Asynchronous Message Communication (AMC) Reference Library

The Asynchronous Message Communication (AMC) Reference Library is a general purpose LabVIEW API for sending messages within a process, between processes, and between different LabVIEW targets (systems on a network) in a LabVIEW application.

The Queued Message Handler (QMH) design pattern is a general purpose VI architecture that can be used as the basis for a wide range of LabVIEW VIs. It uses the AMC API to send and receive messages. This design pattern can be used to implement state machines, user interfaces for applications, asynchronous communication processors, as well as other tasks and system components within a larger application.

This design pattern is similar to other VI architecture implementations often called queued state machines. As the design pattern can be used for many purposes other than a traditional state machine and state-oriented tasks, the name for this design pattern has been chosen to be representative of the implementation of the design pattern.

Overview

The purpose of the Asynchronous Message Communication (AMC) reference library is to provide an easy-to-use extensible messaging architecture for a variety of local and distributed LabVIEW applications. It allows you to send messages within one process of an application, between different processes on one LabVIEW target or between processes on different LabVIEW targets connected over an Ethernet network.

LabVIEW application processes in the context of this discussion are the individual ongoing operations you define as part of your application. They are typically implemented using either a While loop or Timed loop, but in some cases may also consist of a For loop or other finite program structure. Processes may include a user interface VI of your application, an I/O engine, a communication engine, a data logging engine, or a state-machine driven control process. The AMC library is used to send different messages between these processes. A message may be a command for another process to start a specific operation, perform a specific action, may be a status update, may be notification of a fault or error in particular part of the application, etc. The API is designed to be used for asynchronous messages which occur at undefined intervals and typically not very frequently. Messages or data which need to be sent frequently and at regular intervals between specific processes (e.g. data streaming) may be implemented with better run-time performance using other techniques such as raw LabVIEW queues for local processes or TCP-based communication for distributed processes. The Simple TCP Messaging (STM) reference library is good example of a TCP-based communication architecture designed for throughput performance.

Messages

A message is a piece of information contained in a LabVIEW cluster and defined by a Type Definition (TypeDef) in the AMC library. The TypeDef can be customized for the needs of your application by adding additional elements or changing the existing elements. The message definition included in the AMC library contains a message name (string), value (string), Attributes (keyed string array) and some additional parameters such as Receiver Host, Receiver Process, etc. These additional parameters are used by the AMC infrastructure and should not be changed or removed unless you clearly understand how they are being used in the different functions of the library.

Figure 1: Components of a message

Some developers may prefer to use a Variant instead of a string for the message value or to use Variant attributes to store additional attributes instead of a keyed array. If you do modify the message TypeDef you will also need to adapt a few of the functions in the library such as the conversion to and from flattened strings to match the message type.

Sending Messages Locally

The simplest use of the AMC library is to send a message from one part of a process to another. A common such application is the use of the AMC VIs in the queued message handler (QMH) design pattern which is described later on this article. In the QMH design pattern an event handler such as the UI event structure captures events from a source (the UI in this example) and passes them as messages to another part of the process (the message processor) to be taken care of. Other applications of the AMC in a single process are any type of producer/consumer design pattern.

Internally the AMC library VIs use a LabVIEW queue to buffer messages between the sender and receiver.

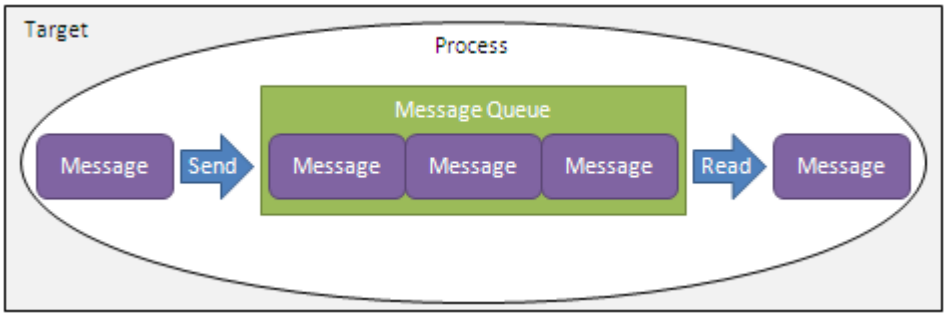


Figure 2: Sending messages within a LabVIEW process

A slightly more advanced use case of the AMC VIs is to send a message from one process to another on the same LabVIEW target. Messages are still sent through a LabVIEW queue as the same queue can be referenced in both processes. In this case each receiver process will instantiate a named message queue and any other process can send a message to the receiver process using the message queue name. Each sender queue references the receiver process and queue using the queue name provided by the receiver process.

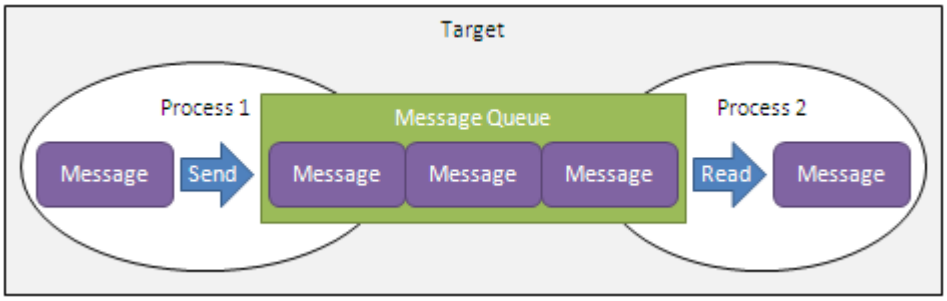


Figure 3: Sending message between LabVIEW processes on one target

For bidirectional message communication between two or more processes each receiver process will create its own receiver queue and any other processes can send their message to one common queue for each receiver. This means that each process only needs one message queue instead of establishing a separate connection between each pair of communicating processes.

