20 Implementation

A description of how the generic networking functionality was created and how it works

Basics

- This project builds on our standard structure
  - 10 Repo Structure
  - 11 Project Structure
  - The modules make use of our own DQMH® flavour
    - HSE DQMH Specifics

The core functionality (`hse-gennet.lvlib`) and the GenNet reuse modules (`GenNet-Server.lvlib` and `GenNet-Client.lvlib`) are part of the HSE Reuse Collection. That way, we can easily use them in all of our projects.

This project - the GenNet Examples repository - is a showcase of how the generic, reusable gennet libs can be used in actual projects or with actual DQMH modules. It contains examples of modules using the generic networking functionality but the GenNet VIs and modules are actually maintained as a separate project in their own repository.

All the GenNet reuse code - `GenNet-Client` and `GenNet-Server` modules as well as the `hse-gennet.lvlib` - are installed via the HSE Core: GenNet VI Package.

Overview

The Generic Networking project aims to integrate network communication into DQMH in a way that is generically reusable and has no module- or project-specific dependencies. This means that the networking functions are oblivious to the actual datatype and content of the messages that are passed on, and network communication can happen totally transparent to both the user and the DQMH module itself. The Generic Networking project also implements network communication in a way that allows for enabling and disabling that feature during runtime.

All of this is achieved by:

- creating generic, reusable helper DQMH modules for sending or forwarding requests via ethernet (this is done through the `GenNet-Client` module) and for receiving requests via ethernet (by the `GenNet-Server` module) which can be used by any DQMH module
- modifying the project-specific DQMH modules to transparently use those helper modules by changing both the source code of the module's Main.VI and its Request

VIs using `hse-gennet.lvlib`, a collection of generically reusable VIs

Generic-Networking-enabled

When we use the term “Generic-Networking-enabled”, we mean overriding the default DQMH Message Queue class and making a number of modifications to the DQMH module:

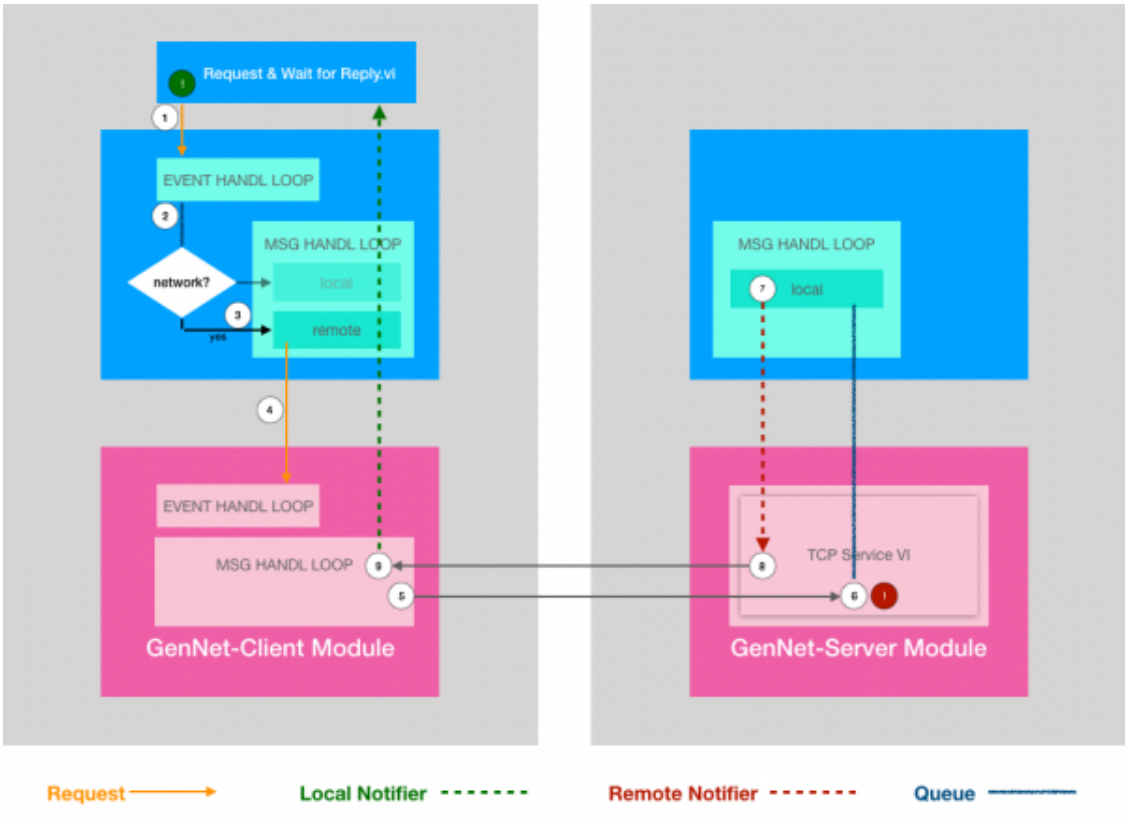
- A new MessageQueue class inherits from `Delacor_Lib_QMH_Message Queue.lvclass`
  - has a member variable “relay via network?”
  - has member variables for network configuration (ip addr, port, timeout)
2. All notifications for requests are of datatype variant
   - this allows for generic access to the notifier reference
   - actual datatype of event's notification is inside variant
3. Functions for relaying messages via network
   - During configuration, a GenNet-Client module is started dynamically
   - The calling module's message queue object is handed to the GenNet-Client `Start Module.vi`
4. Functions for receiving networked messages
   - During configuration, a GenNet-Server module is started dynamically
   - The GenNet-Server module creates a Listener and opens a port for incoming messages
   - The calling module's message queue object is handed to the GenNet-Server `Start Module.vi`

