How to use bgPF?

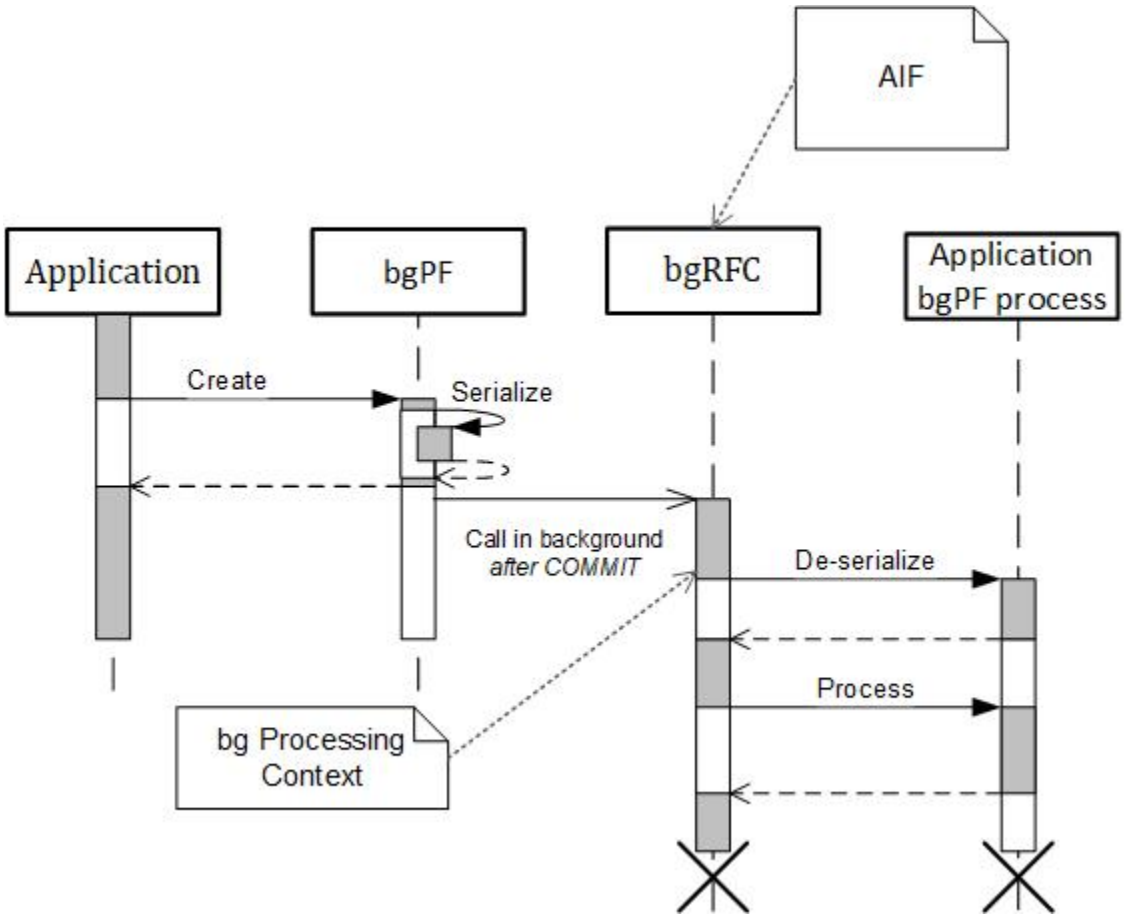
1. What is the background Processing Framework (bgPF)?

You probably know bgRFC. If we ignore the "R" (remote), then the bg(R)FC allows to trigger asynchronous processing in the background. This decoupling improves performance of the leading SAP LUW and makes it more robust. The bgPF is the modern ABAP Cloud variant of the bgRFC which combines many qualities:

- **Consistency**
  Transactional consistency and robustness are essential to ensure data consistency. The bgPF helps to outsource critical coding from actual SAP LUW. Within bgPF transactional consistency can be ensured if needed.
  bgPF supports transactional background processing with service quality "exactly once". An application can provide its own execution context object, to allow e.g. customers to define application specific background process priorities. A default execution context object is part of the bgPF.
  bgPF supports also queued background processing with service quality "exactly once in order". For this the application must provide its own execution context object. This is needed because there is no mechanism to guarantee uniqueness of queues.
- **End-user Performance**
  Better response time for actual SAPLUW, if post-processing can run in a different session asynchronously.
- **Scalability**
  For scalability of an application, it is better to split big processing steps into smaller chunks of work which can be distributed across work processes. The application server will use CL_SSI_DISPATCH in later releases to scale, this is done with the new bgRFC scheduler. Only if application coding is extracted and executed asynchronously in smaller chunks, then mechanisms like auto-scaling can be applied.
- **Tooling integration**
  ADT Cross Trace, AIF, bgRFC monitor can help you during development or later for monitoring your application.

