

The Application Log

The Application Log in SAP R/3 is very interesting, as it can be used for many different purposes, and as many standard solutions uses the application log. In the myProcess Manager I use the application log for storing:

- messages (errors, warnings, info)
- events
- traces
- changes

Any application log entry that are stored by the myProcess Manager will have the Application log object and sub object, that you configure for your subsystem in myProcess Manager.

The myProcess Manager also save the Process Document number as the External Id in the application log. The External Id is very important, as it ensures the best performance when reading the Application Log. The **External id** in the application log is filled with the Message Handler field "UnitOfWork" and contains the unique process document number.

Combined with the ZCL_MESSAGEHANDLER the application log gives you a unified and simple way of handling messages as errors, warnings, information, events, traces and change log. The end user will find it handy as the solution is common for all your own developments and for many standard programs.

Monitoring of messages uses the same set of data and a common way of displaying data.

The Application Log Class

Why create the ZCL_APPLICATION_LOG class, when SAP R/3 has lot of classes, programs and function modules that are working with the application log.

Here are some of the arguments that answers the question

- Encapsulating the standard SAP and hide it from all projects that uses myProcess Manager or other of my solutions.
- I collect all the best methods that I like from the standard SAP, no matter if it is class based or implemented as classic Abap. We therefore have one class that

substitutes several classes and function modules.

- I add my own solutions where I dislike the standard or where I do not find a standard solution.
- I hide the technique and give the Abap programmers a common interface to the standard SAP application log. If the programmers always uses the same class, this class will be used as part of the Abap language.
- The programmers do not need to know all the details of the application log, only how to use it through the common interface.
- When upgrading standard SAP, I do not need to change every own developed solution, but I can concentrate on upgrading the ZCL_APPLICATION_LOG.
- Whenever you like to enhance the class, any new method or new implementation have impact on all programs or systems that uses the class.
- Encapsulation of a standard class gives you an possibility to change the class behaviour like changing methods from static to instance or change the naming convention towards your standard or the naming of the methods in your favourite style.

When looking at the standard solutions in SAP that support the application log you will be surprised, as there are tons of code, some more useful than others, but how should a programmer be able to choose the right function or class method when thousands possible solutions exists. This can of course be done but it's time consuming.

And what happens when SAP AG is changing some of the code and you used different solutions in your own developed code. That's why folks, you need to build your own class that acts as an interface between your own developed solutions and standard SAP solutions.

Development life cycle

It's very important to understand the lifecycle of a class. When you first create the class, you do not develop more methods than you really need. The methods you need could as a start contain calls of your own classic Abap code or some old standard

SAP code. It does not matter, what's important is that you spend time on the interface, trying to nail it in the first place.

The implementation can always be changed without impact on the programs that uses the class. Later on in the development cycle, step back and look at your class from distance and ask, what should you be able to do with this class. For the application log it could be something like:

- It would be nice to be able to configure the application log as a surrogate to using transaction SLG0.
- I need methods to create single and multiple application log messages
- I need methods for displaying single and multiple application log messages
- I would like to be able to change some of the class attributes on the fly
- It would be nice to be able to clean up messages from the application log

And the development life cycle continues. I'm thinking of building some kind of BI or reporting tool that analysis the application log messages. But I think that SAP AG already have build a BI solution. In the future I would dig into that world, and see if there are some think that could be used for enhancing the class.

As you see, the class life cycle really never ends, therefore you need to document your classes, in a way that the next programmer knows your thought behind the solution.

GoF Design Patterns

In the myProcess Manager, I try to follow the design rules of the GoF Design Patterns, originally introduced by the Gang Of Four. In the application log I uses the Design Patterns:

FACADE Design Pattern

The ZCL_APPLICATION_LOG provide a unified interface to a set of interfaces in a subsystem. The class defines a higher-level interface that makes the subsystem easier to use.

ADAPTER Design Pattern

The ZCL_APPLICATION_LOG convert the interface of a class into another interface that the client expect. Adapter lets classes work together that

could not otherwise because of incompatible interfaces.

DECORATOR Design Pattern

The ZCL_APPLICATION_LOG class attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub classing for extending functionality.

PROXY Design Pattern

The ZCL_APPLICATION_LOG class provides a surrogate or placeholder for another object to control access to it.