Schema

This example shows a *Generic-Networking-enabled* module. The identical module (i.e. the same source code) is running in two separate applications:

- On the left-hand (“local”) side, the module is configured to forward messages via network. It loads a `GenNet-Client` module to do that.
- On the right-hand (“remote”) side, the module is configured to receive messages via network. It loads a `GenNet-Server` module to do that.

The following part discusses the flow of data for a **Request and Wait for Reply** that is called on the local side but actually executed on the remote side.



Sequence

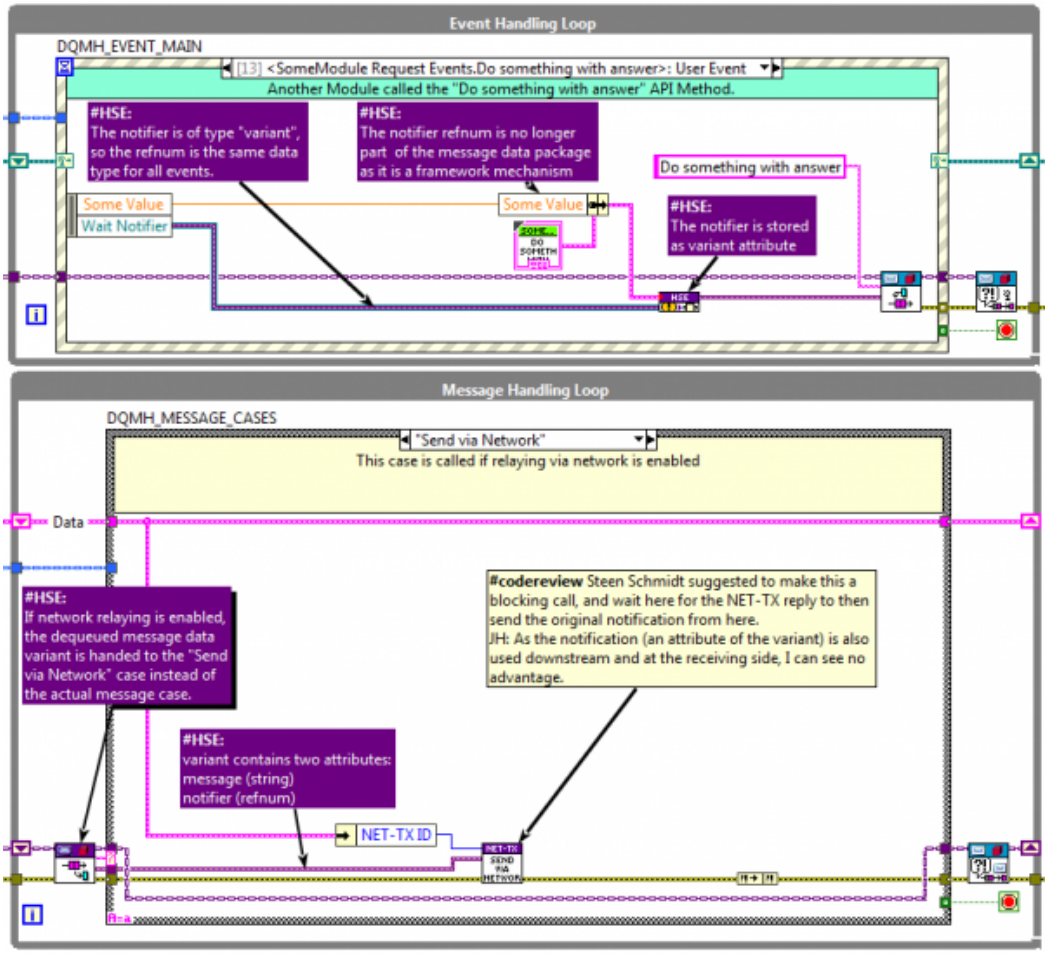
1. Local: DQMH Request Event with Notification
2. Local Module: DQMH_Event_Main sends the message data as variant
    1. Notification refnum (datatype variant) is set as an attribute of the message data variant
3. Local Module: Delacor_lib_QMH_Dequeue_Message.vi knows that the local module is configured to relay messages and thus calls the case “Send via Network”
    1. checks message (string) against blacklist of messages that are not forwarded (eg 'Exit')
    2. The original message (string) is appended as attribute to the message data (variant)
4. Local Module: The case “Send via Network” calls the “Send via Network” request event of the GenNet-Client module
    1. The message data (variant) is the payload of the GenNet-Client's “Send via Network” request event
5. Local GenNet-Client module: MSG_HANDL_LOOP “Send via Network”
    1. The original message is flattened to string and sent via TCP/IP
    2. TCP read waits for response
6. Remote GenNet-Server module: Spawns a reentrant TCP service VI for each client that connects
    1. reads the data from the network and unflattens it to variant
    2. If the variant contains an attribute “notifier” (the original one from the local requester), creates a new notifier and overwrites the invalid one in the received variant
    3. reads the message name from the variant attribute
    4. sends the message directly via the caller's message queue to the

