finPOWER Connect 3

Message Queue Guide

Version 1.01

8th December 2021

Contents

Disclaimer

This document contains information that may be subject to change at any stage.

All information, including code examples, are provided "as is" without warranty of any kind.

Version History

Date	Version	Name	Changes
1/10/2021	1.00	JR	Created.
8/12/2021	**1.01**	**JR**	**Added comment regarding available from version 3.4.2**

Introduction

The Message Queue is used internally by finPOWER Connect – but can also be used by scripts to implement simple Queue functions.

For example, if you needed to send updates to another application when Client information changes, rather than do this when the Client is saved you could queue the change and process later.

This has the advantage of not bogging down Client updates with an extra process that might be slow. Worse, what if the update to the other application fails because that system is temporarily offline, do you fail the Client update – which would be very annoying to the User.

The update to the other application can then happen independently via a separate process – which has the advantage of offloading the grunt work from the main User Interface.

Of course, operating a Message Queue is not simple – because it has to be co-ordinated properly.

- Only one process can update/ process a queued item at a time.
- It may be important to handle messages in the proper chronological order.
    - If you have queued a message to add a contact method, then to delete it – you must process the add before the delete.
- If processing a Queue Item and it gets stuck, there needs to be a graceful way to continue.
    - Getting stuck does not mean it fails, it might mean the process crashed.
- Items in the Message Queue are transient.
    - Once they have been processed, they could be purged at any time.

IMPORTANT: Message Queue functionality is available from version 3.04.02, released on 2 December 2021.

Overview

This section gives an overview of how the Message Queue works.

Note, functions are found at `finBL.MessageQueueUtilities`.

- Add an Item to the Message Queue.
  - Call `CreateItem`.
  - Parameters:
    - `ServiceId`
      - Mandatory.
      - Up to 50 characters.
      - You might have several Message Queues, this identifies the service you are handling.
    - `Data`.
      - A simple string, that could be anything you like.
      - E.g., JSON data to send in a request – Properties might store the request type/ endpoint information.
      - E.g., An `ISKeyValueList` object.
    - `Properties`.
      - An `ISKeyValueList` object containing information you want to store against the Item.
      - E.g., the type of Request.
  - This should quickly return True (Success) as all it is doing is inserting a record into the ISQueue table.
  - The Queued Item will have a Status of "Queued".

- Processing a Message Queue Item.
  - Use **`GetItemsForProcessing`** to return a list of Items ready to be processed.
    - `filterType` should be set to `iseMessageQueueType.User`.
    - Use `filterServiceId` to filter by Service.
    - You can specify the maximum number of Items to return in `limit` (default is 100).
    - All Items with a Status of "Queued" and "Processing" are returned by default.
      - See "Processing in Sequential/ Chronological Order" for more information.
    - The function returns a list of `ISMessageQueueItem` objects.
      - A `ISMessageQueueItem` object must be passed to `ProcessingBegin` and `ProcessingEnd` functions.
  - Processing an individual Item works by executing two calls, in between this you do the work required.
    - `ProcessingBegin`
    - Do work
    - `ProcessingEnd`
  - **`ProcessingBegin`**
    - Parameters:
      - `messageQueueItem As ISMessageQueueItem`
        - Pass in the Item returned in `GetItemsForProcessing`.
      - `Optional processLogData As String = ""`

        - If not blank will add an item to the `ProcessLogs` collection, see later.
    - `Optional newData As String = ""`
        - If not blank will update and replace the `Data` property.
    - `Optional newProperties As ISKeyValueList = Nothing`
        - If not Nothing will update and replace the `Properties` property.
    - `Optional allowInDatabaseTransaction As Boolean = False`
        - See "ProcessBegin versus Database Transactions" section later.
    - `Optional retrySeconds As Integer = ISMessageQueueUtilities.DefaultRetrySeconds`
        - See "Retry Seconds" later.
    - Will fail if Item's `Status` is not one of:
    - `iseMessageQueueStatus.Queued`
    - `iseMessageQueueStatus.ProcessingExpired`
        - See "Retry Seconds" later.
  - This updates the Item's Status to "Processing".
    - This means that no one else can start processing the Item.
- **ProcessingEnd**
  - Parameters:
    - `messageQueueItem As ISMessageQueueItem`
        - Pass in the Item used in `ProcessingBegin`.
    - `newStatus As iseMessageQueueStatus`
        - `iseMessageQueueStatus.Success`
            - Indicates processing was successful.
        - `iseMessageQueueStatus.Failed`
            - Indicates processing failed.
        - `iseMessageQueueStatus.Queued`
            - Indicates processing did not successfully complete and should be tried again.
    - `Optional processLogData As String = ""`
        - If not blank will add an item to the `ProcessLogs` collection, see later.
    - `Optional newData As String = ""`
        - If not blank will update and replace the `Data` property.
    - `Optional newProperties As ISKeyValueList = Nothing`
        - If not Nothing will update and replace the `Properties` property.
  - The `ProcessCount` property is incremented by 1.