The ZCL_APPLICATION_LOG class is mostly a facade and adapter, but does also provide extended functionality.

The main intention is the encapsulation of standard SAP functionality. The standard SAP functionality is implemented as function modules, transactions and diverse classes.

If your SAP site always uses this ZCL_APPLICATION_LOG class instead of SAP provided functionality, an later upgrade will become easier as you only have to check the interface between the ZCL_APPLICATION_LOG class and the standard R/3 functionality. You will not have to walk through all customer developed applications, as changes in SAP, will have no impact as long as the interface between customer application and the ZCL_APPLICATION_LOG class is not changed.

Our own class uses nearly all methods from standard class CL_RSO_APPLICATION_LOG, but where the standard class uses static methods our implementation uses instance methods.

Instantiation

This class uses public instantiation, meaning that the client must have created an instance of the class to use the class methods.

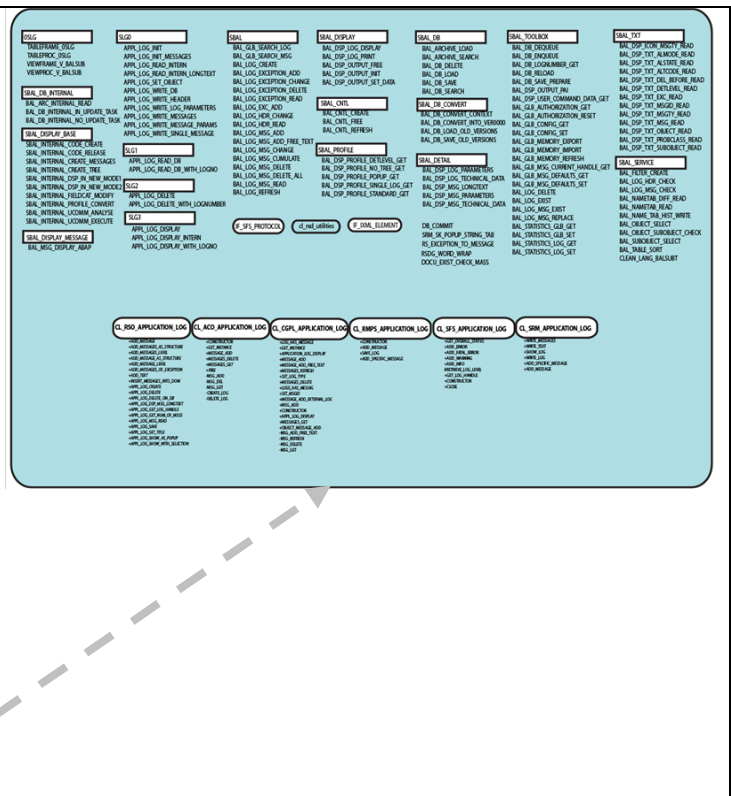
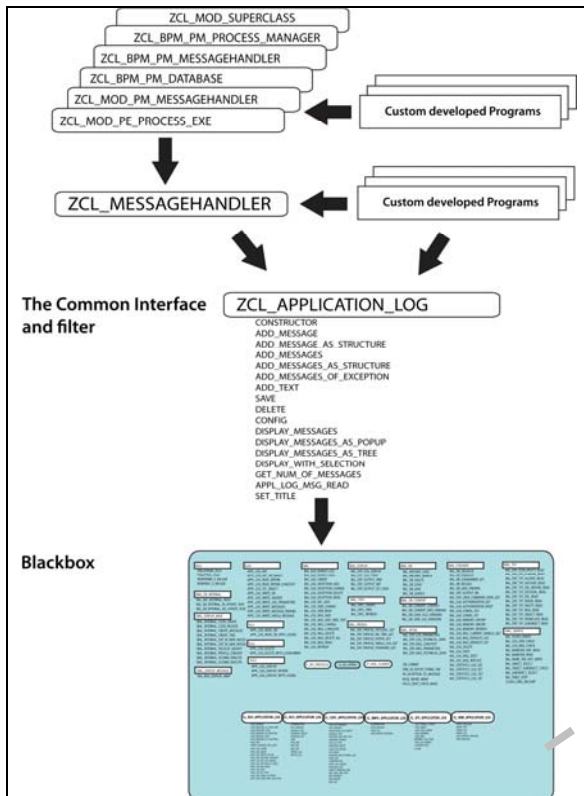
Methods

This class contains two static methods, CONFIG() ,DISPLAY_WITH_SELECTION() but if you need further methods you are free to make enhancements and lot of the standard code could be used. For the myProcess Monitor we only need the

instance methods as we always are instantiating the class.

