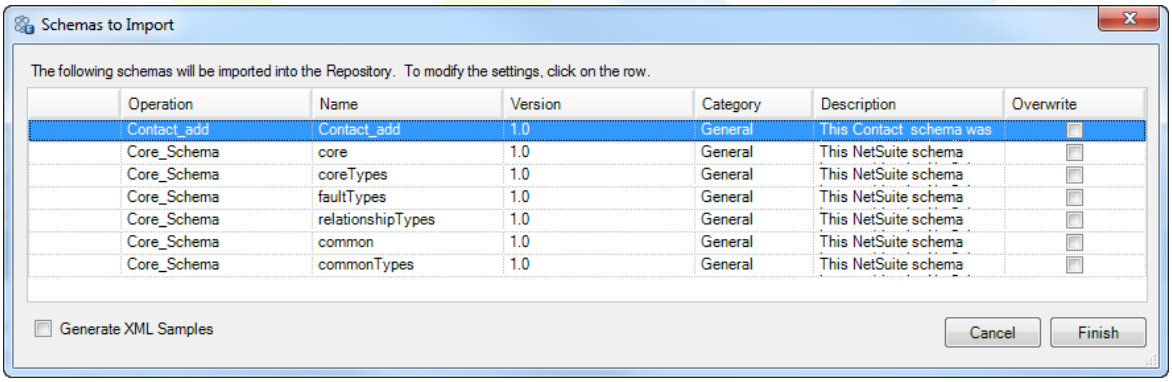

Figure 35 Neuron ESB Schema Import – Users can import NetSuite operation XML Schemas as well as generate sample Xml messages to send to the adapter.

Microsoft Dynamics CRM 2013 Online

The Microsoft Dynamics CRM 2013 Online event based workflow adapter compliments the existing Microsoft Dynamics CRM 2013 subscription adapter and provides similar functionality that exists in the existing Neuron ESB Dynamics CRM 2013 On Premise workflow adapter.

Using Neuron ESB, Microsoft Dynamics CRM 2013 Online administrators can easily capture events through the Dynamics CRM 2013 Process Designer and configure them to be forwarded to Neuron ESB. Processes in Microsoft Dynamics CRM 2013 are based on Windows Workflow Foundation. By leveraging the Microsoft Dynamics CRM 2013 Online process engine, Neuron can provide more options for business rule creation, while offering a greater breadth of event publication options.

The Neuron ESB Microsoft Dynamics CRM 2013 Online workflow adapter extends Microsoft Dynamics CRM 2013, enabling it to send event notifications to Neuron ESB. Some of the features include:

- Send entities to Neuron ESB hosted services.
- Send dynamic entities to Neuron ESB hosted services.
- Send customized XML with data from related entities to Neuron ESB hosted services.
- Execute fetch XML queries and send the results to Neuron ESB hosted services.

