Sage 300 Web Screens SDK

Worker Processing

January 2021

The MIT License (MIT)

Copyright © 2019-2021 The Sage Group plc or its licensors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contents

[1. Introduction 4](#_Toc790300)

[2. Process Overview 5](#_Toc790301)

[2.1 Diagram 5](#_Toc790302)

[2.2 Overview 5](#_Toc790303)

[3. Update Overview 6](#_Toc790304)

[3.1 Diagram 6](#_Toc790305)

[3.2 Overview 6](#_Toc790306)

[4. Full Cycle Overview 7](#_Toc790307)

[4.1 Diagram (Run Mode) 7](#_Toc790308)

[4.2 Diagram (Debug Mode) 7](#_Toc790309)

[5. Workflow Tables 8](#_Toc790310)

[5.1 Database Diagram 8](#_Toc790311)

[5.2 WorkStatus Table 9](#_Toc790312)

[5.3 WorkflowKind Table 9](#_Toc790313)

[5.4 UnitOfWorkKind Table 10](#_Toc790314)

[5.5 WorkflowInstance Table 11](#_Toc790315)

[5.6 UnitOfWorkInstance Table 12](#_Toc790316)

[5.7 WorkflowSchedule Table 13](#_Toc790317)

[6. Web Role Components 14](#_Toc790318)

[6.1 ProcessService Base Class 14](#_Toc790319)

[6.2 Debugging a Process 14](#_Toc790320)

[6.2.1 External Developers 15](#_Toc790321)

[6.2.2 Internal Developers 15](#_Toc790322)

[7. Worker Role Components 16](#_Toc790323)

[7.1 WorkFlowManager 16](#_Toc790324)

[7.2 UnitOfWorkManager 16](#_Toc790325)

[7.3 Worker Service 16](#_Toc790326)

[7.4 WorkerDispatcher 16](#_Toc790327)

[7.5 UnitOfWorkWorker 16](#_Toc790328)

[7.6 UnitOfWorkProcessor 17](#_Toc790329)

[7.7 ProcessUow Base Class 17](#_Toc790330)

[7.8 ProcessingRepository Base Class 18](#_Toc790331)

1. Introduction

This document is intended to provide information on the processing and implementation of the workflow engine via a worker role.

Long running processes and any periodic processing functions in the application will be executed asynchronously via a worker role. This configuration is identical for both an On-Premise and a Cloud installation, but with different components (MSMQ vs. Azure).

Utilizing a worker role, the application can off-load a long running process while not affecting the interaction with the screen.

1. Process Overview
   1. Diagram

Worker Role

Queue

Web Role

Read

Write



Message



* 1. Overview

The user wants to start a long running process. We will talk about the database setup and the web and worker role components later.

The processing task has an already assigned Workflow Kind Id (GUID) associated with the process. This Id is sent to the web role components in order to create a message payload for the queue. The message payload is created and is added to the queue. For an On-Premise installation, the queue is MSMQ and a Cloud installation will use an Azure Queue.

The Landlord Database has workflow tables that will be updated at this time with the request and will be used to track process status (pending, processing, completed) and will also be used to identify which instance of a worker role is processing the request. These tables will be discussed later.

The Message Id (workflow instance id) will be returned to UI in order to allow the UI to poll the server to get a status of the process. This is explained in the next section.

There will also be a background Ajax long polling call which will poll the server periodically with the Message Id token to get the status of the process.

The worker role will pick up the request from the queue, evaluate the message payload, update the status in the Landlord database tables, get the process to be invoked from the Landlord database tables and then invoke that process via reflection.

If the process is being run in the IDE (debug mode), the process will remain in the web role with all updates, statuses and flow performing as if was running in the worker role.

Regardless of whether the process completed or if errors were encountered, the UI is updated with the appropriate information at the completion of the process. In addition to the UI being updated, the Landlord Database tables also have detailed information on the success of the process or detailed error stack information if an error condition was detected. Also, the “trace.log” file in the Worker folder will include error stack information if an error was encountered.

1. Update Overview
   1. Diagram

Worker Role

Web Role

Check DB or Service for Status





Status Returned

* 1. Overview

When the user requests the process status, the web role components will first check the status of the process in the Landlord Database tables. If the process has been successfully completed or failed, the information is returned to the UI without invoking the service.

