RAP : BGPF

Background processing framework (BGPF)

RAP: Background processing framework (BGPF)

To understand about bgpf, lets take a problem statement:

- **you are an end user and you are supposed to create a sales order with sales order items. Now, let's say you create a sales order with 100 sales order items. It takes a lot of time to create a sales order with this many line items. You will be stuck on a loading page until the operation is completed.**

Solution:

- **Now, with the help of BGPF, you can run sessions asynchronously. This means you can create a sales order in the background and still proceed with creating another sales order while the previous one is processing. You no longer have to wait on a loading screen. This is especially helpful in scenarios involving large-scale processing.**

Note: BGPF doesn't make your processing faster, rather run it on background. So that you can continue with other session.

Types of bgpf:

- **Controlled bgpf: we know that rap works on controlled luw(*). To maintain transactional consistency we use controlled bgpf. So that we can write logic based on interactive and save sequence logic.**
- **Uncontrolled bgpf: we directly write logic in a single transaction.**

*** controlled luw: is a key reason why classical ABAP is shifting toward RAP. In classical ABAP, developers are responsible for ensuring data consistency manually. In RAP, changes are made to the transactional buffer during the interactive phase, and only consistent data is saved during the save sequence.**

Working of Bgpf:

There are mainly 2 classes required for bgpf:operation class and starter class:

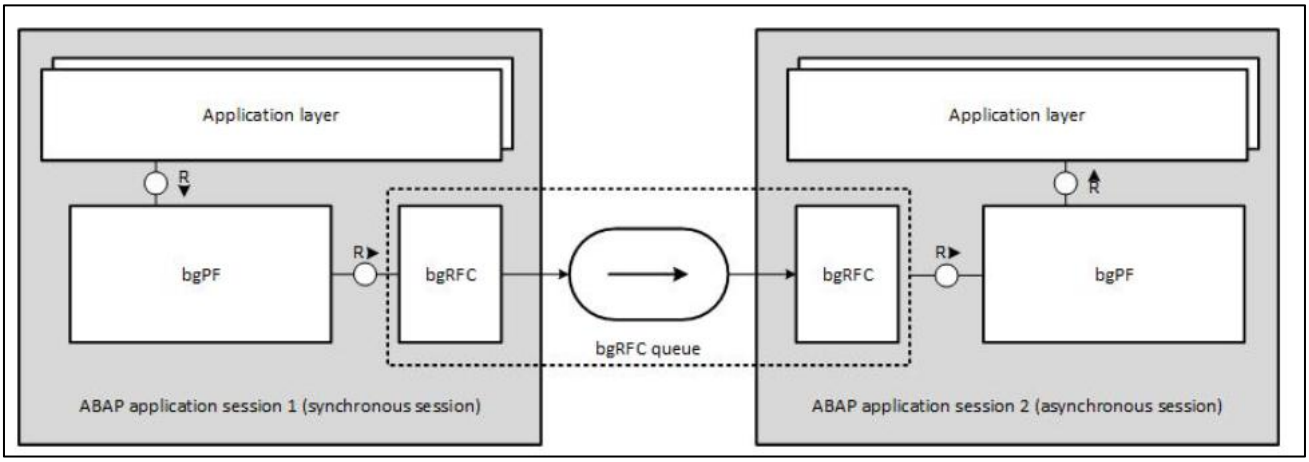
```
METHOD save_modified.
  DATA lo_operation TYPE REF TO if_bgmc_op_single.
  DATA: lt_changed_on   TYPE TABLE OF zi_lfa1_2.
  DATA:lo_process TYPE REF TO if_bgmc_process_single_op.
  LOOP AT create-zi_lfa1_2 INTO DATA(wa_lfa1).
    lt_changed_on = VALUE #( FOR ls_changed_on IN create-zi_lfa1_2
    ( lifnr = ls_changed_on-Lifnr changed_on = ls_changed_on-changed_on ) ).
    lo_operation = NEW zbgpf_class( lt_changed_on[ 1 ] ).

    TRY.
        lo_process = cl_bgmc_process_factory=>get_default(
                                              )->create( ).
        lo_process->set_name( 'My process'
            )->set_operation( lo_operation ).
        lo_process->save_for_execution( ).
      CATCH cx_bgmc INTO DATA(lx_bgmc).
    ENDTRY.
  ENDLOOP.
ENDMETHOD.
```