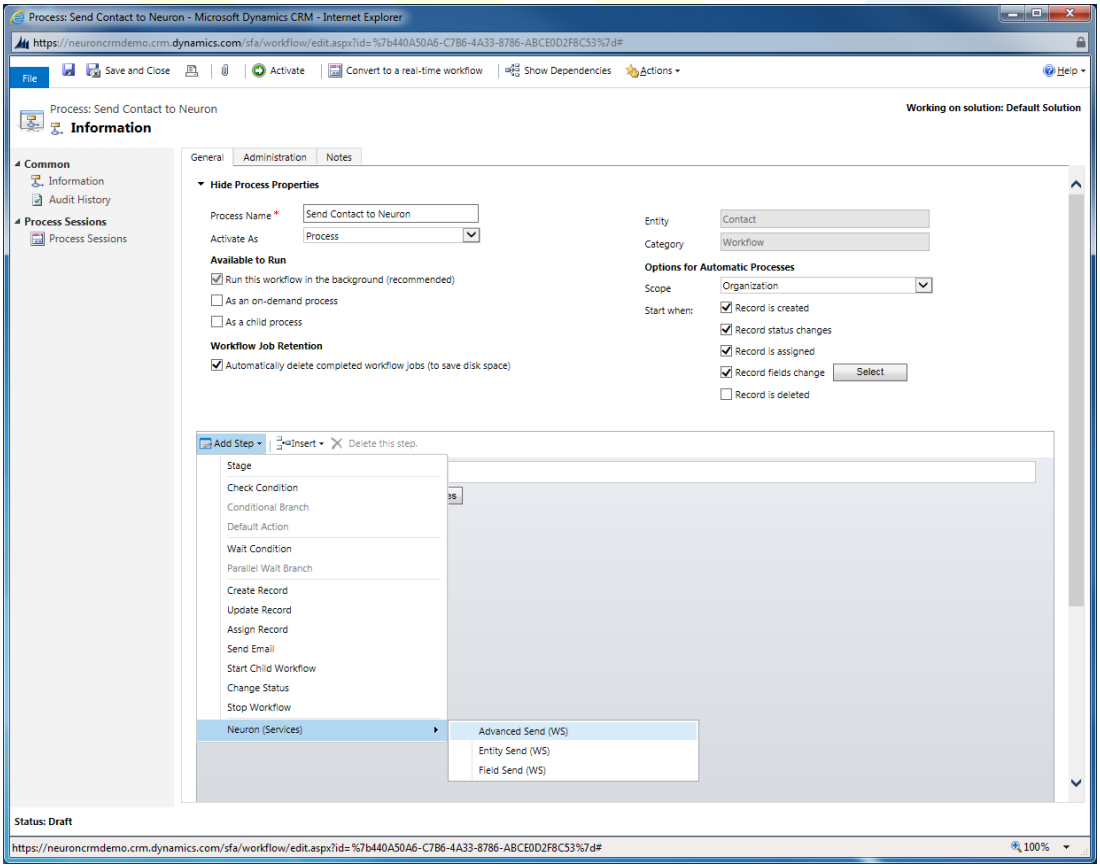

Figure 36 Microsoft Dynamics CRM 2013 Online Process Designer – The Neuron ESB Microsoft Dynamics CRM Online Workflow adapter installs 3 activities that can be used to auto publish entities and related information directly to Neuron ESB hosted services.

The Neuron ESB Microsoft Dynamics CRM 2013 Online workflow adapter must be installed in sandboxed isolation mode on either a Microsoft Dynamics CRM Online instance or Office 365 instance. The plugin can also be used in sandboxed isolation mode in on-premises versions of Microsoft Dynamics CRM 2013.

Neuron ESB Explorer UX

With the release of Neuron ESB 3.5, users no longer are presented with a "Connect" dialog when they first launch the Neuron ESB Explorer. In previous versions (as shown in the image below), users were presented with a dialog where they could either connect to a Neuron ESB runtime instance, open a solution offline, or create a new solution.