Sending Messages Across the Network

When passing messages across the network from one LabVIEW target to another, messages must be passed along using a network transport protocol. The protocol used by the AMC library is UDP as it does not require an established connection between each pair of communicating targets. This allows the AMC library to easily pass messages between any number of targets on a network in a distributed LabVIEW application.

To handle messages sent using UDP at the receiver, each target which will be accepting incoming messages must run the AMC Dispatcher process. This process maintains an open UDP port and listens for incoming messages from any other targets on the network. When a message is received in the dispatcher it is forwarded to the message queue of the local receiver process. The name of the receiver process is included in the parameters of the message itself.

Using this design a process on one target can send a message to any named process on another target in the application. The sender only needs to know the target name (IP address) and the name of the receiver process on the remote target.

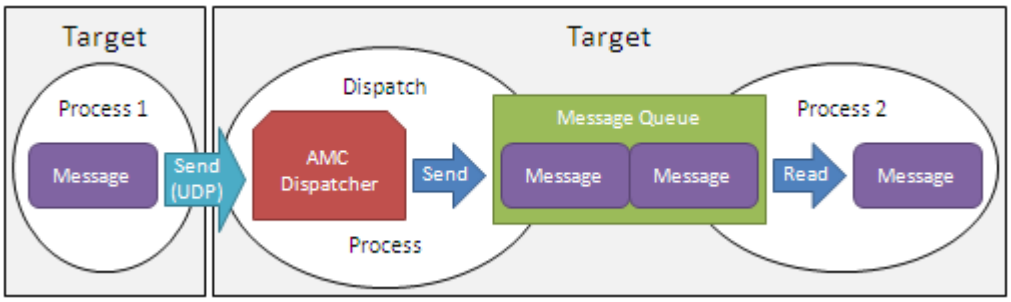


Figure 4: Sending messages to a process on a remote (networked) target

The AMC Dispatcher provides a number of additional service functions described later on which can be used in an application to verify existence of individual remote targets and processes as well as to enumerate all the processes available on a remote target.

Asynchronous Message Communication (AMC) API

The AMC API is organized into a number of different groups of VIs. These groups contain VIs for:

- Sending and receiving messages and managing local message queues
- Managing the local and remote AMC Dispatcher
- Registering and unregistering local message queues

The AMC function palette contains the most commonly used VIs on the main palette and has a number of subpalettes for the remaining groups of VIs. Additional subpalettes contain a number of templates from common design patterns using the AMC VIs.

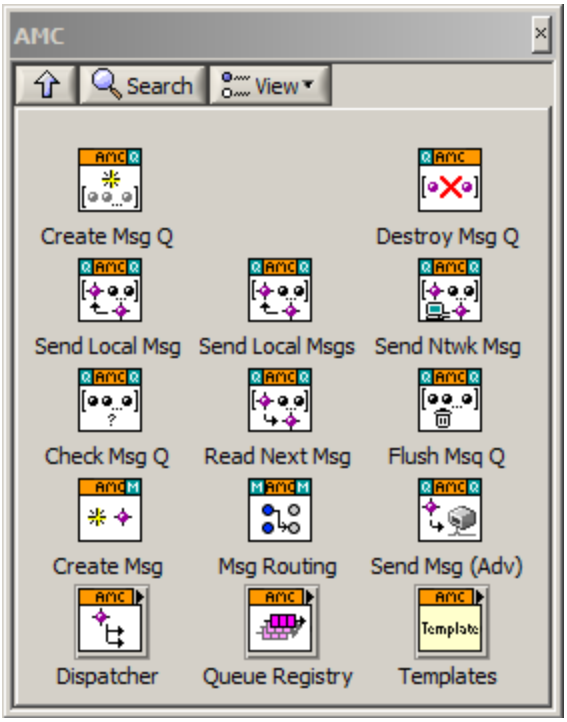
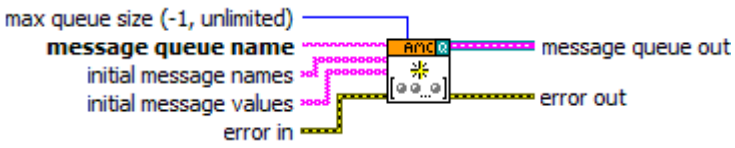


Figure 5: AMC Function Palette

Managing Message Queues

There are four basic VIs included in the AMC API to manage message queues.

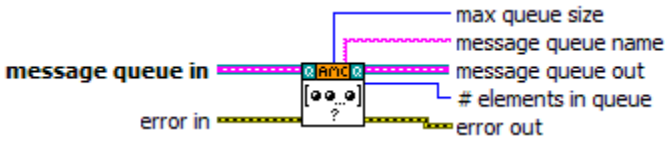


Create Message Queue

Each process which will be receiving messages uses the *Create Message Queue VI* to create a new message queue with a unique name. The name is provided by the process itself. When creating a new message queue you can also place some initial messages in the queue which will be read by the process. This is commonly used in the QMH design pattern to perform some initialization actions in the local process.