Frameworks like e.g. OData and Events use bgPF for asynchronous and reliable execution.

1.1. How does it works?



- The first step for an application is to implemented one of our interfaces in an own class. At runtime the application calls the bgPF with an instance this class.
- Internally the bgPF serialize the class and call the bgRFC.
- The next COMMIT will trigger the bgRFC and a new ABAP session will start in the application server.
- bgPF will deserialize your class with all the data and will call the execute method of you implementation.

The AIF integration is optional.

2. When to use bgPF and when to use local business events?

Business events are modeled entities with a clear semantic (e.g. SalesOrderCreated, SalesOrderAccepted). We distinguish between provider and consumer. Typically these are different parties.

bgPF is used to execute business logic in the background. This has no modeling aspects and the data which is required is very specific for the concrete use case.

Both benefits from decoupling the background processing from the leading LUW, by improving the leading LUW's performance and robustness.

3. How to create your own implementation?

You have to implement interface IF_BGMC_OP_SINGLE or IF_BGMC_OP_SINGLE_TX_UNCONTR. The implementation of the AIF interfaces are optional.

Your operation implementation needs some data to do the processing in the background. It is up to you how you transfer the data to your object. E.g. our sample implementation (CL_BGMC_TEA_OPERATION) uses the constructor.

```
->set_operation( NEW CL_BGMC_TEA_OPERATION_TX_CON( exporting es_data = value #( ... )  )
```

3.1. Interfaces

1. IF_BGMC_OP_SINGLE
2. IF_BGMC_OP_SINGLE_TX_UNCONTR
3. IF_BGMC_OPERATION_AIF & IF_BGMC_OPERATION_AIF_CONF

3.1.1. IF_BGMC_OP_SINGLE (default) - Transactional control

The transaction controlled interface should be implemented per **default**. ABAP will make sure, that the basic rules of the SAP LUW are followed. We introduced the first checks in RAP and extracted the minimal restrictions which are valid in all ABAP transactions into something RAP-independent which is now also natively integrated into bgPF:

- Modify phase
  - No DB modifications on the primary connection are allowed.
  - (Implicit) DB-COMMIT is allowed. (Be aware that http communication in ABAP does an implicit DB-COMMIT)
  - no update task function module allowed
- Save phase (started via call of CL_ABAP_TX=>SAVE() in your implementation)
  - DB modifications are allowed.
  - update task function modules are allowed
  - No (implicit) DB-COMMIT is allowed.

Violations of the transactional contract and will lead to dump (or be logged). The advantage of the transaction controlled interface is that implicit database commits that can are apply by calling other parts are forbidden and controlled by the ABAP runtime. ACID rules that must be fulfilled for the SAP LUW are guaranteed.

Hint: The checks are done if checkpoint group CC_STMT is set to "Abort" (tx SAAB). This check is enabled in Steampunk and S/4 development systems. More details here: Checkpoint Groups - Steampunk - Wiki@SAP

3.1.2. IF_BGMC_OP_SINGLE_TX_UNCONTR - Without transactional control

The transaction uncontrolled interface should be implemented, when you can't follow the rules of the SAP LUW. This is only applicable in specific scenarios - e.g. using the XCO library to activate several DDIC / CDS artifacts where several COMMITs occur (which cannot be changed). An operation, that is not transactional controlled can be implemented freely. E.g. you can do COMMIT WORK or ROLLBACK WORK inside your implementation.