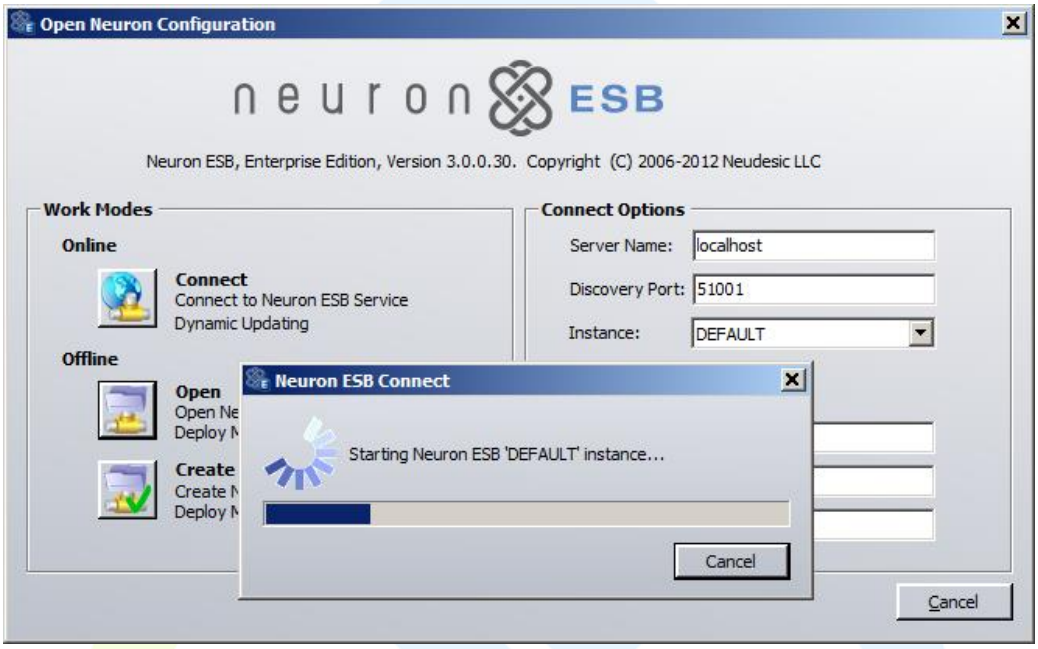

Figure 37 Neuron ESB 3.1 Connect Dialog – In previous versions of Neuron ESB users could be presented with the Connect dialog anytime they needed to either connect to, open or create a new solution.

However, using the Connect dialog was the only way a user could work with any solution. There was no File menu options that allowed users to directly open or create solutions. This could impede productivity for users who needed to switch between several solutions.

In this latest release, when users launch the Neuron ESB Explorer executable (NeuronExplorer.exe), they are presented with the Neuron ESB Explorer IDE rather than the Connect dialog as in previous versions. The Neuron ESB Explorer IDE includes new File menu options such as "*New*", "*Open*" and "*Connect…*", effectively replacing the functionality of the "*Open*", "*Connect*" and "*Create*" buttons that existed in the pre-3.5 Connect dialog.