If the status of the message is Processing, the web role components will invoke the service hosted on the worker role. The Message Id is passed to the service for lookup.

The worker role components will access the meter object in order to get the current status of the process for the UI. The invocation of the worker role components to get the status are synchronous.

1. Full Cycle Overview
   1. Diagram (Run Mode)

Worker Role

Queue

Web Role

Read



Message



ProcessService

WorkflowManager

UnitOfWorkManager

Windows/WCF Service

WorkerDispatcher

UnitOfWorkWorker

UnitOfWorkProcessor

ProcessUow

ProcessingRepository

* 1. Diagram (Debug Mode)

Queue

Web Role

Message



ProcessService

WorkflowManager

UnitOfWorkProcessor

ProcessUow

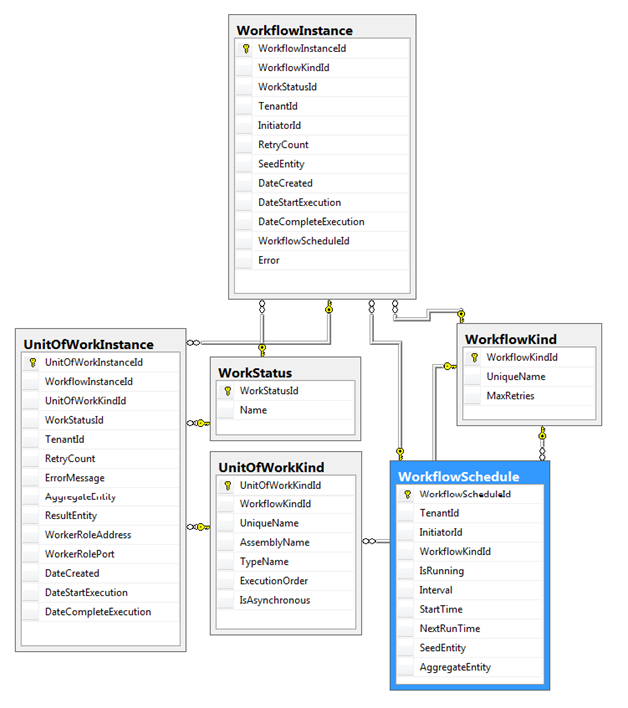
ProcessingRepository

Stays in the Web Role and creates message for queue, but instead of inserting into the queue, the Workflow Manager directly calls the Unit of Work Processor and allows process to remain in the Web Role

1. Workflow Tables

The Workflow tables are in the Landlord Database and are as vital of a component as the web role and the worker role components. They provide the liaison, in addition to the queueing service, between the web role and worker role.

* 1. Database Diagram



* 1. WorkStatus Table

The WorkStatus table stores the minor entity along with a description.

This table is created by the **Create\_WorkerRole\_Schema.sql** script and the contents are created by the **Insert\_WorkerRole\_Data.sql** script. Both scripts are located in the *runtime\Database\Landlord\Scripts* folder and are executed when the Database Setup Utility’s Portal Button is selected.

Example: INSERT INTO [dbo].[WorkStatus](WorkStatusId, Name) VALUES (1, 'Executing')

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Type | Description | Values |
| WorkStatusId | int | Id for the work status | 1, 2, 3 |
| Name | nvarchar(50) | Description of the work status | “Executing”, “Completed”, “Error” |

* 1. WorkflowKind Table

The WorkflowKind table stores the unique identifier (GUID) used to identify a process, or workflow kind, and to specify the number of retries allow for this process.

This table is created by the **Create\_WorkerRole\_Schema.sql** script and the contents are created by the **Insert\_WorkerRole\_Data.sql** script. Both of these scripts are located in the *runtime\Database\Landlord\Scripts* folder and are executed when the Database Setup Utility’s Portal Button is selected.

Additionally, this table will be populated with Partner and ISV processes. Thus, the need for the WorkflowKindId to be unique.