3.1.3. IF_BGMC_OPERATION_AIF* - AIF interfaces (Optional)

For an AIF integration you need to implement two interfaces:

IF_BGMC_OPERATION_AIF - Is used to transfer data to AIF and vice versa. GET_INPUT() and SET_INPUT() needs to be implemented.
IF_BGMC_OPERATION_AIF_CONF - This configuration (design time) interface for operations must be implemented to create a data container for AIF. GET_INPUT_CONTAINER() needs to be implemented.

3.2. Sample implementation

- CL_BGMC_TEA_OPERATION_TX_CON - Implementation of the transaction controlled interface
- CL_BGMC_TEA_OPERATION_TX_UNCON - Implementation of the transaction uncontrolled interface
- CL_BGMC_TEA_OPERATION - Implementation of AIF interfaces.
- CL_BGMC_TEA_APPLICATION is a sample application that use the bgPF to execute the implementation CL_BGMC_TEA_OPERATION*.

3.3. Can I do an automatic retry if an error happen?

In specific error situation (like e.g. when a database LOCK cannot be set) it is possible for an operation to trigger a future retry of the background process when it raises an exception. In your implementation you have to set the attribute TYS_RETRY_SETTINGS of the exception CX_BGMC_OPERATION. The bgPF with your operation will be scheduled for a retry.

- A retry can be attempted up to 3 times.
- It currently only works for queued background processes.

Automatic Retry

```
" In your implementation of IF_BGMC_OP_SINGLE_TX_*

method if_bgmc_op_single_tx_contr~execute.
..
    raise exception new cx_my_bgPF_exception( textid         = cx_my_bgPF_exception=>t100_lock_problem
                                              retry_settings = value #( do_retry = abap_true ) ).
endmethod.
```

3.4. How to call a bgPF

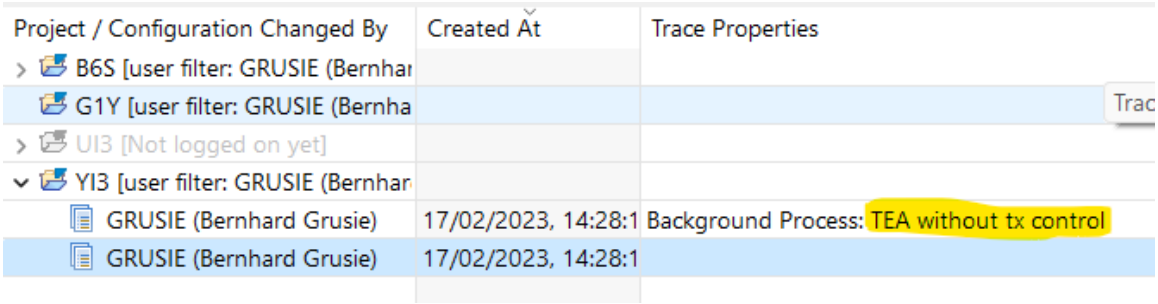
```
cl_bgmc_process_factory=>get_default(
                          )->create(
                          )->set_name( '<YOUR_TEXT>'
                          )->set_operation(    NEW <YOUR_IMPLEMENTATION_OF_IF_BGMC_OP_SINGLE>(   exporting
es_data = value #( ... ) )
                        //  )->set_operation_tx_uncontrolled( NEW
<YOUR_IMPLEMENTATION_OF_IF_BGMC_OP_SINGLE_TX_UNCONTR>( exporting es_data = value #( ... ) )
                          )->save_for_execution( ).

 COMMIT WORK. " If you use bgPF inside of RAP a commit will be done from RAP runtime. Without a commit, your
background operation will not start
```

An application instantiates synchronously a bgPF. Then the operation is added to the bgPF process. A save_for_execution() saves the process for later execution. Later on an explicit or implicit COMMIT triggers the execution of the new application session.

The default background processing context (get_default()) is shared with other applications and cannot be configured for specific applications. Or you use y our own bg Processing Context with: cl_bgmc_process_factory=>get('<NAME>'). For queued execution see How to use a Queue? To create an own Processing Context, see Own Processing Context. See also the current limitation in: bgPC_ADT

Method set_name('<YOUR_TEXT>') is optional. In the picture below we used "TEA without tx control". The use-case is to add runtime information e.g. "SalesOrder-42" to distinguish between different instances of SalesOrders calls of the bgPF. There is no AIF integration of this information so far.