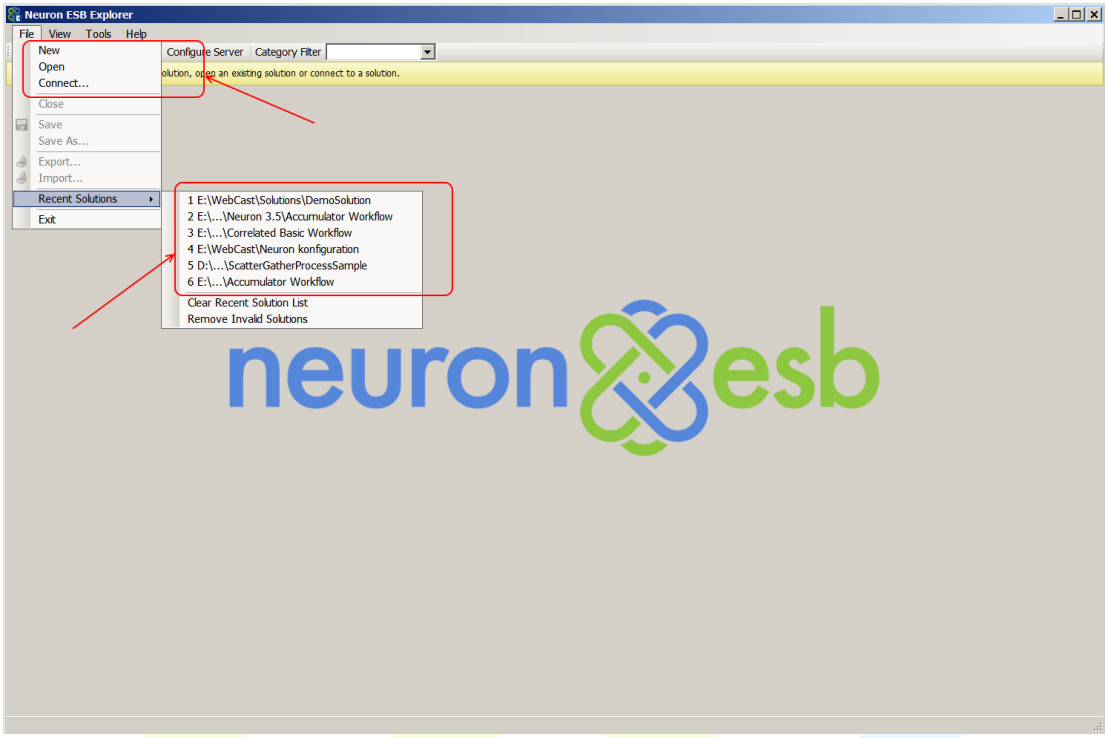

Figure 38 Neuron ESB 3.5 Explorer IDE – Neuron ESB 3.5 has moved the Connect dialog functionality to the File menu with the addition of "New", "Open" and "Connect…" menu items. The Explorer also implements an MRU (Most Recently Used) menu.

Additionally, a "Most Recently Used" menu list has been implemented so that users no longer have to search for solutions that they had previously opened.

When connecting to either local or remote instances of the Neuron ESB runtime, users can select "Connect…" from the File menu, launching a new "Connect" dialog as shown in the figure below.