Destroy Message Queue

When a process is shutdown any message queue created by the process must be terminated using the *Destroy Message Queue VI*.



Check Message Queue Status

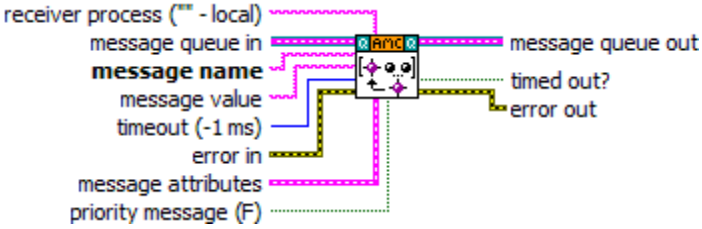
A process can check the status of its local queue and the number of messages located in the queue using the *Check Message Queue Status VI*.

Flush Message Queue

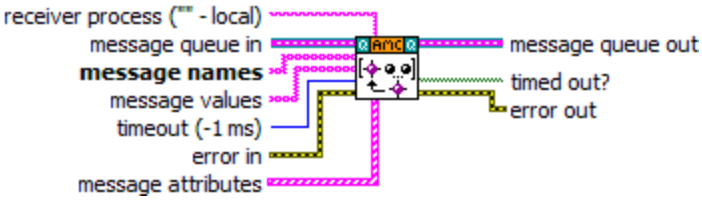
If desired a process can empty the local message queue and remove all waiting messages without processing them using the *Flush Message Queue VI*.

Sending and Receiving Messages

Messages can be sent using a number of different VIs in the AMC API.



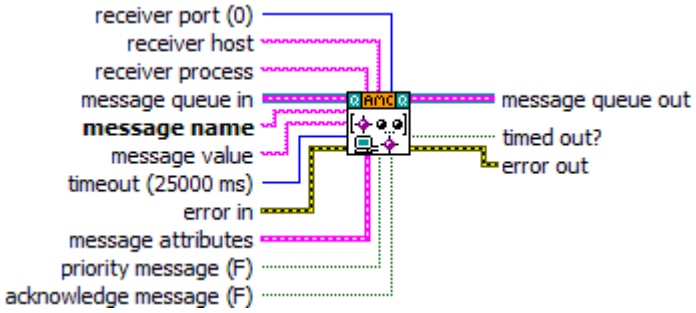
Send Local Message



Send Local Messages

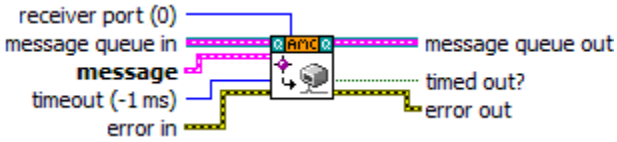
Messages can be sent to a process on the local target using the *Send Local Message* or *Send Local Messages VI*. If the Process input is blank the message will be sent to the message queue referenced by the Message Queue in parameter, otherwise the message is sent to the local message queue specified by the process name. The Priority

Message input allows you to place messages on the front of the message queue instead of the back. Messages placed on the front of the queue will be the first ones dequeued in the receiving process.



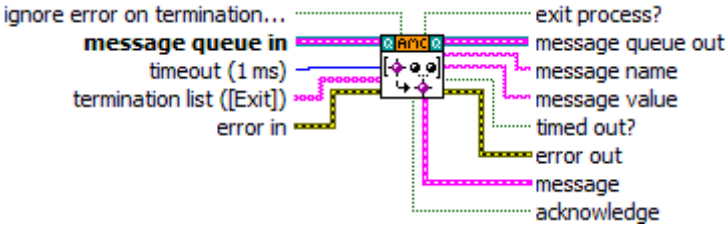
Send Network Message

Messages are sent to a process on a remote target using the *Send Network Message VI*. This VI sends the message using UDP directly to the remote target specified in the Receiver Host input where the AMC Dispatcher will forward the message to the local message queue specified in the Receiver Process input.



Send Message

The *Send Message VI* is a low level generic VI that can be used to send messages to a local or remote process.



Read Next Message

The *Read Next Message VI* is used to retrieve the next message from the message queue to be handled in the local process. The message queue is passed in by queue reference as it will be created in the same process.

The Termination List input is used as part of the QMH design pattern or other similar message processors. Any message being dequeued which matches any of the names in the Termination List will cause the Exit Process output of the VI to be set to True. This can be used to automatically terminate the processor loop. The default termination message is 'Exit'.

Using these Message Send and Read VIs you can quickly and easily send flexible messages to any process in your application. Remember that to send messages to a remote target, the receiver target must have the AMC Dispatcher running as described in the next section.

Network Communication and the AMC Dispatcher

To enable receiving messages using AMC across an Ethernet network the AMC Dispatcher must be running on the receiving LabVIEW target system. The AMC Dispatcher is a standalone VI and process which open and monitors a UDP port for incoming messages and forwards them to different message queues on the local target based on the Receiver Process parameter in each message.