When you use bgPF in RAP the method SAVE_FOR_EXECUTION() in this interface can only be called in the late save, see bgPF in RAP



3.4.1. Process Monitor

Method IF_BGMC_PROCESSSAVE_FOR_EXECUTION returns a monitor instance for the current background process. For most use cases this monitor is not needed.

It is intended for applications that have their own "book-keeping" and monitoring. Such an application might do something like this:

- In the current session it starts a background process and store the corresponding monitor instance in its bookkeeping table.
  It can use method IF_BGMC_PROCESS_MONITORTO_STRING to serialize the instance.
- At a later point in time (in a new session) the application needs to check the state of the background process.
  It re-instantiates the monitor instance via CL_BGMC_PROCESS_FACTORYCREATE_MONITOR_FROM_STRING and uses it to check the state of the background process.

3.4.2. How to use a Queue?

See limitation in: bgPC_ADT

To use the service quality "Exactly Once and In Order" (EOIO) , you have to use the queue method in the factory. The queue name is freely choosable from you. Our recommendation would be to use the key of your object as queue name. To avoid a clash for the queue name with other applications, you must us an own background Processing Context. Therefore, the default background Processing Context can't be used with a queue.

Queue in bgPF

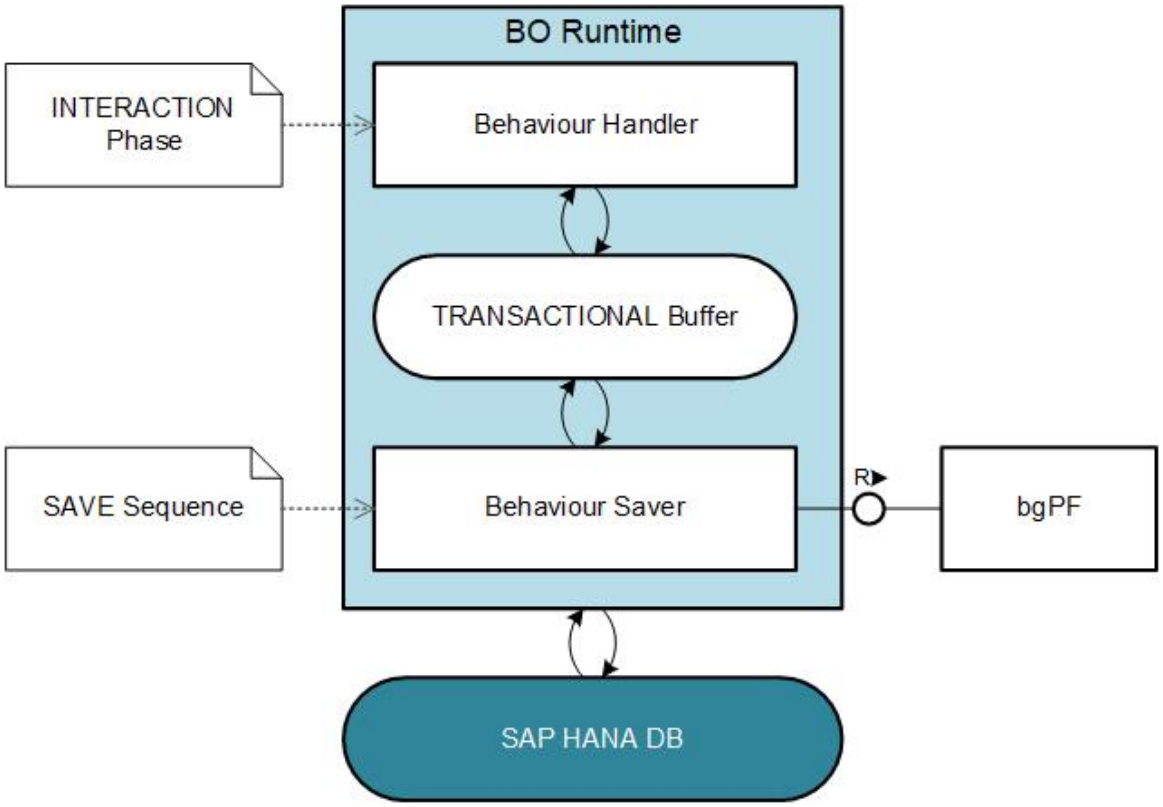
```
cl_bgmc_process_factory=>get_for_queued( iv_bg_processing_context = '<YOUR_bgProcessingContext'
                                         iv_queue                 = 'QUEUE_1' ).
```