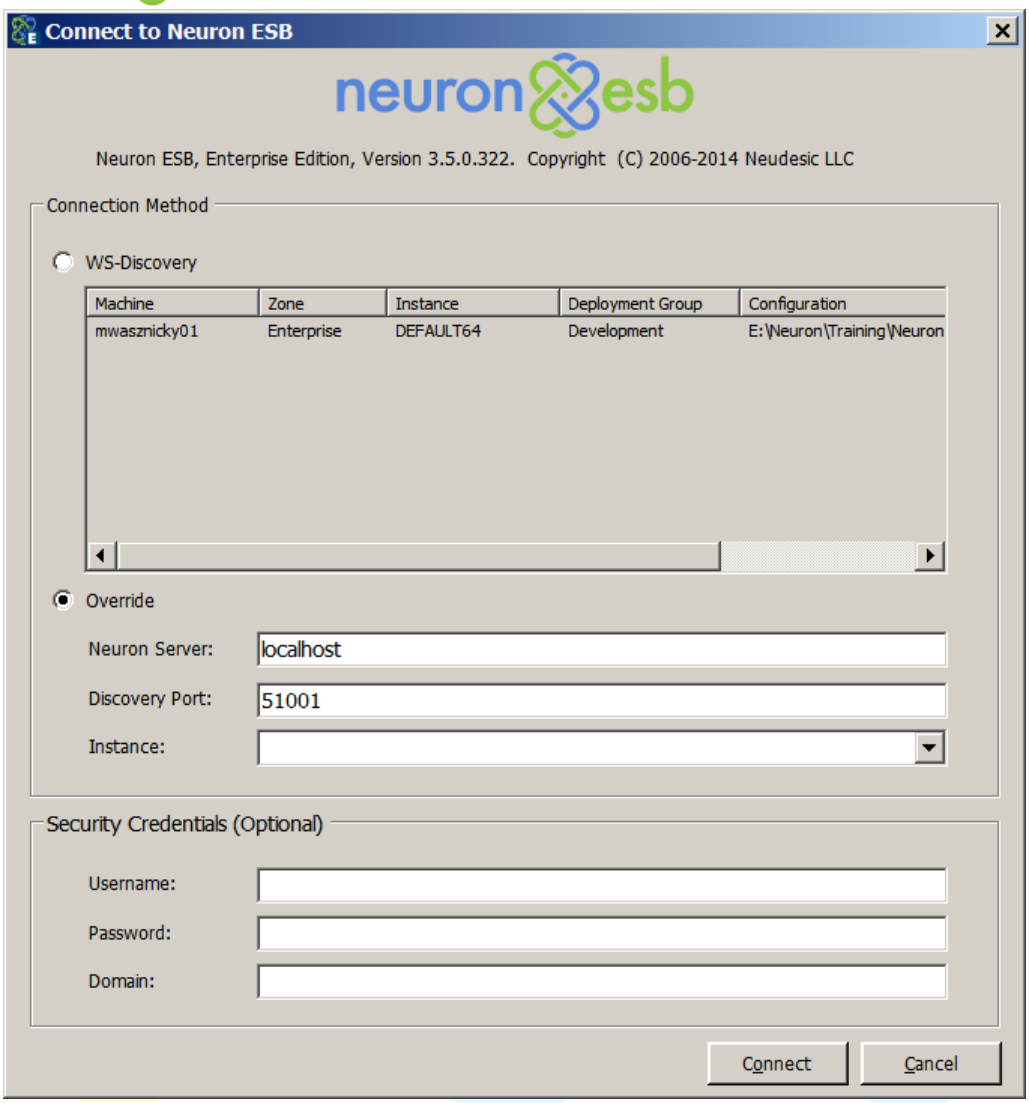

Figure 39 Neuron ESB 3.5 Connect dialog – Allows users to connect to any runtime instance either by specifying the server, or by using WS-Discovery.

The new Neuron ESB 3.5 Connect dialog is used exclusively to connect to local or remote instance of the Neuron ESB runtime. By default, the Override option is selected and uses the same Neuron ESB Discovery service that was used in previous versions, providing a list of all running Neuron ESB instances on a machine. However, this required users to know what machines had Neuron ESB runtime instances installed. Also, the user would never know if the Neuron ESB runtime selected was started until a connect attempt was made.

This same Connect dialog is also exposed through the Neuron ESB Test Client through the Tools -> Connection Settings menu.

Starting with Neuron ESB 3.5, Neuron ESB runtime instances have been enabled with WS-Discovery. Assuming local networks support WS-Discovery broadcasts, users can select the WS-Discovery option on

the Connect dialog. When selected, WS-Discovery will be used to find all the Neuron ESB runtime instances started on the network. Users can double click on any instance to connect to it.

Lastly, although not necessarily related to the UX experience, when users install Neuron ESB 3.5, the local runtime instance is no longer configured with the Sample Solution. If the Neuron ESB runtime instance is started from the Neuron ESB Explorer's toolbar menu, the following dialog will be presented to the user

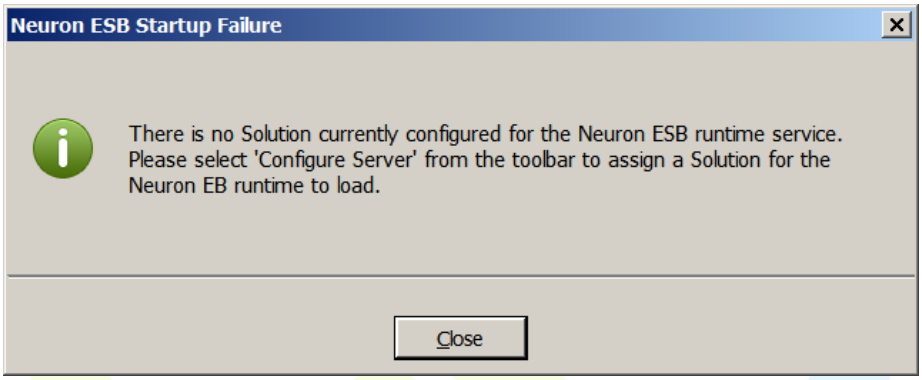

Figure 40 Neuron ESB 3.5 Startup warning – Notification to user that solution is required to be configured for local runtime before starting runtime.

Neuron ESB Database changes

Neuron ESB 3.5 Workflow requires the configuration of the Neuron ESB Database. The Neuron ESB Database can be easily created using the Neuron ESB Explorer. However, we made several changes to support Workflow, the need to clean up tracking information and allow users to regularly purge and archive Neuron ESB Audit and Workflow Tracking data.

Database Schema Changes

In the Neuron ESB 3.5 release, we changed how we manage and update our database schemas. In the past, we released a single database script that would create the entire database, and with each release of Neuron ESB, we would release updates if necessary. For users that were several versions behind, it became confusing on how to upgrade your database from one version to the next.