As SAP provides more functionality than we need for the myProcess Manager we can relax and find all methods among the standard solution.

We need to be able to instantiate the class create/add messages to the log and we need to be able to display the application log. It would also be nice if we were able to configure application log from within our own solution, without the need of [SLG0].

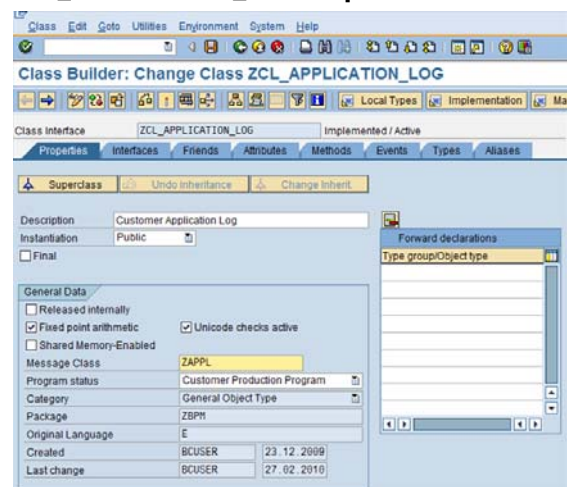


As the picture shows, it is rather complex when having many but not all the standard functionality documented. The picture also shows how nice the common interface encapsulations and hide all the details for business process manager and other customer developed solutions.

Now it should be easily understood that by creating our own application log class we gain a simplified interface that could be our companies unified way of using the application log. The user gain an equal quality in all class methods, function module and programs that uses the customer developed appl. log class.

In the following I show the class implementation, through the properties, the attributes and all the methods:

ZCL_APPLICATION_LOG Properties

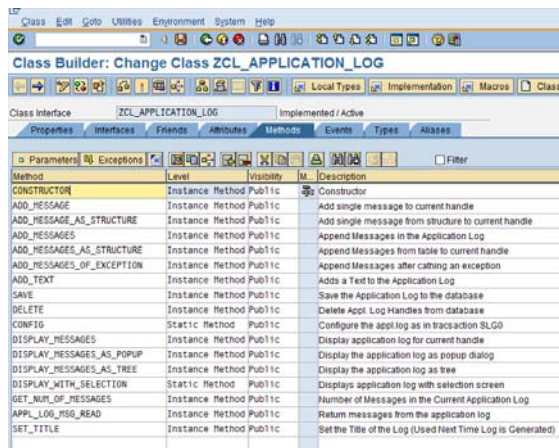
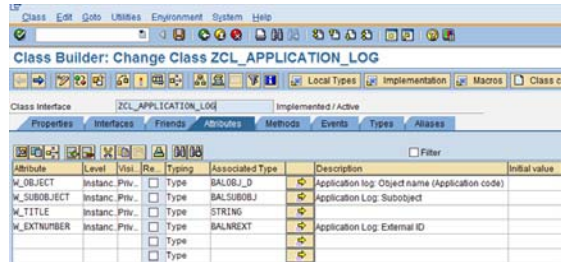


In the class properties we set the instantiation to public. There could be arguments for change the

instantiation to private and create a class with singleton design patterns and a factory method. Then you would be sure that you do not have more than one instance per session. For our goal it's acceptable to use public instantiation.

Remember to fill in appropriate Message Class and Program status.

We only have few attributes, all with level instance and all with visibility as private. In general all attributes should be private, you are better of creating methods that manipulates your attribute than exposing your attributes as public.



We do not need more methods than these and many are implemented using standard solutions. In general all the methods are instance methods we only have two static methods. One for making configuration and on for displaying application log that already exists in the database. All the methods have visibility as public. When you are finish with your own class, you should consider the methods visibility. Often you will end up with few public methods some protected and some private. In this case we are building a tool, that should be used for many different purpose, therefore it makes sense to keep the methods as public methods.

Publishing the implemented code would exceed the purpose of this article. But I will properly in time publish the code under the download section of the website.