Example: INSERT INTO [dbo].[WorkflowKind](WorkflowKindId, UniqueName, MaxRetries) VALUES (CONVERT(uniqueidentifier, 'ab08bb64-73bc-4681-87fd-08dd23af21a9'), 'TUClearStatistics', 3)

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Type | Description | Values |
| WorkflowKindId | uniqueidentifier | Unique Id for the process | GUID |
| UniqueName | nvarchar(50) | Description of the process | The pattern used is {module} + Description with no spaces (i.e. “TUClearStatistics”) |
| MaxRetries | int | Number of retries for process | This is usually set to 3 |

* 1. UnitOfWorkKind Table

The UnitOfWorkKind table stores one or many sub-tasks (unit of work) for the process. There will be at least one row in this table for every row in the WorkflowKind table. This table stores the details regarding the unit of work such as the assembly and type name in order to process the unit of work, the execution order if there are multiple sub-tasks and an asynchronous/synchronous flag.

This table is created by the **Create\_WorkerRole\_Schema.sql** script and the contents are created by the **Insert\_WorkerRole\_Data.sql** script. Both of these scripts are located in the *runtime\Database\Landlord\Scripts* folder and are executed when the Database Setup Utility’s Portal Button is selected.

Additionally, this table will be populated with Partner and ISV processes. Thus, the need for the WorkflowKindId to be unique.

Example: INSERT INTO [dbo].[UnitOfWorkKind](WorkflowKindId, UniqueName, AssemblyName, TypeName, ExecutionOrder, IsAsynchronous)

VALUES(CONVERT(uniqueidentifier, 'ab08bb64-73bc-4681-87fd-08dd23af21a9'), 'TU Clear Statistics', 'ValuedPartner.TU.Services', 'ValuedPartner.TU.Services.UnitOfWork.ClearStatisticsUow', 1, 1)

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Type | Description | Values |
| UnitOfWorkKindId | int | Unique Id for the table | Auto-generated |
| WorkflowKindId | uniqueidentifier | Unique Id for the process | GUID |
| UniqueName | nvarchar(50) | Description of the process task/sub-task | The pattern used is {module} + Description with spaces (i.e. “AR Clear Statistics”) |
| AssemblyName | nvarchar(200) | Assembly to search via reflection for type | Sage or third party binary |
| TypeName | nvarchar(100) | Type name for unit of work | Based upon ProcessUow |
| ExecutionOrder | int | Order if multiple sub-tasks | 1-n |
| IsAsynchronous | bit | Flag for synchronicity | 1 if async otherwise 0 |

* 1. WorkflowInstance Table

The WorkflowInstance table stores information pertinent to the workflow kind (process).

This table is created by the **Create\_WorkerRole\_Schema.sql** script and the contents are created by the worker role components when the worker role picks up the process from the queue. The script is located in the *runtime\Database\Landlord\Scripts* folder and is executed when the Database Setup Utility’s Portal Button is selected.

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Type | Description | Values |
| WorkflowInstanceId | int | Unique Id for the table | Auto-generated |
| WorkflowKindId | uniqueidentifier | Unique Id for the process | GUID |
| WorkStatusId | int | Status of task | 1=Executing, 2=Completed, 3=Error |
| TenantId | uniqueidentifier | Tenant initiating process | GUID |
| InitiatorId | uniqueidentifier | User initiating process | GUID |
| RetryCount | int | In case of failures, the number of retries before success or eliminated by failure | 0-MaxRetries in WorkflowKind table |
| SeedEntity | nvarchar(max) | Context and Model information to supply to Execute method of ProcessUow type | XML Serialized string |
| DateCreated | datetime | Date record created |  |
| DateStartExecution | datetime | Start date and time of process/sub-task |  |
| DateCompleteExecution | datetime | Complete date and time of process/sub-task |  |
| WorkflowScheduleId | int | If process/sub-task is scheduled, the schedule id | Null or schedule id |
| Error | nvarchar(max) | Error details | Null or error stack |

* 1. UnitOfWorkInstance Table

The UnitOfWorkInstance table stores information pertinent to the unit of work (process/sub-task).

This table is created by the **Create\_WorkerRole\_Schema.sql** script and the contents are created by the worker role components when the worker role picks up the process/sub-task from the queue and marks it as executing and when it is completed, the worker role looks for the next item in the execution order. The script is located in the *runtime\Database\Landlord\Scripts* folder and is executed when the Database Setup Utility’s Portal Button is selected.

