

Mass Data Runtime - Developers Guide

Manual

7.2 — Last update: 2015/03/04

Basis Technologies

Table of Contents

Audience	2
Introduction	3
When to use MDR	4
Basic Concepts	5
Instance	6
Intervals	7
Interval Processing	9
Collation.....	10
Transformation.....	11
Main Program	12
General.....	13
Declaration.....	14
Selection Screen Definition	15
START-OF-SELECTION Event	16
Interval Processing Subroutine.....	17
Collating Interval Results	19
AT SELECTION-SCREEN OUTPUT	21
Transformation Program.....	22
Regular Transformation Program	23
Selection Screens and Transformation Programs.....	25
Setting Up MDR	26
License Keys	27
Program Definition	28
Default Technical Settings.....	30
Interval Generation	32
Running MDR Programs	34
Technical Settings.....	35
Developer Notes	39
Administering MDR Programs	41
Expert Mode	42
Results.....	43
Intervals	44
Variants	45
App Servers	46
Increase Jobs	47
Decrease Jobs	48

Stop	49
Resume	50
Delete	51
Force Error.....	52
Scheduling MDR Programs.....	53
Exception Handling and Application Logs.....	54
Email Notification	56
Debugging and Troubleshooting	58
Debugging MDR Programs.....	59
Troubleshooting	60
Advanced Concepts	61
Dynamic Interval Generation	62
Authority Checks	65
Implementation	66
Technical Settings	68
Expert Mode	72
Individual Actions.....	76

Audience

This guide has been developed for the following audience:

- ABAP™ Developers intending to use the MDR framework
- Performance Specialists or Advisers
- SAP® Architects

Introduction

Basis Technologies' Mass Data Runtime (MDR) provides a framework and run-time environment for custom developed SAP reports that must work with large volumes of data. It provides for parallel execution of custom reports such that these programs can run within acceptable time constraints to deliver timelier business information.

MDR programs can be easily developed using the standard SAP ABAP Workbench. The MDR methodology guides the developer, by making them adhere to a structured program design and standardized technical program structure that separates data processing logic from presentation logic. For each MDR program, there are three main components that need to be set up. This refers to the MDR program configuration (or MDR definition) and also two programs in the ABAP repository for each MDR program, these three components are:

1. Main Program – This ABAP program retrieves data from the database, processes this information and then stores the data.
2. Transformation Program – This ABAP program retrieves the data that was processed in the main program and presents this to the user.
3. MDR Definition – This defines the name of the technical components of each MDR program, as well as a number of other settings relevant to the MDR program.

This guide will take you firstly through the Basic concepts of the MDR framework and then get you started on how to develop your own MDR programs. This will allow you to write ABAP reports that process large volumes of data in your SAP system.

When to use MDR

MDR is essential when the data volumes are so large that the processing time to run your reports is unacceptable. If you are able to write a report that runs within acceptable time-constraints then the use of MDR may not be required. However, even if your report runs within 2 hours (and this is considered acceptable) you are still able to use MDR to bring this run-time down even further.

It is our consideration that almost any report that executes in the background can gain benefits by being implemented in MDR. This is because the MDR format not only promotes performance improvements, but also cost reductions in maintenance by having a generic format. It also provides the benefit of separating the processing and presentation logic. Separate presentation logic allows the data to be viewed interactively and in a user-friendly manner – far easier than lengthy static list output.

Basic Concepts

Before starting development on an MDR program, a few basic concepts must be explained.

- [Instance](#)
- [Intervals](#)
- [Interval Processing](#)
- [Collation](#)
- [Transformation](#)

Instance

Each run of an MDR program is called an 'Instance' this can be given a separate label and using the framework a user can view a number of previous instances and pick which run they wish to view the details of. The framework allows the saving of data against the instance.

Intervals

MDR works on the principal that data processing can be divided into independent “pieces” of work. These pieces of work are referred to as Intervals and usually represent a range of master or transactional data that needs to be worked upon. This for example may be Intervals of Business Partner Numbers, Sales Order Numbers, or Account Numbers. MDR ensures that these Intervals of data are processed in parallel instead of the traditional sequential approach – this in essence is why MDR can provide such impressive performance gains.

An interval can be thought of as a range of values that represent X number of master or transactional data objects. It is based on some data domain having a data type and a length. An example interval might be “All Sales Order’s from 1000 through to 2000”. It can then be implied that an interval has a Low value (e.g. 1000) and a High value (e.g. 2000).

If an SAP system has 100,000 Sales Orders, and an ABAP report is required to process all of them, then this range of Sales Orders can be broken down into, for example, 100 intervals, each representing 1000 Sales Orders. The list of 100 intervals might look something like:

Interval	Low	High
1	1	1000
2	1001	2000
3	2001	3000
...
100	99001	100000

The concept of an Interval Object is used to create these Intervals for use by an MDR program this is where an object such as a Sales Orders are broken down before the MDR program is run. As part of the definition of an MDR program, an Interval Object is selected. This refers directly to the type of Intervals the program uses, for example, some Interval Objects that are provided with MDR are:

- Sales Order
- Business Partner
- Contract Account

The generation of Interval Objects is detailed in the Setting Up MDR [Interval Generation](#) section.