Remote Module's MHL
5. Wait for notification

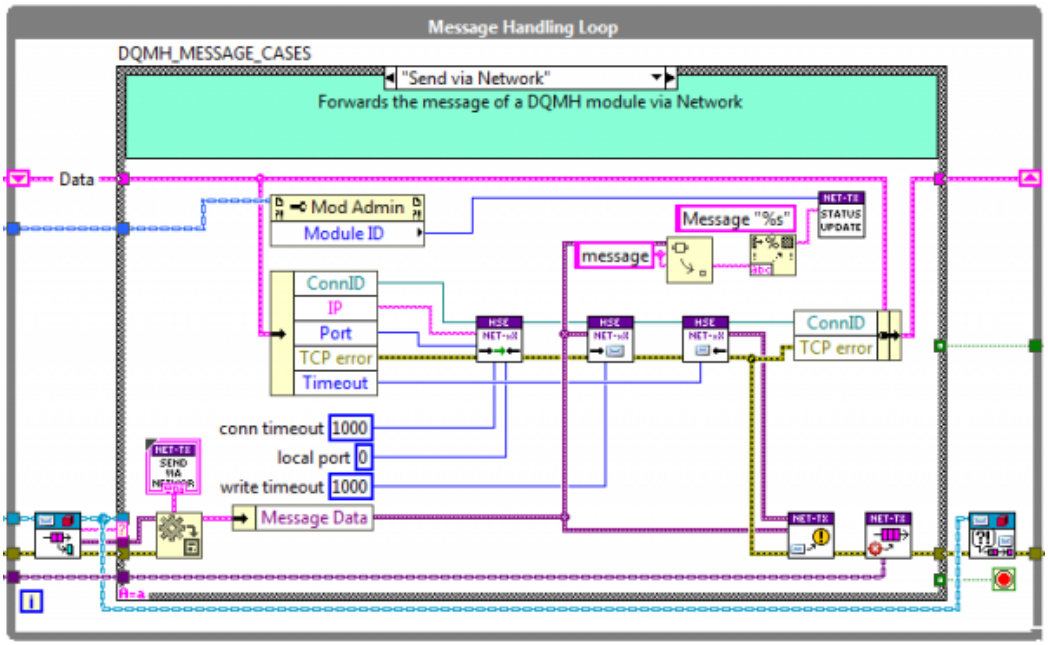
7. Remote Module: MSG_HANDL_LOOP
   1. executes the called case
   2. reads the notifier reference from the variant attribute
   3. sends the notification from TCP service VI
8. Remote GenNet-Server module: TCP service VI
   1. receives the notification from MSG_HANDL_LOOP
   2. sends the variant that was sent with the notification back via TCP
9. Local GenNet-Client module: MSG_HANDL_LOOP “Send via Network”
   1. reads data from TCP, which is unflattened to variant
   2. If the original message variant contained an attribute “notifier”, the received data is sent via the original notifier refnum back to the request VI

Screenshots

GenNet-Enabled Module



Sender (GenNet-Client)



Receiver (GenNet-Server)

TCP Service.vi