Retry Seconds

If processing a Queue Item and it gets stuck, there needs to be a graceful way to continue.

Getting stuck does not mean the Item has failed, it most likely means the process terminated abnormally – and the Item's `Status` is stuck at `iseMessageQueueStatus.Processing`.

> This is a very unusual scenario, but still needs to be handled appropriately.

- To handle this there is a special Status of `iseMessageQueueStatus.ProcessingExpired`.
  - **Note, this Status is NEVER saved to the database**.
- This indicates the Item has been processing for more than the "Retry Seconds".
- Properties on `ISMessageQueueItem` include:
  - `ProcessingSeconds`
    - The number of seconds the Item has been processing for.
  - `StatusResolved`
    - The same as `Status`, with one exception
    - `iseMessageQueueStatus.ProcessingExpired`
      - Where `Status` = `iseMessageQueueStatus.Processing` and `ProcessingSeconds` > Retry Seconds.
- Retry Seconds has a default value of 300 seconds, i.e., 5 minutes.
  - This can be varied in the following functions:
    - `finMessageQueueUtilities.GetItemsForProcessing`
    - `finMessageQueueUtilities.ProcessingBegin`
    - `ISMessageQueueItem.StatusResolved`
- Importantly:
  - `ProcessingBegin` will fail if `StatusResolved` is `iseMessageQueueStatus.Processing`.
  - If `StatusResolved` is `iseMessageQueueStatus.ProcessingExpired` it can continue and assume that the Item was stuck and can be processed.
    - You may still wish to process the Item different if you detect this Status.
      - For example, some APIs allow you to query on the status of a Request, so you could confirm what happened to the original request.
      - Other APIs may support Idempotent requests, which means you may safely be able to send the same Request again.

Statuses

- Active Statuses:
  - `iseMessageQueueStatus.Queued`
  - `iseMessageQueueStatus.Processing`
  - `iseMessageQueueStatus.ProcessingExpired`
- Terminal Statuses:
  - `iseMessageQueueStatus.Success`
  - `iseMessageQueueStatus.Failed`
  - `iseMessageQueueStatus.Aborted`

Sample Code

Creating an Item.

```
Dim Properties As ISKeyValueList

Properties = finBL.CreateKeyValueList()
With Properties
  .SetString("Type", "Update")
  .SetString("Client", "C10000")
End With

If finBL.MessageQueueUtilities.CreateItem("SERVICE_A", "Some JSON Data", Properties) Then
  ' Success
Else
  Success = False
End If
```

Processing Items.

```
Dim MessageQueueItem As ISMessageQueueItem
Dim MessageQueueItems As List (Of ISMessageQueueItem)
Dim MessageQueueStatus As iseMessageQueueStatus

' Process - SEQUENTIAL
If finBL.MessageQueueUtilities.GetItemsForProcessing(iseMessageQueueType.User, "SERVICE_A",
MessageQueueItems, True) Then
  For Each MessageQueueItem In MessageQueueItems
    If finBL.MessageQueueUtilities.ProcessingBegin(MessageQueueItem) Then
      ' Process Item
      MessageQueueStatus = Me.DoWork(MessageQueueItem)

      ' Complete Processing
      If Not finBL.MessageQueueUtilities.ProcessingEnd(MessageQueueItem, MessageQueueStatus)
Then
        Success = False
        Exit For
      End If
    Else
      ' Quit - we cannot continue processing as must be processed in sequence
      Success = False
      Exit For
    End If
  Next
Else
  Success = False
End If
```

```
Public Function DoWork(messageQueueItem As ISMessageQueueItem) As iseMessageQueueStatus

' DO WORK HERE

'  ' If Failed, but can be retried, set Status to Queued
'  Return iseMessageQueueStatus.Queued
'
'  ' If Failed, but should not be tried again, set Status to Failed
'  Return iseMessageQueueStatus.Failed

  ' If Successful, set Status to Success
  Return iseMessageQueueStatus.Success

End Function
```

Processing in Sequential/ Chronological Order

It may be important to handle messages in the proper chronological order.