With Neuron ESB 3.5, we have introduced a new database schema management scheme that will make it easier to create new databases and upgrade existing databases to stay in sync with Neuron releases. The new database scheme will also make it easier for the product team to release bug fixes and new features with future releases.

The new database scheme uses a series of database scripts based on a single version number. Each script is named using the following form:

<4-digit-version-number>_<description-of-change>.sql

With the Neuron ESB 3.5 release, we have walked back through our database history from prior releases and produced the following database scripts:

- 0001_CreateNeuronDB.sql
- 0002_UpdateTo2_6.sql
- 0003_UpdateTo3_0.sql
- 0004_UpdateTo3_1.sql
- 0005_UpdateTo3_2.sql
- 0006_UpdateTo3_3.sql
- 0007_UpdateTo3_1_0.sql
- 0008_UpdateTo3_1_405.sql
- 0009_UpdateTo3_5_0.sql

The current database version number for the Neuron ESB 3.5 release is version number 9. In the new database scheme, the first database script will create the basic Neuron ESB database corresponding to the Neuron ESB 2.5 release, and each additional script will apply the incremental changes that were made to the database with each Neuron software release. Moving forward with future releases, we will add new features and fix bugs to the database by releasing new incremental update scripts to the database, leaving the existing scripts in a stable, frozen state.

The advantages to this new version numbering scheme are:

- Less confusion on how to create or update a database. Users simply run the scripts in numerical order.
- We can make it easier to automatically upgrade your database for users.

Please note: Upgrading an existing database has risks including data loss of existing data. Only upgrade an existing database after backing up your existing data. This is important should any data loss or errors occur during the upgrade so that you can restore your database to a known operational state.

Upgrading an Existing Database

The upgrade process for databases is now much easier, and is also automated. There are two ways to create or upgrade a Neuron ESB database:

- Using Neuron ESB Explorer
- Using a new PowerShell script

Using Neuron ESB Explorer

The process for creating or updating a database is the same in Neuron Explorer for the 3.5 release as it has been in the past. However, there is a new feature that will allow the Neuron ESB Explorer to upgrade an existing database to the latest version.

If you configure a database that is incompatible with the installed version of Neuron ESB, the Neuron ESB Explorer will now prompt you to upgrade your database. If you choose to upgrade your database, the Neuron ESB Explorer will determine the current database version and will apply the incremental updates that are necessary to upgrade your database to the latest version.

Using PowerShell

The PowerShell script uses the sqlps PowerShell module installed by Microsoft SQL Server. This script will not work if the sqlps PowerShell module is not installed on the server.

You do not need to import sqlps prior to running the PowerShell script. The PowerShell script will detect if it is available, and if not, will import the sqlps module automatically.

The Neuron ESB 3.5 release includes a new PowerShell script that you can use to create or delete an existing Neuron ESB database. The PowerShell script will not create the actual database. The database must exist before running the PowerShell script. But the PowerShell script will create the database structures needed by Neuron ESB in an empty database. Given an empty database or an existing Neuron ESB database, follow these steps to create or update your database:

- Open a PowerShell console.
- Use the **Set-Location** cmdlet to change the directory to the installation directory for Neuron ESB.

  > Set-Location "C:\Program Files\Neudesic\Neuron ESB v3\"
- Execute the PowerShell script, providing the correct values for the parameters (line breaks are for illustrative purposes and should not be entered into the PowerShell console):

  > .\PowerShell\UpdateNeuronESBDatabase.ps1
  –neuronEsbInstallPath "C:\Program Files\Neudesic\Neuron ESB v3"
  -serverInstance "<server-name>"
  -database "Test"

The **serverInstance** parameter can be a dot (".") for the local machine, or can be a machine name. If you have installed a named instance of SQL Server, you would specify that as well. For example, to install the database objects in a SQL Express instance, you would use **.\SQLEXPRESS** for the value of the **serverInstance** parameter.