Intervals can also be built at runtime, this is programmed into the MDR program, this is useful where you have complex intervals or need to define the number of intervals down to a smaller size, a subset of

materials for example, this concept is detailed in the Advanced Concepts [Dynamic Interval Generation](#) section.

Interval Processing

For an MDR program to complete it needs to process all available intervals. Interval Processing refers to the custom source code that processes the data relating to each interval – for example processes all Sales Orders with the keys 0000000001 – 0000001000. Note that the processing of each interval should be performed independently of the processing of any other interval. The main focus of the processing component of the MDR program is to process each interval independently; this means dealing with only with the master and transactional data within the range of the interval.

When an MDR program is executed, it is run with X jobs (where X is a number from 1 upwards). MDR achieves its performance gains by executing each job in parallel, hence the reason why the processing of one interval should be independent of any others. As each interval is processed, the results of that interval are stored. MDR provides a number of easy to use statements as a generic storage mechanism for each interval's results. These will be explained in more detail in this document.

Collation

In most cases, once each interval has been successfully processed there exist data results for that interval. The data from all intervals will need to be merged back into a single result. This is achieved by the use of a Collation process. The Collation refers to another custom process that is responsible for taking the data of each interval and combining them all into a single result (as if the program had been run sequentially without the concept of intervals). This single result is saved against the instance.

It is possible that a program does not require any collation, for example, a program that creates outbound IDocs.

Transformation

Once the collation has been performed, causing the results to be brought back together, the remaining piece of the custom program is the presentation logic. As mentioned previously, this is performed in a separate program called the Transformation program. The reasoning behind the separation of the processing and transformation logic is three-fold:

- Processing logic and presentation logic are completely separate, leading to increased program clarity and reduced maintenance costs
- Further restriction of the collated results can be achieved
- Results of different instances can be viewed interactively and repeatedly by different users

Here is the link to further details on [Transformation Programs](#)

Main Program

This section takes you through the steps to construct a main MDR program.

- [General](#)
- [Declaration](#)
- [Selection Screen Definition](#)
- [START-OF-SELECTION Event](#)
- [Interval Processing Subroutine](#)
- [Collating Interval Results](#)
- [AT SELECTION-SCREEN OUTPUT](#)

General

The Main Program is a simple “Type 1” (Executable) ABAP program/report and can be created or changed via the transaction /BTR/MDR, or the standard SAP transaction SE38. For ease we recommend the transaction /BTR/MDR. Within the Main program, the basic structure of a custom ABAP report is:

- Declaration – Declaration of working variables, include files, type declarations and so forth
- Selection Screen – Parameter screen definition of select-options and parameters
- Start-Of-Selection Event – Program Initialization
- Interval Processing Subroutine – Initial Data Restriction logic and Processing logic
- Collating Interval Subroutine – Collation logic to combine interval results (This is optional)

Note that there is no mention of result presentation in the Main Program; this is implemented in the Transformation program. This means that you should not do the following in the Main program:

- Use the WRITE statement for display to the SAP spool or screen (You can of course use the WRITE statement to format data into variables)
- Use the AT USER-COMMAND or AT LINE-SELECTION processing blocks
- Declare headings or text elements that are to be used for presentation
- Perform any sort of presentation logic

Declaration

When you create the Main program, you must include /BTR/MDR_INCLUDE. This can be done at the start of the Main program as in the below example:

```
*-----*
* Sample MDR Program
*-----*
REPORT Z_MDR_SAMPLE_PROGRAM.

INCLUDE:
  /btr/mdr_include.
```

This is also where you should include other declarations such as other includes, type declarations, global data declarations and table statements.

Selection Screen Definition

Almost all custom ABAP reports enable the user (or background variant) to supply select-options to restrict the data that is retrieved and processed. Standard ABAP reports allow the developer to use the keywords:

- SELECTION-SCREEN
- SELECT-OPTIONS
- PARAMETERS

In your MDR program, you must define the select-options and parameters of a program within the following MDR statements:

mdr-begin-selection-screen and mdr-end-selection-screen

Everything else regarding syntax remains exactly the same. This allows the [technical settings](#) button to be seen on screen.