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Type | Description | Values |
| UnitOfWorkInstanceId | int | Unique Id for the table | Auto-generated |
| WorkflowInstanceId | int | Unique id for WorkflowInstance table |  |
| UnitOfWorkKindId | int | Unique id for UnitOfWorkKind table |  |
| WorkStatusId | int | Status of task | 1=Executing, 2=Completed, 3=Error |
| TenantId | uniqueidentifier | Tenant initiating process | GUID |
| RetryCount | int | In case of failures, the number of retries before success or eliminated by failure | 0-MaxRetries in WorkflowKind table |
| ErrorMessage | nvarchar(max) | Error message if any | Null or error stack |
| AggregateEntity | nvarchar(max) | Future Use |  |
| ResultEntity | nvarchar(max) | Result if WorkStatusId == 2 | XML Serialized Result |
| WorkerRoleAddress | nvarchar(max) | Address of worker role |  |
| WorkerRolePort | nvarchar(max) | Port used by worker role |  |
| DateCreated | datetime | Date record created |  |
| DateStartExecution | datetime | Start date and time of process/sub-task |  |
| DateCompleteExecution | datetime | Complete date and time of process/sub-task |  |

* 1. WorkflowSchedule Table

The WorkflowSchedule table stores information pertinent to a scheduled workflow kind (process).

This table is created by the **Create\_WorkerRole\_Schema.sql** script and the contents are created as needed to schedule a process along with a queue entry. The script is located in the *runtime\Database\Landlord\Scripts* folder and is executed when the Database Setup Utility’s Portal Button is selected.

The table provides the flexibility of scheduling a process. When the execution is complete, the worker role will retrieve details from the schedule and get the next scheduled time of execution. If there is to be another scheduled run, a new workflow and unit of work will be created and also inserted into the queue.

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Type | Description | Values |
| WorkflowScheduleId | int | Unique Id for the table | Auto-generated |
| TenantId | uniqueidentifier | Tenant initiating process | GUID |
| InitiatorId | uniqueidentifier | User initiating process | GUID |
| WorkflowKindId | uniqueidentifier | Unique Id for process | GUID |
| IsRunning | bit | Is scheduled task running | 1 = true other 0 |
| Interval | bigint | How often to run the scheduled task |  |
| StartTime | datetime | Start date and time of scheduled process |  |
| NextRunTime | datetime | Next start date and time of scheduled process |  |
| SeedEntity | nvarchar(max) | Context and Model information to supply to Execute method of ProcessUow type | XML Serialized string |
| AggregateEntity | nvarchar(max) | Future Use |  |

1. Web Role Components

This section will document the web role components.

* 1. ProcessService Base Class

Every Process task/screen has a service class which inherits from the ProcessService base class. This class has a Process method that is invoked when the user presses the “Process” button on the UI.

The service class is passed the WorkflowKind Id in the constructor. This id is a GUID to ensure uniqueness in the Landlord Database Workflow tables.

Example:

/// <summary> AR Clear Statistic </summary>

public static readonly Guid ARClearStatistic = new Guid("e88f769f-dba1-4c49-9a75-b61a5d83f137");

The Process method in the base class is responsible for:

* Creating a handle to the queue based upon the configuration in the web.config file for On-Premise (MsmqQueueFactory) or Cloud (AzureQueueFactory) installation. The queue name is also retrieved from the web.config file.
* Retrieves an instance of the Workflow Manager
* Serializes the model data for the SeedEnitty columns in the workflow tables
* Creates and queues the unit of work
  + WorkflowManager.InternalCreateAndQueueFirstUnitOfWork
* Returns the WorkFlowInstanceId to the UI
  1. Debugging a Process

Debugging a Process in the Visual Studio has been difficult at best due to part of the process running in the Web Role while the other part of the process is running in the Worker Role.

Debugging was previously accomplished with either Logger statements or attaching to the IIS process. Both options involved shutting down the Sage.CNA.WindowsService, deploying the assemblies to your local installation’s Worker folder, re-starting the service and then either watching the log file or stepping through via the attached process.

As of Sage 300 2019.2, debugging was made a lot easier. The Workflow Manager has some added code to detect if the process is being run in the Visual Studio IDE and if it is:

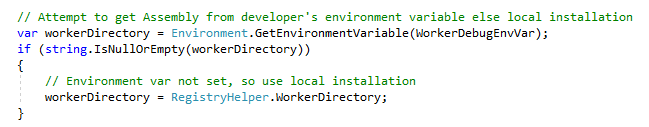
* Does not add the message payload to message queue
* Instantiates, via Activator.Create, the Worker Processor assembly from the …\Online\Worker folder
* Passes message payload and Workflow Manager to processor
* Starts the worker process just as the Worker Dispatcher would have done, but now running in the IDE in the web role
* All Landlord database communication and flow is unaffected
  + 1. External Developers

There are no code changes or other requirements for external developers (partners). That’s it!