3.4.3. How to use bgPF & RAP?

3.4.3.1. Use bgPF inside RAP

The RAP business object runtime mainly consists of two parts:

The first part is the **interaction phase**, in which a consumer calls the business object operations to change data and read instances with or without the transactional changes. The business object runtime keeps the changes in its internal transactional buffer which represents the state of the instance data. This transactional buffer is always required for a business object. After all changes were performed, the data can be persisted. This is realized with the **save sequence.**



The bgPF method save_for_execution( ) must be called in the save sequence. You can call all bgPF methods inside the save sequence. As alternative you can create the operation and set it into the bgPF process in the interaction phase. Method save_for_execution( ) must be called on the same bgPF instance as in the interaction phase.

SAP help: The RAP Transactional Model for the SAP LUW / behaviour handler / behaviour saver

bgPF call in RAP

```
" E.g. modify handler
CLASS <CL_MY_BEHAVIOUR_HANDLER> DEFINITION INHERITING FROM cl_abap_behavior_handler.
  METHODS my_modify_method FOR MODIFY
      IMPORTING it_param FOR ACTION <MY_BO>~my_action RESULT result.

" E.g. SAVE sequenz
CLASS <CL_MY_BEHAVIOUR_SAVER> DEFINITION INHERITING FROM cl_abap_behavior_saver.
  PROTECTED SECTION.
    METHODS save REDEFINITION.  "save_for_execution() must be called here
```

3.4.3.2. Use RAP inside bgPF

Your bgPF process is executed in a new ABAP session. The Entity Manipulation Language (EML) is a part of the ABAP language that enables access to RAP business objects. Here you can use the EML inside of bgPF to read or modify RAP Business Objects as usual. If you use the IF_BGMC_OP_SINGLE_TX_CONTR you have to follow the transaction buffer and commit rules that are explained here: IF_BGMC_OP_SINGLE_TX_CONTR

3.4.4. When do I need an own background Processing Context and how can I create such an object?

Important hint: The delivery of the background Processing Context will be done with 2402. At the moment the only option is to use the default destination from bgPF in the Cloud (CE-0326 - Background RFC (bgRFC) - Technical Configuration). The name of the bgRFC destination is BGPF.

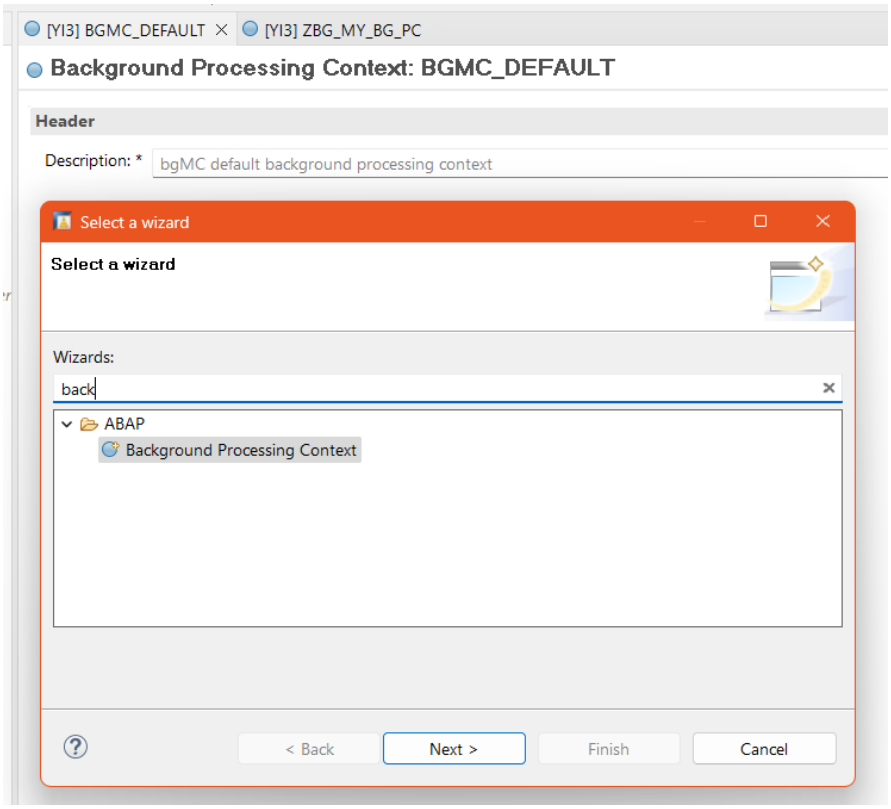
OnPremise customers have to create an bgRFC inbound destination with the name BGPF in their systems. To achieve this they have to configure a BGRFC inbound destination with transaction SBGRFCCONF.