The **UpdateNeuronESBDatabase.ps1** PowerShell script will examine the database to determine whether the database is empty or contains an existing Neuron ESB database. If the database is an existing database, the **UpdateNeuronESBDatabase.ps1** script will determine the version number for the database and will apply the update scripts necessary to bring your database up to date with the currently installed release of Neuron ESB.

Workflow Tracking Cleanup Job

Starting with the Neuron ESB 3.5 release, the Neuron ESB database needs a SQL Server Agent job to purge workflow tracking records marked for deletion. When creating a database through Neuron Explorer, an attempt will be made to create and schedule the job. If the user has insufficient database permissions for the creation of the job to succeed, then you will need to modify and run the script manually.

The job creation script is called **CreateJob_PurgeWorkflowTracking.sql** and can be found in the Sql folder under the default Neuron ESB installation folder (ex: C:\Program Files\Neudesic\Neuron ESB v3\Sql). Open the script file and replace the ${DatabaseName} placeholder. By default the script enables the job, sets the job's owner as "sa" and schedules the job to execute every 10 minutes. Modify these values as needed then execute the script. You will see a new job in SQL Server Management Studio. Make sure to start the SQL Server Agent service. NOTE: If you are saving your script changes specify a different filename so that Neuron Explorer will have access to the unchanged **CreateJob_PurgeWorkflowTracking.sql** file.

Archive and Purge Neuron DB Job

Neuron ESB 3.5 introduces the ability to automate the purging and archiving of the Neuron ESB database. For example, when Neuron ESB Auditing is used, over time the auditing tables will grow, consuming more disk space. Additionally, when using Workflow, Workflow tracking will produce increasing amounts of data in their respective tables. Overtime, there will be a need to delete older data to maintain performance. Perhaps as well, based on an organization's data retention policy, there will be a need to archive older data to an offline database or store.

There is a now a SQL Server Agent job, **CreateJob_PurgeArchiveNeuron.sql**, located in the Neuron ESB Sql folder under the default Neuron ESB installation folder (ex: "C:\Program Files\Neudesic\Neuron ESB v3\Sql\ ").

Open the script file and replace the following parameters with the values according to your organization's data purge and archive policies:

- @DatabaseName - The Neuron ESB database which you want to purge and backup
- @LoginName - The username which has ability to run the Job
- @FolderName - The folder where you want to store Neuron ESB database backup. The backup database name will be in the format of "***SERVERNAME_DATABASENAME_Mmm_dd_yyyy_hh_mm.bak***"
- @NumberOfDays - This variable is set at a default of 7. when you run your job with this default setting, any data that is older than 7 days will be moved to the backup folder (for archiving) and then purged from the current Neuron ESB database

Once the modifications are done, the script can be executed in the SQL Server Management Studio to create the SQL Server Agent Job. Once created, users can schedule the job to run during off hours.

Log4Net Neuron ESB Provider

Prior to Neuron ESB 3.1, Neuron used two facilities for outputting diagnostic information and events at runtime: writing directly to the Windows Event Log, and using the classes in the System.Diagnostics namespace to write to trace files. We heard from customers that they were looking for other logging features to integrate with what the customers were using, with log4net being the most common. In

Neuron ESB 3.1, we introduced log4net into the Neuron ESB SDK and programs not only to meet our customer requests, but to use the configurability and extensibility of log4net's logging features.

After receiving feedback from customers, we extended the logging facility in Neuron ESB 3.5 to provide a pluggable provider-based model for logging. This allows customers to integrate the logging platform of their choice into the Neuron ESB SDK in order to allow the Neuron ESB SDK components to output to the same locations or logging services that customers are using for their software.

Note that this pluggable model is only for use with customer solutions that are linking against the Neuron ESB SDK. The Neuron ESB applications such as the Neuron ESB Messaging and Workflow runtime services will continue to use log4net for their logging output.

Source code for a sample logging provider can be found on Github:
https://github.com/neuronesb/log4net-provider