* + 1. Internal Developers

Internal developers have access to the framework source and if changes to the Worker Processor objects are being made, it will be cumbersome to manually copy these assemblies to the Worker folder to test the changes. Therefore, the Workflow Manager will first look for the presence of an Environment Variable. If detected, it will use that location for the Worker Processor as opposed to the locally installed Worker folder.

* Environment Variable
  + “Sage300WorkerDebugDir”
* Environment Value
  + Directory for the Worker Processor assembly which will typically be the Assemblies folder OR the Web project’s bin folder



1. Worker Role Components

This section will document the worker role components.

* 1. WorkFlowManager

This class is used by both the web and worker roles. This is a helper class for the Landlord Database tables interactions.

* 1. UnitOfWorkManager

This is a helper class for the Landlord Database UnitOfWorkInstance and UnitOfWorkKind table interactions.

* 1. Worker Service

The service is a self-hosted service which exposes some methods (GetStatus and CancelWork) for the web role. The worker role creates this service on startup.

Also, in an On-Premise installation, this service is stopped and re-started via the Database Setup Utility’s Portal Button.

* 1. WorkerDispatcher

This class in the Worker project is the queue listener and is responsible for dispatching the work. The dispatcher uses the UnitOfWorkWorker class to monitor the queue and to retrieve the process/task to be executed.

The listener monitors the queue at regular intervals for new messages. When the queue is empty, the listener continues polling the queue and even slows down by entering a sleep state.

The current sleep state is 5 seconds when the queue is empty. If the queue is not empty, the dispatcher will de-queue the message and will create tasks that will be executed asynchronously.

The dispatcher supports multi-threading and multiple tasks can be created with 5 tasks running concurrently.

* 1. UnitOfWorkWorker

This class in the Worker project helps the dispatcher class in retrieving the next message from the queue and also assists in executing the tasks.

It uses either an MSMQ Queue or Azure Queue depending upon the installation determined in the web.config file. It uses the UnitOfWorkProcessor class to execute the work.

* 1. UnitOfWorkProcessor

This class in the Worker project processes the queue message.

* UnitOfWorkProcessor.DoWork
  + Processes the queue message
  + Retrieves the UnitOfWorkInstance table details
  + Executes the unit of work
* UnitOfWorkProcessor.ExecuteUnitOfWork
  + Retrieves the UnitOfWorkKind table details
  + Determines type to be invoked from either local Sage unit of work type or from an external assembly via reflection for Partners and/or ISVs
  + Retrieves the WorkFlowInstance table details
  + Instantiates unit of work assembly via reflection (Activator) to get unit of work object
  + Updates execution start time in UnitOfWorkInstance table
  + Invokes Execute method of unit of work class for processing
  + Updates UnitOfWorkInstance table for completion time and status
  1. ProcessUow Base Class

Every Process task/screen has a service class in the UnitOfWork folder of the Services project which inherits from the ProcessUow base class. This class has an OnExecute method that is invoked when the worker role’s UnitOfWorkProcessor.ExecuteUnitOfWork method run.

The unit of work class’s constructor is passed numerous pieces of information for it to execute the processing logic.

Example:

public ClearStatisticUow(

WorkflowInstance workflowInstance, UnitOfWorkInstance unitOfWorkInstance,

IQueue queue, Action keepAlive, IUnityContainer container) :

base(workflowInstance, unitOfWorkInstance, queue, keepAlive, container) { }

The OnExecute method in the base class is responsible for:

* Retrieving the see entity data from the WorkFlowInstance table
* Instantiating the repository class belonging to the Process
* Invoking the Process method of the repository
* Returning a ProcessResult message
  1. ProcessingRepository Base Class

Every Process task/screen has a repository class in the BusinessRepositiory project which inherits from the ProcessingRepository base class. This class has a Process method that is invoked when the worker role’s ProcessUow.OnExecute method run.

The Process method in the base class is responsible for:

* Creating any business entities needed by the process
* Checking rights for the process
* Invoking the entity’s Process method (wrapper of Accpac business view)
* Returning a model along with any warnings and errors