Therefore, "In Order" processing is not possible with 2308 & 2311

If you application needs "In Order" processing you need an own background Processing Context object, see How to use a Queue?.

The idea behind an own object is that in the future (not with 2308) customers can define priorities with this object. For sure you can use the default Processing Context that is delivered with bgPF. All applications that use this default object have the same priority and customers can't be distinguishing between different applications. This is relevant if mass data are processed in the system.

With ADT 3.33.300 you can create an own background Processing Context.



3.4.5. I am wondering why my background operation does not start. What could be the problem?

1. You can use the trace to see if the background operation from you was started, see ABAP Cross Trace
2. If not, ensure that a COMMIT WORK was called. The RAP runtime will do a commit.

3.5. How to write Unit tests?

You can use class CL_BGMC_TEST_ENVIRONMENT to create a test environment. A spy is created by CL_BGMC_TEST_ENVIRONMENT=>CREATE_FOR_SPYING( ).

- An application's unit test can inspect the processes its productive coding generates.
- The tests validate, that the application's operation is instantiated and added to a process correctly.
- The operation themselves is tested independently.

Spy in action

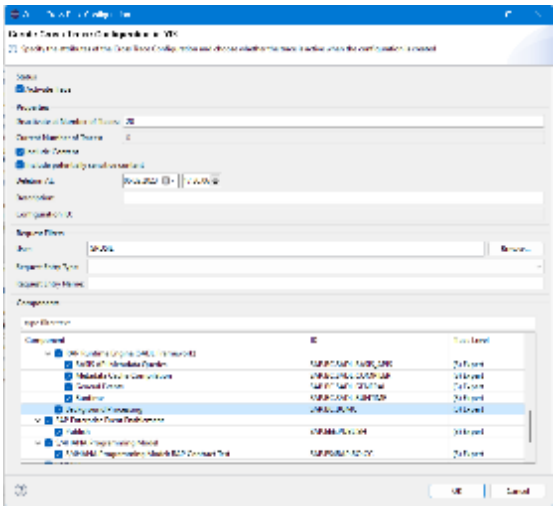
```
data(lo_bg_spy) = CL_BGMC_TEST_ENVIRONMENT=>CREATE_FOR_SPYING( ).

mo_cut->if_bgmc_tea_application~run_single_tx_controlled( ms_operation_input ).

lo_bg_spy->assert_number_of_processes( 1
            )->get_process( 1
            )->assert_is_saved_for_processing(
            )->assert_number_of_operations( 1 ).
```