An example declaration of parameters and select-options is as follows:

```
*-----*
* PARAMETERS and SELECT-OPTIONS
*-----*

mdr-begin-selection-screen.

SELECTION-SCREEN BEGIN OF BLOCK general WITH FRAME TITLE text-t01.
PARAMETERS:
  p_bukrs  TYPE vbak-bukrs OBLIGATORY,    p_keydat TYPE d OBLIGATORY.

SELECT-OPTIONS:
  s_belnr  FOR vbak-belnr,    s_crdat  FOR vbak-erdat.
SELECTION-SCREEN END OF BLOCK general.

mdr-end-selection-screen.
```

START-OF-SELECTION Event

The START-OF-SELECTION event must be defined in the MDR Main report. However, unlike standard ABAP reports this event does nothing except execute an MDR statement. This statement must immediately follow the START-OF-SELECTION keyword and nothing else should follow it. In effect, this single MDR statement begins processing the entire report. The following is a sample of how the START-OF-SELECTION event should appear:

```
*-----*  
* START-OF-SELECTION event  
*-----*  
START-OF-SELECTION.  
    mdr_program_initialize.
```

This statement will begin execution of the MDR program. Intervals will be generated (or retrieved) and one or more jobs started. For each interval, the interval processing subroutine will be called (defined next).

Interval Processing Subroutine

The special MDR subroutine `mdr_interval_processing` is called once for each interval to be processed. It is passed a data dictionary structure of type `/BTR/ST_INTERVAL_VALUES` that has a number of fields defined. The most important of these fields is “LOW” and “HIGH”. At the start of the interval processing, you should typecast these two values to your own local variables.

The next step is to retrieve the data you require in your application specific program. However, you need to ensure that your program only retrieves the data relevant for that particular interval. This means your `SELECT` statements (if appropriate) must use the LOW and the HIGH values in the restriction of data retrieved.

This differs only slightly from a standard ABAP report. Put simply, the selection of the master/transactional data must be extended to include a further `WHERE` clause.

Once all data in the interval has been retrieved and processed appropriately, you need to save off the results of the processing. This is done using the MDR macro `mdr_interval_result_put`. It expects two parameters (1) a label identifying the data and (2) the data to be saved. The data to be saved can take any form whether it is a locally declared type or a data type defined in the data dictionary. The data can also be an internal table or even a complex type containing both structures and internal tables.

An example Interval Processing subroutine is shown below:

```
*-----*
* FORM mdr_interval_processing
*-----*
* This form is called by the MDR architecture to process each      *
* interval range. Results for each interval should be calculated   *
* within this subroutine and be saved for later collation          *
*-----*
FORM mdr_interval_processing
  USING x_interval TYPE /btr/st_interval_values.

DATA:
  lt_vbak      LIKE vbak OCCURS 0 WITH HEADER LINE,
  lv_belnr_low TYPE vbak-belnr,
  lv_belnr_high TYPE vbak-belnr.

* Type cast the interval low and high values
  lv_belnr_low = x_interval-low.   lv_belnr_high = x_interval-high.

* Retrieve data
  SELECT *
    FROM vbak
    INTO TABLE lt_vbak
    WHERE belnr BETWEEN lv_belnr_low AND lv_belnr_high
```

```
    AND buhrs = p_buhrs.  
  
* Process data  
  PERFORM process_data TABLES lt_vbak.  
  
* Save results of interval  
  mdr_interval_result_put 'RESULTS' gt_results.  
  
ENDFORM.
```

Note that the interval low and high range values have been typecast at the start of the subroutine into the local variables. The data is then retrieved using the LOW and HIGH values to restrict the list of Sales Orders returned. Furthermore, the parameters declared in the Selection screen definition are also used to restrict the data that is retrieved.

Once the data has been retrieved, a subroutine is called to process the data. This subroutine uses the Sales Order data and accumulates a global internal table called GT_RESULTS. It is not important what logic this subroutine performs, or the structure of this data, but rather that once this internal table is built, it is then saved using the MDR statement `mdr_interval_result_put`.

Note that you can save multiple results for a particular interval. For example:

```
* Save results of interval  
  mdr_interval_result_put 'RESULTS' gt_results.  
  mdr_interval_result_put 'OTHERRESULTS' gt_other_results.
```

The important point to note is that the label must be unique. Now that the results have been determined and saved for the interval, the results of all intervals can be combined into a single “final” result. This is the function of the Collation subroutine, which is described in the following section.

Collating Interval Results

After all the intervals of the program run have been processed, MDR calls a special subroutine to collate the interval results back together into a single result. Once collated, sets of results exist for the program run. The logic required to collate the interval results is often quite simple but will vary depending upon the nature of the report.

In order to develop your own collation routine, it is a simple matter of adding a new subroutine to the Main program. This subroutine must be called `mdr_interval_collation`. It has a single parameter passed to it, which is of type `/BTR/TT_MDR_INTERVALS`. This parameter contains the complete list of intervals and the associated result(s) that have been calculated for each interval in the interval processing subroutine. It is the task of the Collation routine to loop through each interval and retrieve the result for that interval. It should then combine the data results of each interval together to form one large combined result. An example collation routine is shown below:

```
*-----*
* FORM mdr_interval_collation
*-----*
* This form is called by MDR after all intervals have been processed. *
* It can be used to collate the results of the intervals together.    *
*-----*
FORM mdr_interval_collation
  USING xt_intervals TYPE /btr/tt_mdr_intervals.
  DATA:
    lv_interval    LIKE LINE OF xt_intervals,
    lt_summary     LIKE gt_summary_list[],
    lv_parameters  TYPE ty_parameters.

  REFRESH gt_summary_list.

  LOOP AT xt_intervals INTO lv_interval.
*   Get the summary results for this interval
    mdr_interval_result_get lv_interval co_label_summary lt_summary[].

*   Accumulate in the collated summary
    LOOP AT lt_summary INTO gt_summary_list.
      COLLECT gt_summary_list.
    ENDLOOP.
  ENDLOOP.

* Save the summary list for access by the transformation program.
  mdr_instance_result_put co_label_summary gt_summary_list[].
ENDFORM.
```