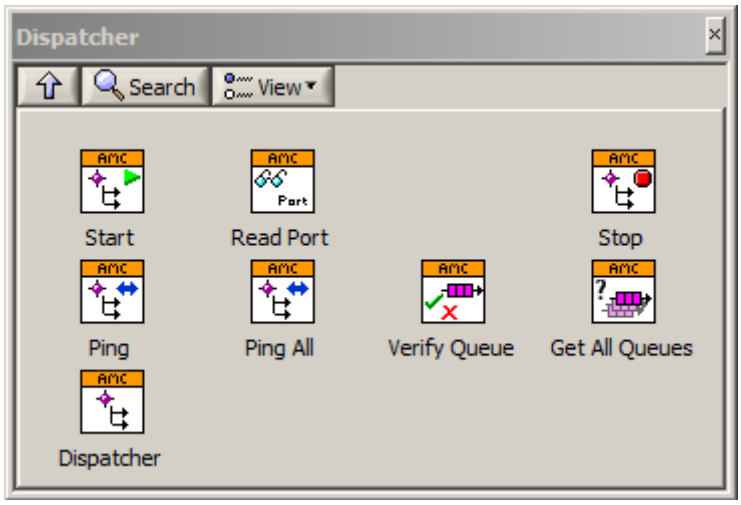


Figure 6: AMC Dispatcher Palette

On most LabVIEW targets (Windows XP, Windows XP embedded, Windows Vista and LabVIEW Real-Time) you can use the *Dispatch Start VI* and *Dispatch Stop VI* to dynamically start and stop the AMC Dispatcher. On a LabVIEW Windows CE target you must call the AMC Dispatcher statically by placing it in your main VI in parallel to the rest of your application. When called statically the AMC Dispatcher is a regular subVI and will not return until terminated it by send an Exit message to the AMC Dispatcher itself.

The AMC Dispatcher function palette provides additional service functions which can be used to further automate your application or verify remote targets. These VIs send different requests to AMC Dispatchers running on remote targets. All of these functions require that the AMC Dispatcher is also running on the local target which is sending out these requests as the remote Dispatchers will send back responses which are received by a local AMC Dispatcher.

The *Dispatch Ping VI* is used to send out a ping to a specific network target and AMC Dispatcher in order to verify that the AMC Dispatcher is running and able to receive messages on the specified remote target.

The *Dispatch PingAll VI* is used to send out a ping to all network targets on the local subnet which will prompt all AMC Dispatchers on the local subnet to respond. This function is used to establish a list of targets on the local subnet which can receive AMC messages.

The *Dispatch Verify Queue VI* is used to send out a query to a specific target and verify the presence of a specific named message queue and associated process. This VI is used to establish if a particular process is running and able to receive messages.

The *Dispatch Get All Queues VI* is used to request a list of all messages queues available on a specified remote target.

AMC Queue Registry

The AMC Queue Registry is a LabVIEW Functional Global Variable and associated interface VIs which is used to keep track of the list of message queues used within one target.

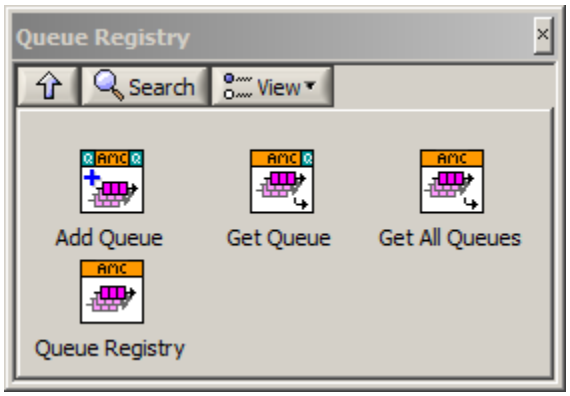


Figure 7: AMC Queue Registry Palette

The *Create Message Queue* and *AMC Destroy Message Queue VIs* use these VIs to update the list of message queues. The registry is also used by the AMC Dispatcher to respond to a Get All Queues query from another target.

Queued Message Handler Design Pattern

The Queued Message Handler (QMH) design pattern is a VI template which is included with the AMC reference library. It is available on the AMC function palette and can be dropped directly onto the diagram of a blank VI. THE QMH is a basic producer-consumer architecture which can be used as the basis for a wide variety of LabVIEW VIs throughout an application. Common uses of the QMH design pattern are the user interface of an application, dialog windows, command and communication parsers, state-based controllers and more.

The purpose of the QMH design pattern is to receive, store, and then process messages. Messages can be events from a user interface, commands received from a communication interface and others. In general a message is an asynchronous event which requires some action, operation, service, or other response.

The QMH is implemented using the Asynchronous Messaging Communication (AMC) library, which stores messages from one or more message generators or sources. Messages are read from the AMC message queue by the message processor. The message processor consists of multiple cases or states, each dedicated to process a specific message coming from the queue.