Starter class

```
CLASS zbgpf_class DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
    INTERFACES if_bgmc_op_single .
    METHODS constructor
      IMPORTING
        iv_data TYPE zi_lfa1_2.
  PRIVATE SECTION.
    DATA mv_data TYPE zi_lfa1_2.
    METHODS save.
ENDCLASS.


CLASS zbgpf_class IMPLEMENTATION.

  METHOD constructor.
    mv_data = iv_data.
  ENDMETHOD.

  METHOD if_bgmc_op_single~execute.
    cl_abap_tx=>save( ).
    save( ).
  ENDMETHOD.

  METHOD save.
    MODIFY zlog_book FROM @(
    VALUE #( lifnr = mv_data-Lifnr
             changed_on = mv_data-Changed_on ) ).
  ENDMETHOD.

ENDCLASS.
```

Opertion class

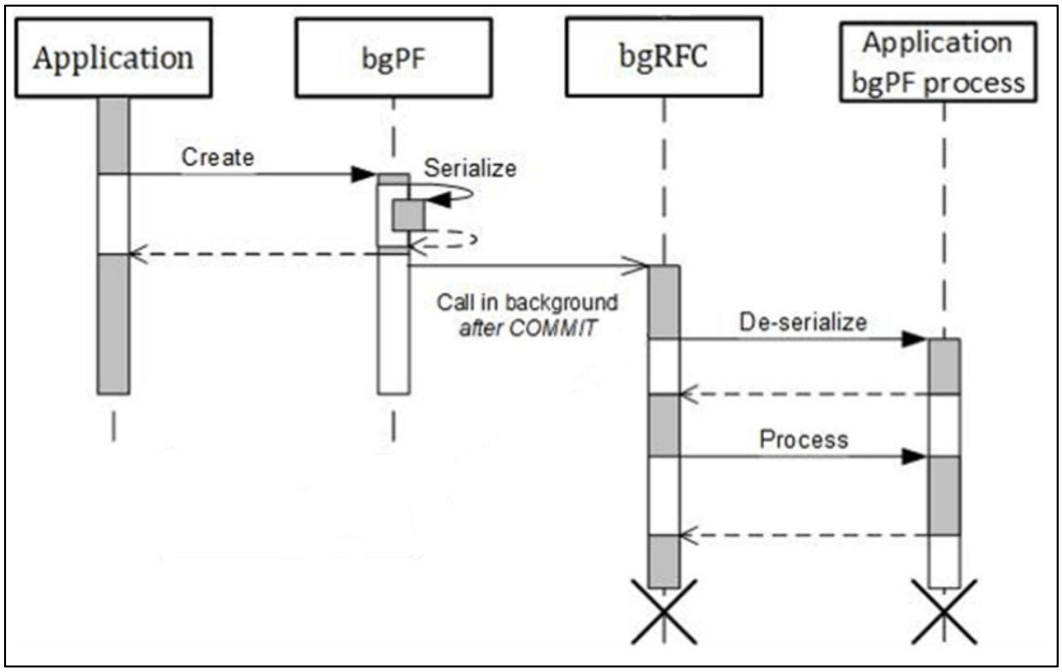
- **Starter class(in our case local saver class):** **does the operation and handover to the bgpf.**
- **Operation class:** **holds the logic to be executed asynchronouly in background.**



- **Bgpf is wrapper of Bgrfc**

Workflow of bgpf:

1. We first create an instance of bgpf inside starter class(lsc), using: cl_bgmc_process_factory=>get_default( ), for transactional bgpf cl_bgmc_process_factory=>get_for_queue, For queued bgpf.
2. Save the process for background processing in a separate abap session. using lo_process->save_for_execution( ).
3. After the first session is executed and commit work is triggered.
4. New session will start for bgpf and bgrfc is called . Inturn bgrfc calls if_bgmc_op_single~execute of operation class. To switch save sequence, we can use cl_abap_tx=>save( ).



Requirement: To Maintain a log book of when the buisness object is created. Even though this is not a large processing . I am using this as an example to show working of bgpf. We can have normal creation of bo in 1 session and log it on a table seperately inside another asynchronous session.

