|  |  |  |
| --- | --- | --- |
| Database Activity Logging Infrastructure | August 23  2016 | |
| This document details out the overall architecture of logging infrastructure its usage in scripts, database processes etc & its pros & cons. | |  |

Table of Contents

[Introduction 3](#_Toc459719067)

[Objective 3](#_Toc459719068)

[Logging Infrastructure Overview 3](#_Toc459719069)

[Logging Infrastructure Work Flow 4](#_Toc459719070)

[Severity Constants 4](#_Toc459719071)

[Logging Tables 4](#_Toc459719072)

[1. ActLog.ActivityContext 4](#_Toc459719073)

[2. ActLog.ActivityDetail 4](#_Toc459719074)

[Logging Tools 5](#_Toc459719075)

[1. ActLog.LoadActivityContext 5](#_Toc459719076)

[2. ActLog.UnLoadActivityContext 5](#_Toc459719077)

[3. ActLog.LogActivity 5](#_Toc459719078)

[Detailed Logging Example 6](#_Toc459719079)

# Introduction

The logging infrastructure is a solution which helps to keep track of database processes with ease. Provides a way to monitor & measure performance of any running processes.

# Objective

The objective of this document is to detail out the overall logging infrastructure. Its usage in database scripts, processes & routines. How it can provide efficient way to monitor & review the performance & activity of any process execution in database front.

# Logging Infrastructure Overview

# Logging Infrastructure Work Flow

The logging infrastructure is nothing but the collection of tools (small routines) & couple of tables which provides an efficient way to monitor & manage all your database activities within ease. Let’s see & review them now.

## Severity Constants

There are few key constants which are used in this logging mechanism to show the severity of the error occurred. The key constants are listed below.

1. LOG\_FATL (1) - This constant is used for fatal errors like environmental issue, tempDB log space issue etc.
2. LOG\_CRIT (2) - This constant is used to show critical errors like show stopper errors, failure of process.
3. LOG\_ERR (3) - This constant is used to show errors like some prerequisites are non handled.
4. LOG\_WRN (4) - This constant is used to show warnings like some step is taking more than expected time.
5. LOG\_INFO (5) - This is constant is used to inform that a log message is just for informational purpose.
6. LOG\_DBG (6) - This constant is mostly used by developers to execute debug blocks in processes that they can investigate the issues.

## Logging Tables

### ActLog.ActivityContext

This table stores all the information related to the start & end of the process execution in a way that all the steps in that process should be covered in start & end logged in ActLog.ActivityContext table.

e.g.

|--- > Start process send Notification

Param1 : ‘ac@yahoo.com’

Param2: ‘email body’

Param3: ‘Priority’

Pulling database mail profiles

Pulling SMTP details

|< --- Stop process send Notification

The highlighted text in above logging sample is the start (|-- >) & end (|< --) of the process. This information is stored in ActLog.ActivityContext table.

### ActLog.ActivityDetail

This table is most important as it stores all the details of the process in execution.Taking previous example now with additional details will provide usage of this table.

e.g.

|--- > Start process send Notification

Param1 : ‘ac@yahoo.com’

Param2: ‘email body’

Param3: ‘Priority’

Pulling database mail profiles 00:00:00:014

Pulling SMTP details 00:00:00:012

Send email 00:00:00:200

|< --- Stop process send Notification

The above highlighted details i.e. the message & time logged for every step performed in Send Notification routine is captured & stored in log. Message table. This table will always maintain logging details for all the processes unless it is explicitly cleaned. So a User can also do compare on the amount of time taken by each run for that same process & workout which step is causing process to take more time.

## Logging Tools

These are the CLR functions which are used to perform logging.

### ActLog.LoadActivityContext

This tool is used to push the process startup context to the stack. (The table ActLog.ActivityContext will be used as stack).

e.g. |--- > Start process send Notification

There is a key LogID which is unique & should be allocated to every process so that process & logging associated with it can be easily identified.

Below are the inputs which need to be provided to ActLog.LoadActivityContext.

* LogID
* Context Name
* Severity

**Syntax:** ActLog.LoadActivityContext LogID, ContextName, severity.

### ActLog.UnLoadActivityContext

This tool is used to push the process startup context to the stack. (The table ActLog.ActivityContext will be used as stack).

e.g. |--- > Stop process send Notification

The key which is used to push the context needs to be used to pop the context. The inputs for this tool is same as for ActLog.LoadActivityContext.

* LogID
* Context Name
* Severity

**Syntax:** ActLog.UnLoadActivityContext LogID, ContextName, severity.

### ActLog.LogActivity

This tool is used to load messages which needs to be logged for each & every step of the process. This tool provides user with an ability to store details about any resultsets, key columns details, schema details which can be reviewed by User. This feature saves user from spending time in executing the whole process again & again & prepare the required resultset. The xml dumps storage is also supported by this tool.

The input needed for ActLog.LogActivity are listed below. Please note the same LogId should be used which is used for push & pop context. This helps to structure the log activity in user friendly & user readable format.

* LogID
* Message text
* Severity
* ResultSetTable (optional)
* ResultSetShcema (optional)
* ResultSetFieldClause (optional)

**Syntax:** ActLog.LogActivity LogID, Message Text, Severity

## Detailed Logging Example

Below example gives the overall Idea about the logging of major process which has small process called internally.

The process **PP** highlighted in red is the main process & it has internal processes **QQ** & **RR.**

1=00:00.032 This is time since first context PP started execution

2=00:00.018 This is time since second context QQ started execution

3=00:00.007 This is time since third context RR started execution

With the above way of time logging one can easily find out the amount of time taken by each process context or time taken by each step in given process, like (3=00:00.007) of message 1 shows it took 7 milliseconds by this step & (3=00:00.0022) of message 2 shows it took 15 milliseconds by this step (subtract 7 milliseconds by previous step)

|--> PP

|--> QQ

|--> RR

1=00:00.032 2=00:00.018 3=00:00.007 Put a message1

1=00:00.047 2=00:00.033 3=00:00.0022 Put a message2

|<-- RR

1=00:00.070 2=00:00.042 After pop (RR)

|<-- QQ

1=00:00.000 After pop(QQ)

1=00:00.000 now pop(PP)

|<-- PP