- This is the default behaviour for Message Queues.
- For example:
  - Queued a message to add a Contact Method to a Client.
  - Queued a message to delete the same Contact Method to a Client.
  - If these were processed in the reverse order the other application may error when trying to delete the Contact Method and/or end up with a Contact Method that is now deleted in finPOWER Connect.
- To handle processing in sequential order.
  - Call `GetItemsForProcessing`.
    - Pass in `sequential` as True (the Default).
    - `retrySeconds` has no effect.
    - The list of Items includes all Items with a Status of "Queued" or "Processing".
  - Process Items, starting from the first item.
    - STOP if any item fails when `ProcessingBegin` is called.
      - `ProcessingBegin` will fail if the Item's Status is "Processing" and has been like this for less than Retry Seconds.
        - `retrySeconds` is an optional parameter passed into `ProcessingBegin` – if not defined will used the default 300 seconds, i.e., 5 seconds.
    - If an Item fails, e.g., the other Application returns a failure response you will have to decide whether:
      - This means the Item fails – AND you continue processing.
        - E.g., the Client doesn't exist in the other Application, so no point trying again.
      - Or, whether `ProcessingEnd` should set the Item back to a Status of "Queued", so it gets processed again – AND you STOP processing.
        - E.g., the other Application returns a server error.
        - You could take other actions at this time, e.g., send an email to an Administrator.
        - There are `ProcessCount` and `ProcessLogs` properties that you can check to help determine what action to take.

Processing where Sequential Order doesn't matter

To handle processing of items in any order.

- Call `GetItemsForProcessing`.
  - Pass in `sequential` as False.
  - Optionally, pass in `retrySeconds` as the number of seconds to treat "Processing" Items as still processing.
    - The default is 300 seconds, i.e., 5 minutes.
    - Items that have been processing for less than this time are NOT returned.
  - The list of Items includes all Items with a Status of "Queued" or "Processing" more than Retry Seconds.
- Process Items returned.
  - You can process any item, but generally you would start from the first item.
  - If `ProcessingBegin` fails, ignore that Item and continue to process other items.

Process Logs collection

The `ISMessageQueueItem` object contains a collection of `ISMessageQueueItemProcessLog` objects.

- This can be used to track processing of the Item, especially if it fails.
- There is a property `ProcessCount` on the `ISMessageQueueItem` object.
  - This is a simple count of the number of times the Item has been processed.
- The `ISMessageQueueItemProcessLog` object contains the following properties:
  - `Status`
    - The Status noted when the Log was created.
  - `Data`
    - The Data you specified in `ProcessBegin` or `ProcessEnd`.
    - This could be a simple JSON string or a `ISKeyValueList` saved to XML.
  - `UserPk`
    - The Primary Key of the User who was processing the Item.
  - `UtcDate, Date`
    - The Utc and local Date and Time when the item was processed.
- A new Log is added in the functions `ProcessBegin` and `ProcessEnd` when the `processLogData` is non-blank.
  - This becomes the `Data` property in `ISMessageQueueItemProcessLog`.

Aborting a Message Queue Item

Any Item that has a `StatusResolved` of `iseMessageQueueStatus.Queued` or `iseMessageQueueStatus.ProcessingExpired` can be aborted.

This allows you to remove Items that are no longer valid.

- Call `AbortItem`.
  - Parameters:
    - `MessageQueueItem`
      - The Item to Abort.
    - `Optional newData As String = ""`
      - If not blank will update and replace the `Data` property.
    - `Optional newProperties As ISKeyValueList = Nothing`
      - If not Nothing will update and replace the `Properties` property.

Process Begin versus Database Transactions

The default is to fail `ProcessingBegin` if there is an active Database Transaction.

This ensures that the database is updated and there are no database locks in place that might block the ISQueue table from being read or updated.

Also, in most scenarios used internally by finPOWER Connect, it is likely that API calls to third parties will be made – and these will fail if there is an active Database Transaction.

However, there is an option to ignore this check. You might consider using this in some circumstances, but our recommendation is to not use this option.

- The `ProcessingBegin` function includes the parameter `allowInDatabaseTransaction`.
  - The default, False, will fail if there is an active Database Transaction.
  - Set to True to ignore this test.

Purge

You can purge Queue Items at any time.

- It is called automatically at the start of `GetItemsForProcessing`.
- Items with Terminal Statuses can be purged:
  - `iseMessageQueueStatus.Success`
  - `iseMessageQueueStatus.Failed`
  - `iseMessageQueueStatus.Aborted`
- Call `finBL.MessageQueueUtilities.Purge`.
  - By default, all Items with a Terminal Status last updated more than 7 days ago will be purged.
  - Parameters:
    - `Optional serviceId As String = ""`
      - The optional ServiceId to filter by.
    - `Optional days As Integer = 7`
      - The Number of Days before an Item is purged.
      - Default is 7 days.
    - `Optional ByRef recordsPurged As Integer = 0`
      - Returns the number of Items purged

Example code:

```
If finBL.MessageQueueUtilities.Purge(,,RecordsPurged) Then
  finBL.DebugPrintFormat("{0} records purged", RecordsPurged)
Else
  Success = False
End If
```