Steps:

1. **Our local saver class acts as the starter class. We use cl_bgmc_process_factory=>get_default( ) to create an instance of BGPF. Then, pass the data from the transactional buffer to the constructor of the operation class. Inside a try-catch block, create a BGPF process to run asynchronously. Declare what process to be carried asynchronously, we pass instance of operation class and save the process for background execution**

```
METHOD save_modified.
  DATA lo_operation TYPE REF TO if_bgmc_op_single.
  DATA: lt_changed_on   TYPE TABLE OF zi_lfa1_2.
  DATA:lo_process TYPE REF TO if_bgmc_process_single_op.
  LOOP AT create-zi_lfa1_2 INTO DATA(wa_lfa1).
    lt_changed_on = VALUE #( FOR ls_changed_on IN create-zi_lfa1_2
    ( lifnr = ls_changed_on-Lifnr changed_on = ls_changed_on-changed_on ) ).
    lo_operation = NEW zbgpf_class( lt_changed_on[ 1 ] ).

    TRY.
        lo_process = cl_bgmc_process_factory=>get_default(
                                              )->create( ).
        lo_process->set_name( 'My process'
            )->set_operation( lo_operation ).
        lo_process->save_for_execution( ).
      CATCH cx_bgmc INTO DATA(lx_bgmc).
    ENDTRY.
  ENDLOOP.
ENDMETHOD.
```

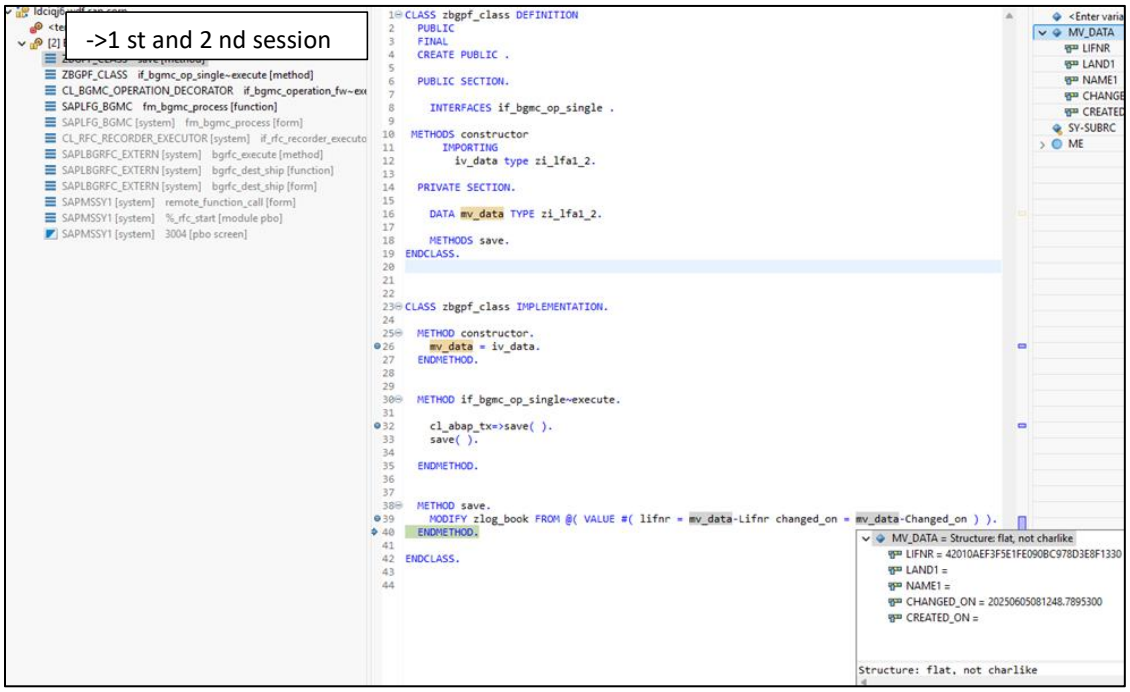
```
METHOD save_modified.
  DATA lo_operation TYPE REF TO if_bgmc_op_single.
  DATA: lt_changed_on   TYPE TABLE OF zi_lfa1_2.
  DATA:lo_process TYPE REF TO if_bgmc_process_single_op.
  LOOP AT create-zi_lfa1_2 INTO DATA(wa_lfa1).
    lt_changed_on = VALUE #( FOR ls_changed_on IN create-zi_lfa1_2
    ( lifnr = ls_changed_on-Lifnr changed_on = ls_changed_on-changed_on ) ).
    lo_operation = NEW zbgpf_class( lt_changed_on[ 1 ] ).

    TRY.
        lo_process = cl_bgmc_process

        lo_process->set_name( 'My pr
            )->set_operation( lo_ope
        lo_process->save_for_executi
      CATCH cx_bgmc INTO DATA(lx_bgm
    ENDTRY.
  ENDLOOP.
ENDMETHOD.
```