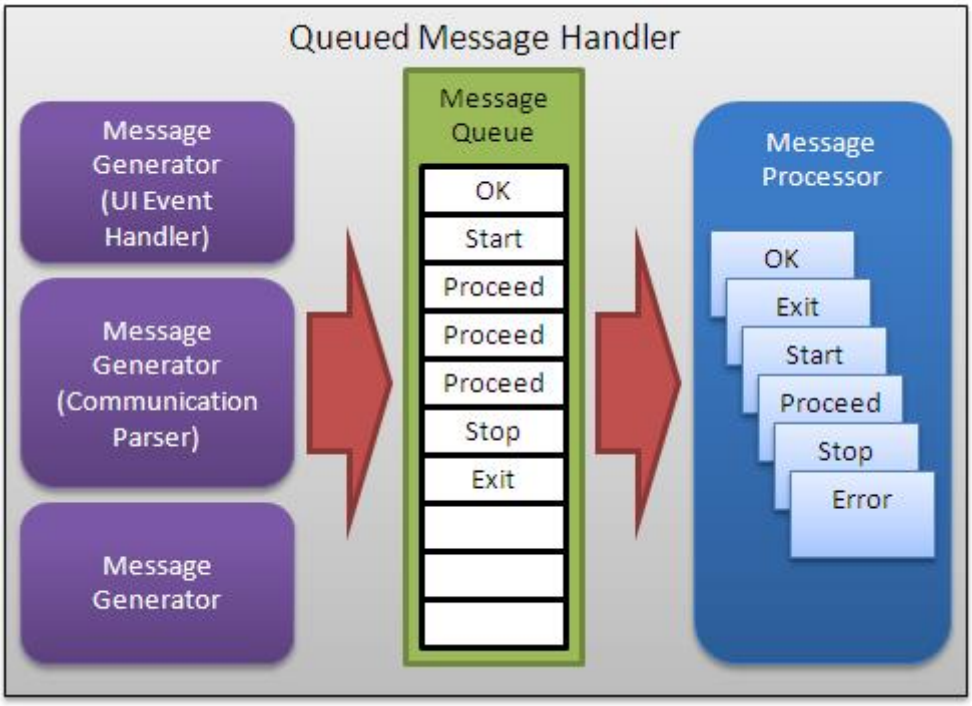


Figure 8: Queued Message Handler Design Pattern

The message queue can be accessed from multiple message generators who all place messages on the queue for processing by the message handler. However, it is common that only a single message generator exists in a particular implementation such as a user interface VI.

In addition to dedicated message generators, processing routines within the message handler may also place new messages on the message queue while processing other messages. For example, an error in processing one message may cause the message handler to place an error message on the queue to be subsequently processed by another routine in the message handler.

During normal operations of the QMH, messages are typically placed at the end of the queue and processed from the front of the queue in a First-In First-Out (FIFO) fashion.

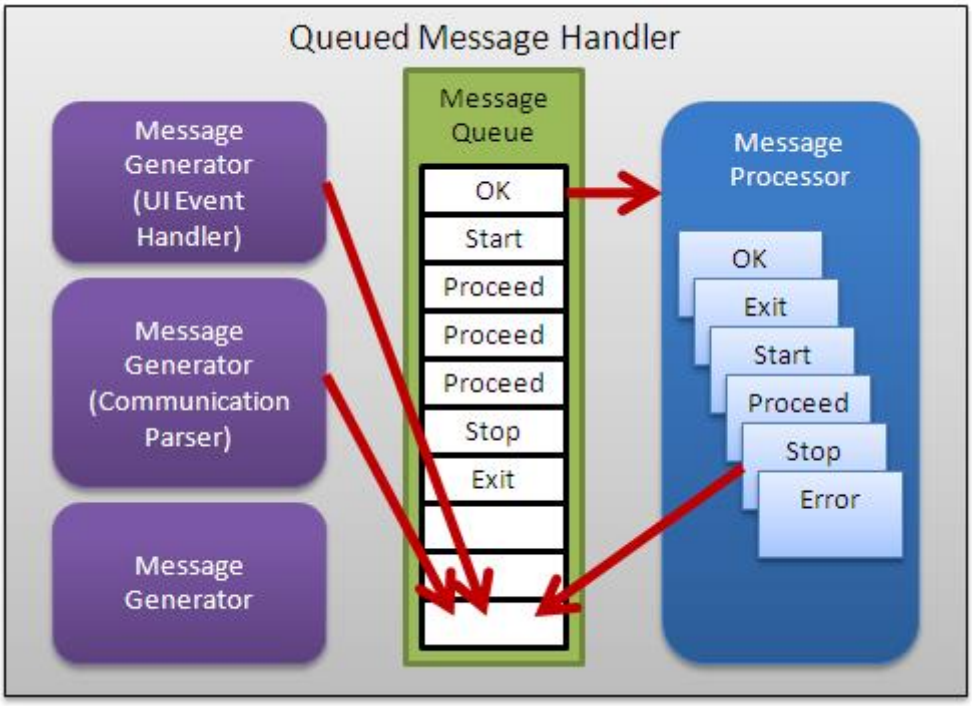


Figure 9: QMH normal operation placing new messages at the end of the queue

In some cases it is desirable that a new message on the queue will be processed as soon as possible. In this case the message may be put at the front of the queue using the Priority Message flag in the AMC library and will be processed the next time the message processor retrieves a message from the queue.