As can be seen in the example code the steps of the collation routine are to:

1. Loop through each interval passed into the subroutine
2. For each interval, retrieve the results that have been calculated by the interval processing routine by using the MDR statement `mdr_interval_result_get`
3. For each result of the interval, combine the data into a “complete” result
4. Store the “complete” result back into MDR using the statement `mdr_instance_result_put`

Let's first look at the MDR statement `mdr_interval_result_get`.

```
mdr_interval_result_get lv_interval co_label_summary lt_summary[].
```

The first parameter of this statement “`lv_interval`” is the interval detail that we are retrieving the data from; this is simply the appropriate row of the parameter `xt_intervals`.

The second parameter of this statement “`co_label_summary`” is a character string that represents the label of the data. This must be the label that was used to originally store the result during the interval processing.

The final parameter is the result variable itself “`lt_summary[]`”. This must have the same structure as was originally stored when the result was “put” during the interval processing routine using the MDR statement `mdr_interval_result_put`. It is crucial that this data type is the same.

Once the Collation subroutine executes, the program will now have a single “complete” result (or results) for the entire interval set. This complete result is a combination of the results of each interval of work as though one job had executed the entire report.

With the processing component of your MDR program finished and the end result collated – the final step is to present this data to the user.

AT SELECTION-SCREEN OUTPUT

If you intend to use the AT SELECTION-SCREEN OUTPUT event in your MDR program, you will find that MDR does not allow you to use this, this is due to some constraints in defining multiple AT SELECTION-SCREEN OUTPUT events in a program (MDR also uses this for the technical settings tab of your program). You can still implement this code by creating the subroutine `AT_SELECTION_SCREEN_OUTPUT` in your Main MDR Program as below. There are no parameters required to this form, and you can access the `SCREEN` structure as normal.

```
*-----* FORM at_selection_screen_output *-----*
* This form allows custom AT SELECTION-SCREEN OUTPUT events to *
* be used                                                         *
*-----*
FORM mdr_at_selection_screen_output.

* Here you can access the screen variable as if you were using
* the standard AT SELECTION-SCREEN OUTPUT event. All other
* report events can be defined as normal.
LOOP AT SCREEN.
    CASE screen-name.
        WHEN 'S_CARRID-LOW'.
        WHEN 'S_CARRID-HIGH'.
    ENDCASE.

    MODIFY SCREEN.
ENDLOOP.

ENDFORM.                                "at_selection_screen_output
```

Transformation Program

- [Regular Transformation Program](#)
- [Selection Screens and Transformation Programs](#)

Regular Transformation Program

As mentioned earlier, MDR requires that the processing and presentation components of a report/program are separate. This section describes how the Transformation (Presentation) program is to be structured. The Transformation program is its own stand-alone program with a different name to that of the Main program. The common component of the Transformation and Main program should be the data type(s) of the results. If you have defined your own data type to store the result sets, then either make sure they are defined in the Data Dictionary, or they have been defined in a common include file for use by both the Main and Transformation programs.

The Transformation program is designed to run every time that the user is to be presented with the end result data. This means it needs to first retrieve the results and then present them to the user. One of the major benefits of having the Transformation program separate to the Main program is that it can have its own select-options/parameters. This can be used to further restrict the “collated” data when it is presented. An example transformation program is shown below:

```
*-----*
* Sample Transformation Program
*-----*
REPORT z_sample_mdr_transform_program.

INCLUDE:
  /btr/mdr_include.

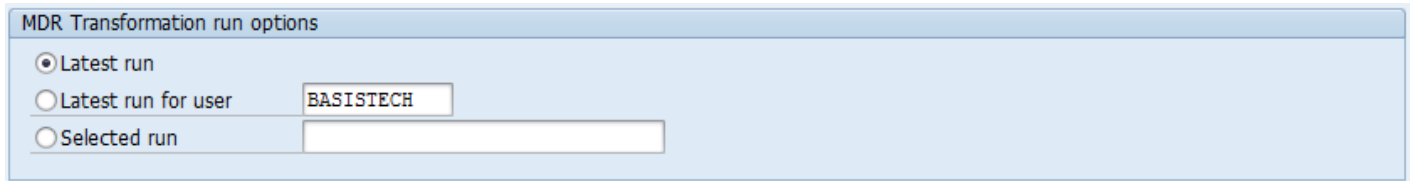
* Transformation run options
  mdr-begin-select_screen_trans.

START-OF-SELECTION.

* Get the summary results
  mdr_instance_result_get co_label_summary gt_summary_list[].

* Display the details to the user
  PERFORM display_result USING gt_summary_list.
```

The MDR statement `mdr-begin-select_screen_trans` must be used before the `START-OF-SELECTION` event. This allows the transformation program to be run separately from the run history and doing so will produce the options below the main part of the selection screen allowing the user to check their latest runs or select runs.