LT_CHANGED_ON = [1x5(118)]Standard Table
[1] = Structure: flat, not charlike
LIFNR = 42010AEF3F5E1FE090BC978D3E8F1330
LAND1 =
NAME1 =
CHANGED_ON = 20250605081248.7895300
CREATED_ON =

[1x5(118)]Standard Table

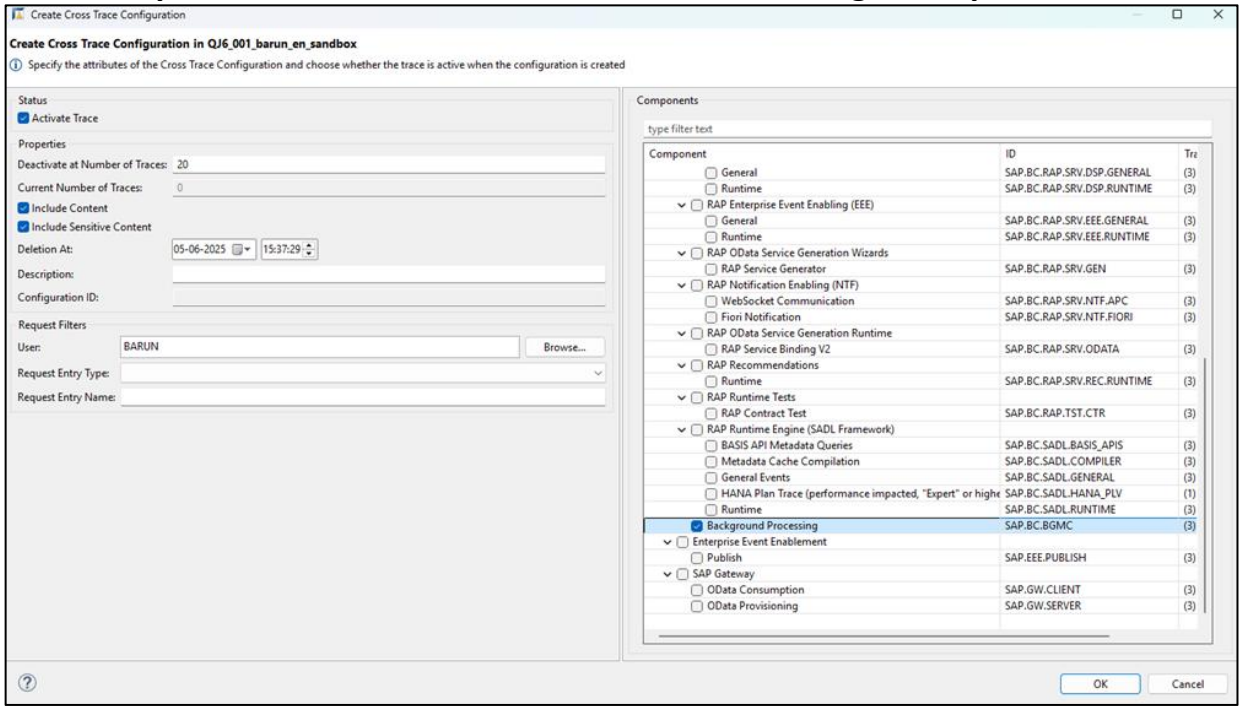
2. **After, the save sequence is executer Rap luw commit will happen. Then your data will be persisted to persistence table. After which 2nd session will run in background.**

3. **After commit work. if_bgmc_op_single~execute will be triggered. Inside operation class, I have used constructor to get the data from transactional buffer and used cl_abap_tx=>save( ) to switch to save sequence and append the log into custom table.**



How to monitor bgpf:

1. **Go to abap cross trace, and create cross trace for background process:**



2. After completion of bgpf, you can deactivate andcheck in results tab.You would see both the sessions

Procedure	Processed Objects	Message	Record Properties
Background processing		Background processing started	
Execute	ZBGPF_CLASS (Class)	Execute started	Operation:ZBGPF_CLASS
Execute		Execute completed	
Background processing		Background processing completed	