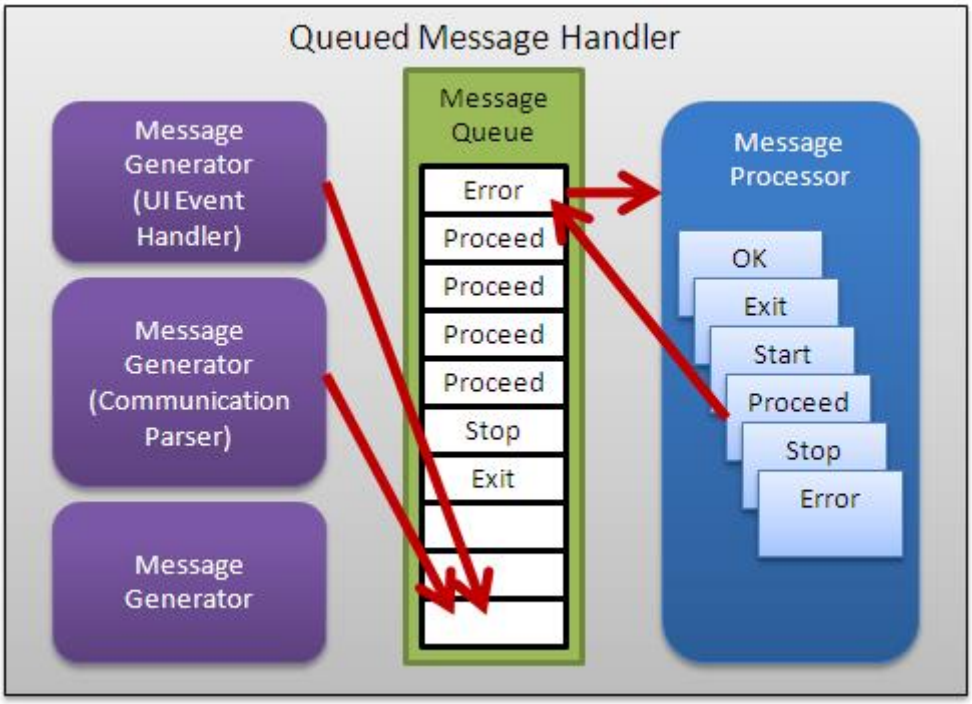


Figure 10: QMH priority operation placing a new error message at the front of the queue

The Queued Message Handler (QMH) design pattern is built around the AMC library and LabVIEW queues in order to store messages and uses a While Loop/Case Structure-based message processor. To simplify the use of the QMH design pattern in new VI's and applications, the AMC library includes a set of templates and examples, which can be used as a starting point for new VIs and applications. After installation of the AMC reference library the QMH templates are available in the AMC subpalette of the Data Communication functions palette.

Messages Definition

When developing a queued message handler, there are a number of possible ways to define the message stored in the queue. Common implementations include strings, variants and enumerations. Each of these choices offers different advantages and disadvantages.

The AMC reference library uses a string as the basic data type for messages, as it provides the ability to easily pass any message to the queue without the need to define a set of permissible messages in the VI or application. This simplifies the process of adding new messages to the application and simplifies using the reference library in a wide range of applications. However, using a string as the message name datatype does make it more likely to create a typo in the message name or to add messages to the queue which are not handled in the message processor. Care must be taken when specifying messages and a special error case in the message handler should be used to detect and report any unhandled messages.

Instead of a string, an enumeration can be used to define a specific set of messages allow by the application. Creating a TypeDef enumeration does make it easier to change the set of allowed messages in an application, while preventing any typos in selecting a message during programming. However, when using an enumeration, the set of allowed messages needs to be adapted for each use of the AMC library and therefore one set of VIs for manage the message queues and messages cannot be used for multiple applications of the QMH design pattern.

Creating the Message Queue and Adding Messages

Before adding messages to the queue it must be created using the *Create Message Queue VI*. As part of creating the queue you can place a number of initial messages on the queue. These messages will be the first messages handled in the message processor. Typically these messages are used for the startup or initialization of the VI.

After creating the queue, messages can be added to the queue using the *AMC Send Local Message VI*.

In the following diagram the queue is created and three initial messages are added to the queue. In the user interface event handler the Write Config File message is added to the queue when the operator presses the Save State button on the UI. The message processor is not shown in this diagram.