The image shows a dialog box titled "MDR Transformation run options". It contains three radio button options: "Latest run" (which is selected), "Latest run for user", and "Selected run". To the right of the "Latest run for user" option is a text input field containing the text "BASISTECH". To the right of the "Selected run" option is an empty text input field.

The next statement in the transformation program is `mdr_instance_result_get`. This statement will retrieve the “complete” result that was calculated in the Collation subroutine. The label that is passed is very important in that it identifies which result you are trying to retrieve since there may be more than one. The last parameter to the statement is the actual variable that you have defined and must be the exact type as what you have defined in the Collation subroutine when you saved the “complete” result.

Now that the complete result (or results) has been retrieved, you can display them to the user. It is up to you how you do this. In the example on the previous page, a subroutine has been defined that will perform whatever steps are required to display the data. This might involve using `WRITE` statements and outputting to the spool. In many cases, a better approach would be to use the ABAP List viewer and call the `REUSE*` function modules. There are no restrictions to the presentation possibilities.

An important feature of the Transformation program is that it can be interactive. Hence the `AT USERCOMMAND` and `AT LINE-SELECTION` events can be implemented, unlike a traditional batch report that would have the results stored in the spool. The Transformation program will be executed every time a user decides to view the results of an MDR program run.

Selection Screens and Transformation Programs

Transformation programs can also be enhanced with selection screens to enable the data retrieved from the MDR framework to be filtered as per the example below filtering by company code. Or alternatively selection screen options could impact the output or offer file download options in the usual way you might control any ABAP program.

```
*-----*
* Sample Transformation Program
*-----*
REPORT z_sample_mdr_transform_program.

INCLUDE:
  /btr/mdr_include.

SELECTION-SCREEN BEGIN OF BLOCK seloptions WITH FRME TITLE texts01.

SELECT-OPTIONS s_bukrs FOR vbak-bukrs.

SELECTION-SCREEN END OF BLOCK seloptions.

* Transformation run options
  mdr-begin-select_screen_trans.

START-OF-SELECTION.

* Get the summary results
  mdr_instance_result_get co_label_summary gt_summary_list[].

* Filter the summary results
  delete gt_summary_list where bukrs not in s_bukrs.

* Display the details to the user
  PERFORM display_result USING gt_summary_list.
```