4. Monitoring

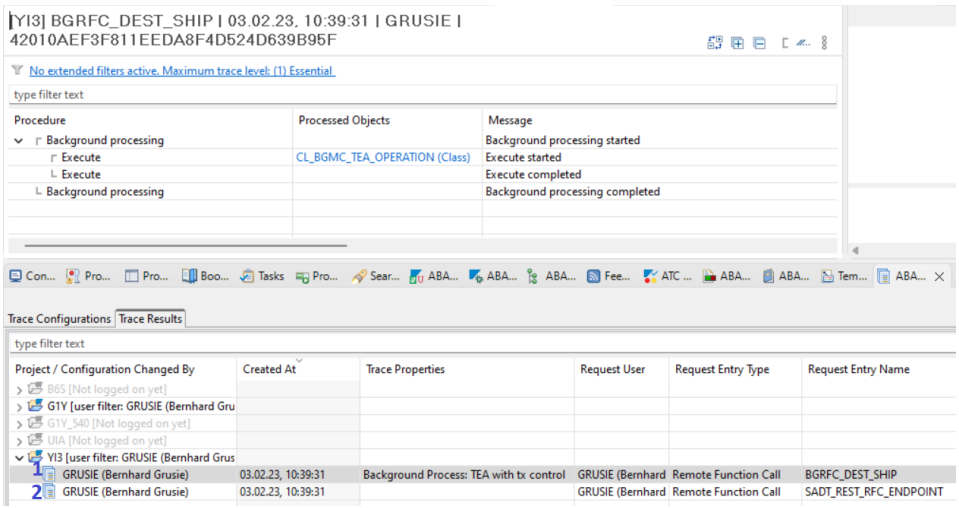
4.1. ADT ABAP Cross trace

background Process Framework in ABAP Cross Trace.

1. Create Configuration:

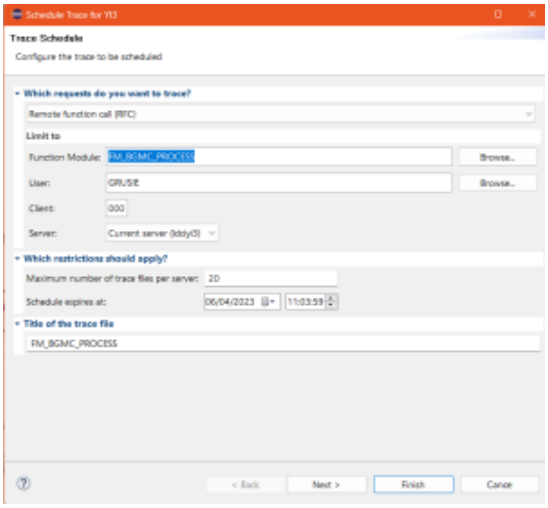


2. Trace Result:
Entry two is from the synchronous call. Entry one is from the new background process

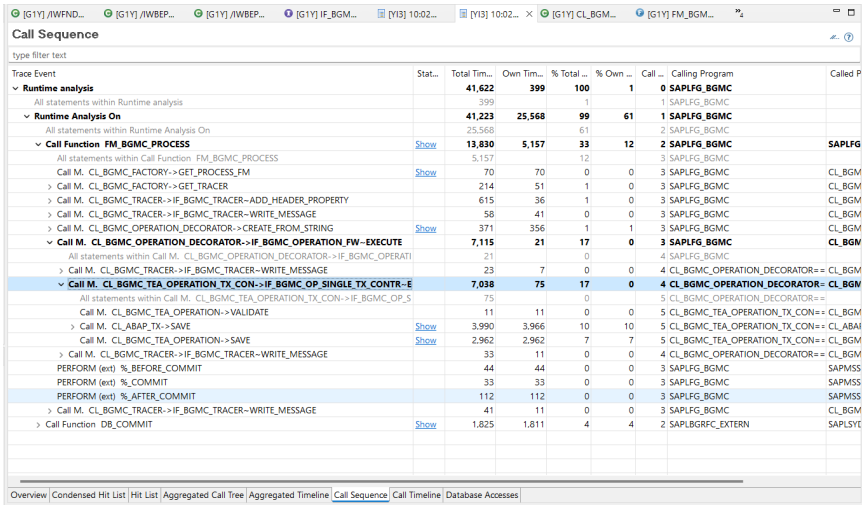


4.2. ADT Performance Trace

If you like to find the bgPF in the ADT Performance trace you can use Function Module FM_BGMC_PROCESS as filter:



In the result list you will have at least to entries for the background process and for the caller of the bgPF. In the call sequence you will find you operation implementation, see the marked entry in the picture below:



4.3. bgRFC monitor

With transaction SBGRFCMON you can start the bgRFC monitor. As alternative you can use the Fiori App: SAP Help

4.4. AIF integration

The AIF integration needs beside the implementation of the bgPF AIf interfaces, also AIF customizing. You have to follow the the AIF how to guide: AIF Interface Creation for Monitoring bgPF

4.5. How to deal with dumps?

If you operation implementation leads to a dump the monitoring tools like ADT Cross Trace or AIF (**FBSAIF-2093** - Getting issue details... STATUS) are not aware of this dump. With 2308 there is no solution from bgPF side for this problem.

When you have an own monitoring tool, you can use the EPP Passport to find in the dumps your operation.