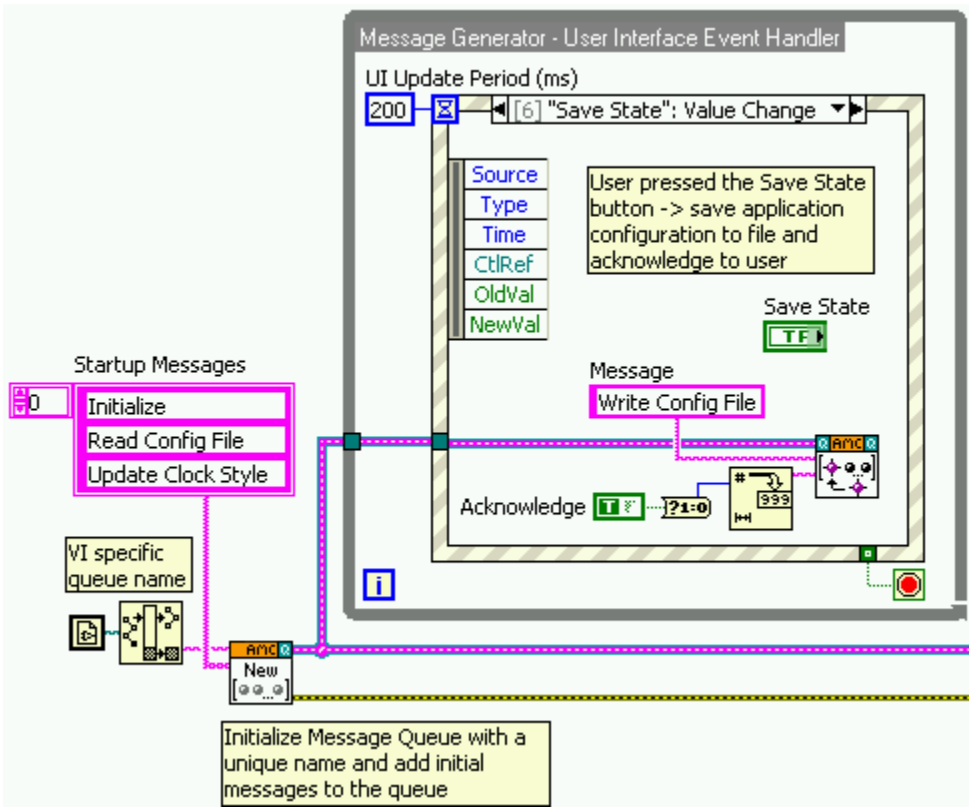


Figure 11: Message Queue Initialization and UI Event

You can also add multiple messages to the queue in a single operation using the *Send Local Messages VI*. Multiple messages are passed as a string array to the VI.

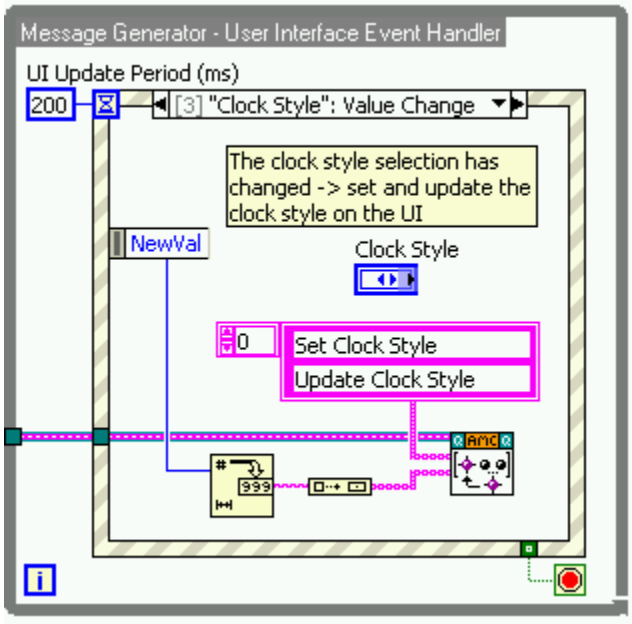


Figure 12: Example of adding multiple messages to the message queue

In addition to the messages stored on the queue in some cases it is useful to be able to pass additional data such as values or attributes along with the messages. This additional data is used in the message processor to customize the processing of the message. Message values are passed as a string or string array to *Send Local Message(s).* The previous diagram includes the new value of the Clock Style control as a parameter with the Set Clock Style message.

Processing Messages

From the queue, messages are read in the message processor loop and then handled by a separate case for each of the messages used in a message handler. The basic message processor includes the *Read Next Message VI* and a case structure with individual cases for each message.

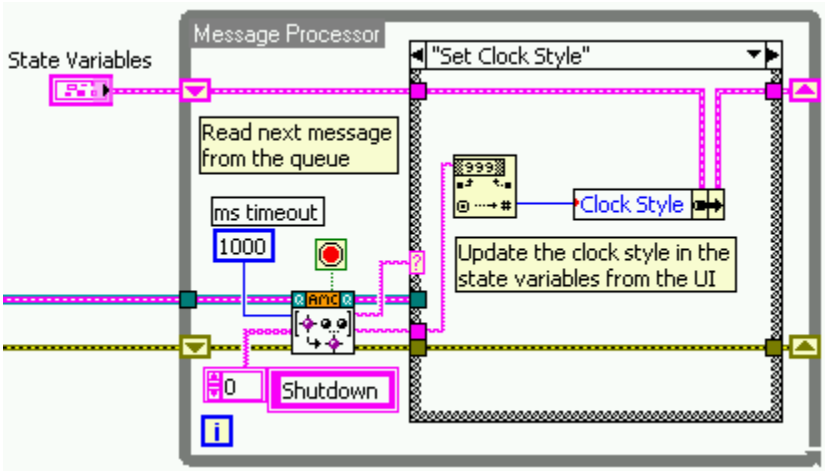


Figure 13: Message processor loop showing one message case 'Set Clock Style'

Reading Messages from the Queue

The *Read Next Message VI* includes a few specific features that should be observed in the message processor. The default operation of this VI is to return the next message from the queue and the associated value string and attributes keyed array. The message name is used as the case structure selector for the message processor and the value and attributes may be used in the specific message processor case.

Timeout

The timeout value on *Read Next Message* determines when the VI returns if no message is available on the queue. In this case an empty message string is returned from the VI which should be handled in an appropriate case. The message handler template included in the QMH design template contains a message processor case labelled *""*, *"Wait"* which is called if there is a timeout on the *Read Next Message VI*. This message processor case can be used to handle background tasks necessary in the VI operation such as front panel updates, etc. This case can also be called by placing a *Wait* message on the queue.