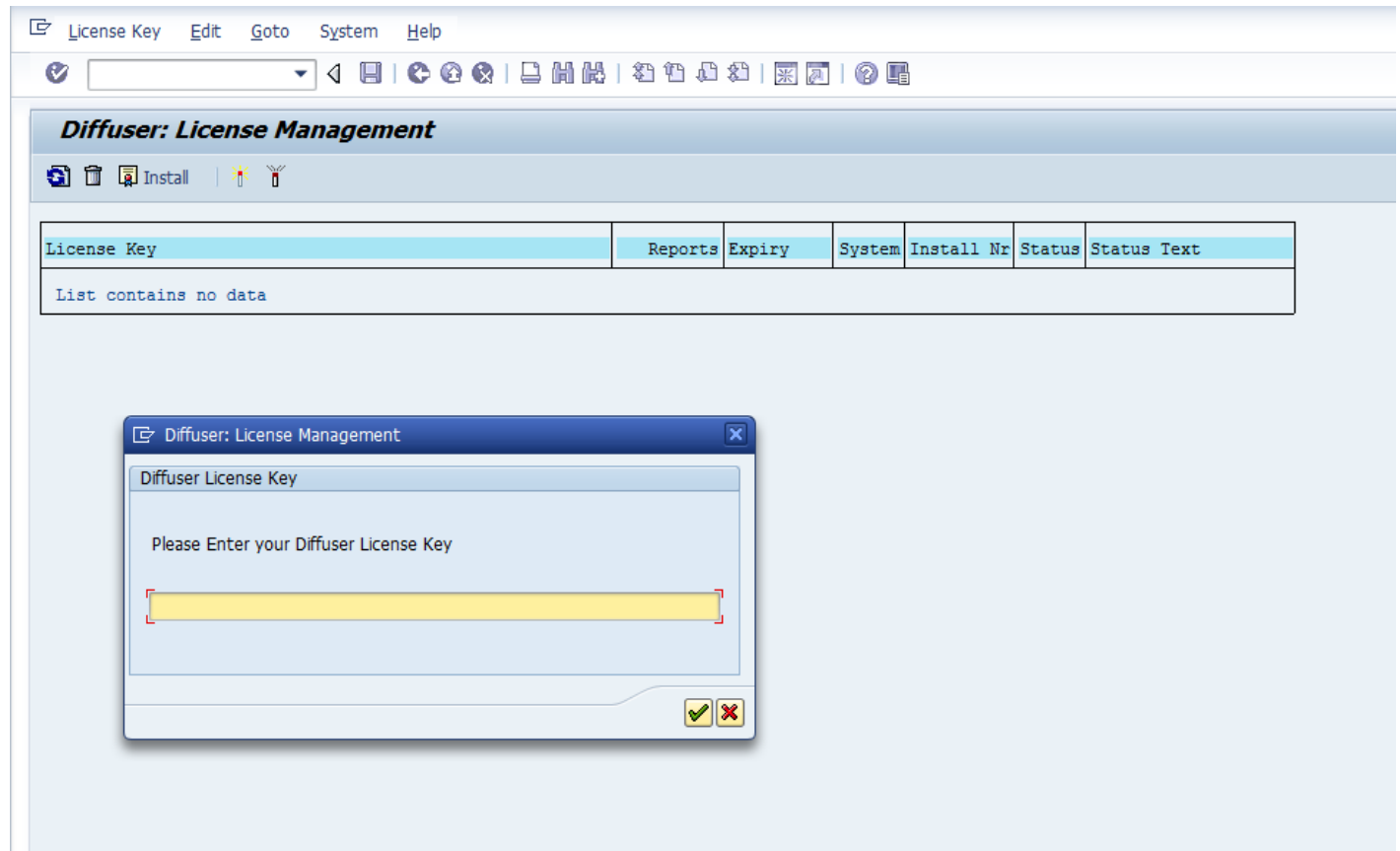
Setting Up MDR

Once the MDR program is constructed you define the MDR program, setup the technical settings and generate intervals where required. The Definition of an MDR program is set up via the transaction /BTR/MDR.

- [Program Definition](#)
- [Default Technical Settings](#)
- [Interval Generation](#)

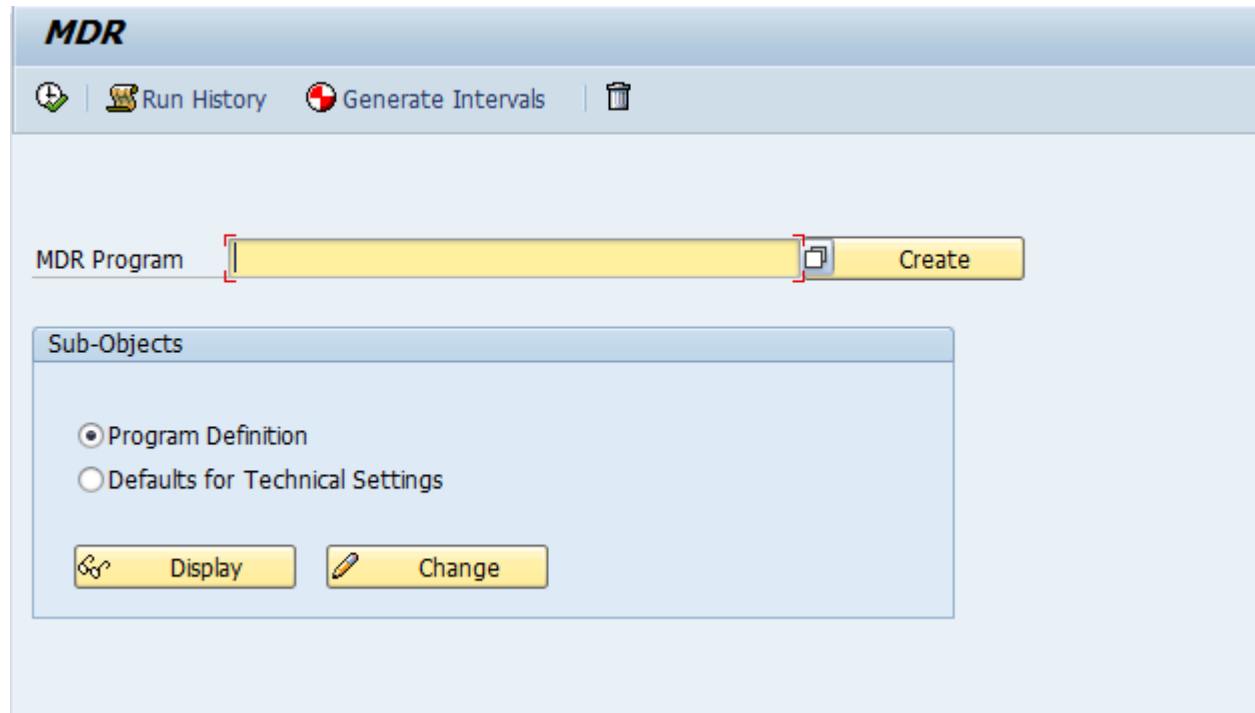
License Keys

Use the transaction /N/BTR/MDRLICENSE then select the option “Install” and a dialog appears into which you can enter the MDR license key.