QMH Termination

Terminating the QMH must be handled so that both the message processor and all message generators such as the user interface event loop are shutdown properly. In some cases the message generating loop may continue to run if it also handles other operations in the VI. However, it should not add any more messages to the queue after the message processor has been shut down.

Typically the QMH will be terminated in response to a user action from the front panel such as a Quit button or a message received from another part of the application. In the message generator a message is added to the queue to indicate to the message processor to shutdown. In addition the message generator loop can be shutdown by passing a True to the While loop conditional Stop terminal.

When the message to shutdown the QMH is received in the message processor, it should execute any necessary operations in the corresponding message case before terminating the message processor loop. The *Read Next Message VI* is designed to recognize one or more messages for termination of the QMH and will return a True value on the Exit Process output which can be used to terminate the message processor loop. The default message to shutdown the local process is *Exit*, but it can be changed by passing the desired shutdown message name(s) to the *Read Next Message VI*.

Default Message Processor

Due to the use of strings as the message name datatype it is possible to make errors or typos when adding messages to the message queue. Any such erroneous messages will not have a corresponding message case in the message processor. Therefore the Case structure default case of the message processor should be used to catch any undefined messages from the queue. Normally any messages passed to this case are the result of a programming error and should be reported as such. Once a VI or application has been completely tested no erroneous message should trigger this message case.

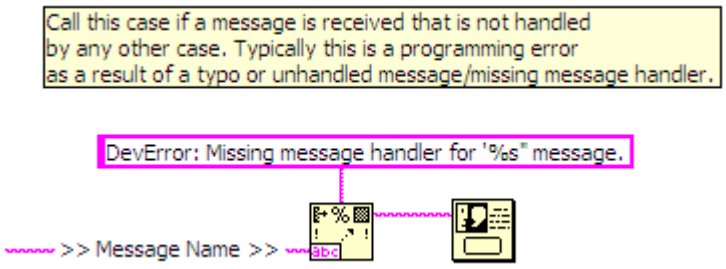


Figure 14: QMH Default Message Case

QMH Template

The Asynchronous Message Communication reference library includes a template of the QMH design pattern. The QMH template can be placed on the diagram of an empty VI from the AMC Templates function palette. Due to some slight differences in the LabVIEW execution engine for Windows CE there is a separate QMH template for Windows CE-based touch panel targets.

The following figure shows the diagram of the QMH design pattern template.

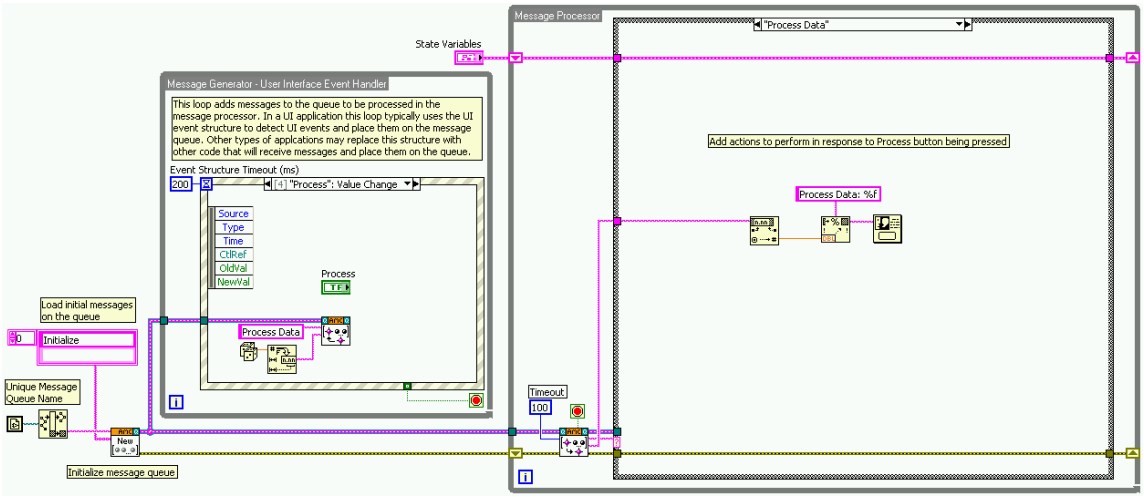


Figure 15: Queued message handler template available in the QMH function palette

The QMH template is targeted at a UI-based VI which processes UI events in the message handler. The UI events handled by the VI can be customized in the message generator loop on the left. Message processor cases can be added to the loop on the right.

The template can be adapted to other types of applications and VIs by modifying the message generator loop. If necessary the message processor may also be adapted. The template should be used as a reference design and customized according to the needs of the individual application.

Examples

The Asynchronous Message Communication reference library includes several examples showing the use of the AMC library and QMH design pattern. These example are installed along with the reference library VIs in the <LabVIEW>\examples\AMC\ folder.

References

The Asynchronous Message Communication (AMC) reference library and Queued Message Handler (QMH) design pattern presented in this article is only one possible implementation of the general concept of a queued message handler or queued state machine. The following links provide references and discussions on similar and related design patterns from a variety of sources.

Developer Zone: Application Design Patterns: State Machines

ExpressionFlow: LabVIEW Queued State Machine Architecture

JKI Software: State Machine