Program Definition

The definition of an MDR program is set up via the transaction /BTR/MDR.



The screenshot shows the 'MDR' transaction interface. At the top, there is a header bar with the title 'MDR' and a toolbar containing icons for 'Run History' (a clock), 'Generate Intervals' (a red circle with a white cross), and a trash icon. Below the header, there is a main area with a label 'MDR Program' followed by a yellow input field and a 'Create' button. Below this, there is a 'Sub-Objects' section with two radio buttons: 'Program Definition' (selected) and 'Defaults for Technical Settings'. At the bottom of the 'Sub-Objects' section, there are two buttons: 'Display' (with a magnifying glass icon) and 'Change' (with a pencil icon).

Simply enter the program name and press the create button for new programs or change button for an existing program with the sub-object as program definition.

The definition is now displayed as below.

MDR

Run History

MDR Program: /BTR/MDR_SAMPLE_FLIGHT_REPORT
MDR Sample: Flight Report

Transform Prog.: /BTR/MDR_SAMPLE_FLIGHT_TRANALV
Interval Obj.: Customer Interval Object
Launch Transform program: ☐
Generate variant internally: ☐

Object: /BTR/MDR MDR application logs
Subobject: DEFAULT Default sub-object

Job Format ID:

Main Program: MDR Forms
Interval Generation Interval Processing Interval Collation

The main program is already populated from the first screen, a transformation program can now be configured if required.

The "Interval Object" object is also populated here, refer [here](#) for more information on intervals.


The program definition also allows the user to configure which application log object and sub-object any messages are written to that are called during the execution of the program. The default object and sub-object are /BTR/MDR/DEFAULT.

Furthermore, the transaction /BTR/MDR allows the developer to maintain the Main Program and Transformation Program directly instead of using the standard SAP transaction SE38, with the code buttons at the bottom linking directly to the right subroutines.

Default Technical Settings

The second sub-object managed through the transaction /BTR/MDR is “Defaults for Technical Settings” this screen contains two main sections. The first section “Defaults for Technical Settings” allows to set default values for a specific MDR program. Once set, these values will always appear on the Technical Settings’ sub screen for that program see [Technical Settings](#) under Running MDR Programs for more details.

MDR



Instance Settings

Label

Interval Settings

Perform processing using intervals of

Interval variant

Distribution

☒ Number of batch jobs across all servers


☐ Distribution according to server group

☐ Manual Distribution

☐ Run online as a single process (debugging mode)

Other settings

☐ Wait for run to complete

Distribution List 

Message log level

Technical Settings Access

☐ Background Program

☐ Lock Technical Settings

☐ Lock Expert Mode

The second section “Technical Settings Access” we will explore in more detail.

Background Program

NOTE: This functionality requires “Wait for run to complete” to be set.

This option suppresses the default display of the Run History screen after completion of an instance run. This is a useful feature that allows an MDR program to be called from another program without interrupting the latter with the Run History display.

The input field “Check for completion (in sec)” allows to set in seconds a time interval in which the parent job of a running MDR instance will wake up and check if all child parallel processes have completed. The default wake up and check time is 30 seconds which is suitable for very long running programs but not for speeding up, say, web services where every second counts.

Lock Technical Settings

This option allows to lock all input fields for Technical Settings. This is useful if when a program can repeatedly run with the same default values and users should not change those values. When this option is set, the Expert Mode in the Run History will be locked as well.

This restriction applies at program level and not at user level. That is, once set the Technical Settings will be locked for all users. Restrictions at user level can be implemented with the MDR enhancement spots (see MDR Enhancement Spots document).

Lock Expert Mode

This option is similar to “Lock Technical Settings”. The only difference is that on the Technical settings screen only the input fields under “Distribution” are locked. This allows the user to change settings like label name while protecting the more critical job distribution section from potential misuse. This option applies at program level as well. Restrictions at user level can be implemented with see the MDR Developers Guide [Authority Checks](#)

Interval Generation

When an MDR program uses Interval Objects, an Interval Variant needs to be created from the Interval Object, before the MDR program is run. An Interval Variant can be thought of as the set of Intervals that the MDR program is going to use. It is necessary that new Interval Variants are generated regularly (potentially before each batch run) to ensure that the intervals are split evenly as the data in the system grows.

To use an Interval Objects, they must first be configured into the framework via table /BTR/INTVALOBJ.

There are two different types of Interval Objects; standard SAP Mass Run Interval Objects and MDR Interval Objects. Both functions in the same manner, except for the generation of the Interval Variants is performed differently. The Intervals (or Interval Variant) are created before the MDR program is executed; this is done via the program /BTR/MDR_INTERVAL_GENERATION or alternately for standard SAP Interval variants via transaction FQD2. Either an Interval Count or Interval Size can be used as parameters to how the Intervals get generated.

The screenshot shows the SAP transaction 'MDR: Generate Interval Variant'. The interface includes a title bar, a menu bar, and a toolbar. The main area contains the following fields and sections:

- Interval Object:** A dropdown menu with 'Customer Interval Object' selected.
- Interval Variant:** A text field containing 'INT:CNT:10'.
- Intervals Section:** A table with two rows:
 - Interval Size:** An empty text field.
 - Interval Count:** A text field containing '10'.

The above example shows the generation of a new Interval Variant called INT:CNT:10 from the Interval Object "Customer Interval Object", with the requirement that 10 Intervals are to be created.

As can be seen below, the result is 10 generated Intervals.

Menu: List Edit Goto Settings System Help

Toolbar: [Icons for various functions]

MDR: Generate Interval Variant

Toolbar: [Icons for various functions]

Int.Object	Variant	Interval	Low	High
SCUSTOM	INT:CNT:10	1		00000514
SCUSTOM	INT:CNT:10	2	00000514	00000978
SCUSTOM	INT:CNT:10	3	00000978	00001442
SCUSTOM	INT:CNT:10	4	00001442	00001906
SCUSTOM	INT:CNT:10	5	00001906	00002370
SCUSTOM	INT:CNT:10	6	00002370	00002834
SCUSTOM	INT:CNT:10	7	00002834	00003298
SCUSTOM	INT:CNT:10	8	00003298	00003762
SCUSTOM	INT:CNT:10	9	00003762	00004226
SCUSTOM	INT:CNT:10	10	00004226	99999999

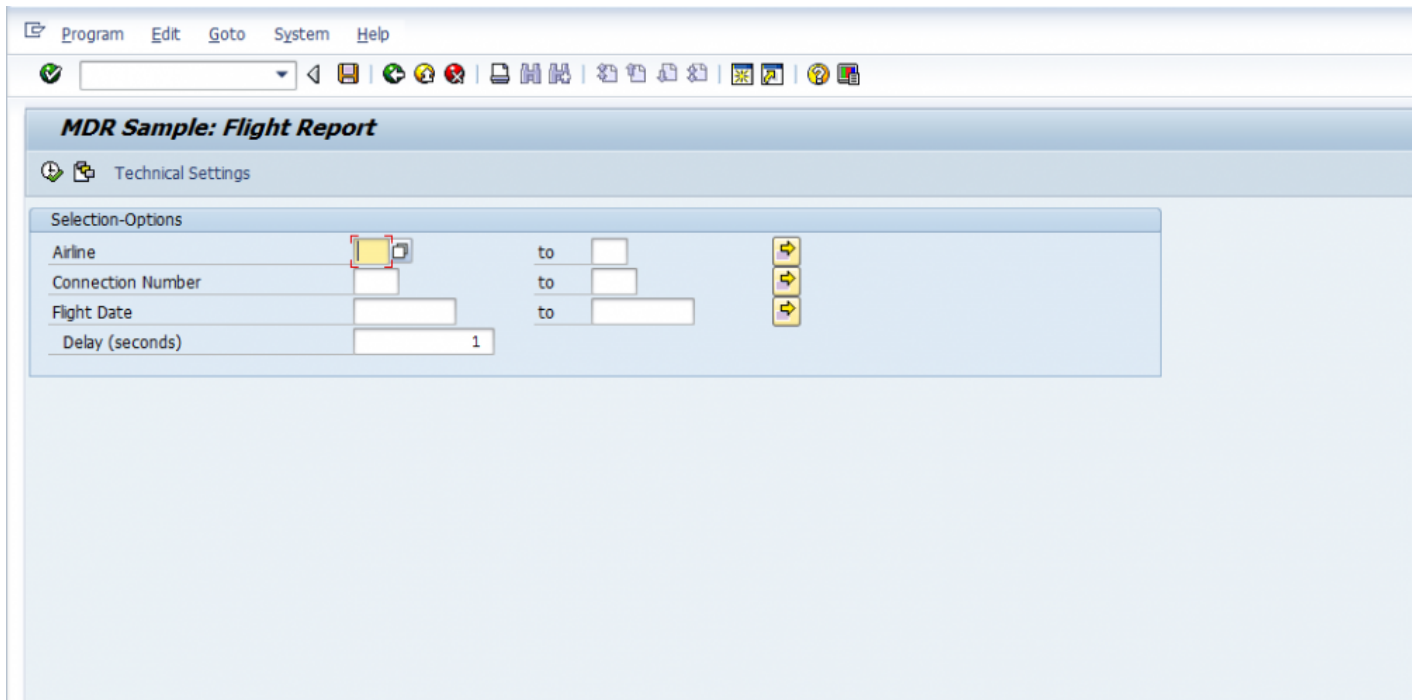
Running MDR Programs

Before running an MDR program the technical settings need to be completed, once running to administer the program see [Administering MDR Programs](#) section. For the first run of a newly developed program check the developers notes.

- [Technical Settings](#)
- [Developer Notes](#)

Technical Settings

The key part that the user sees is the “Technical Settings” button as below.



If you select the “Technical Settings” button, you will be prompted for the MDR specific technical settings.

The screenshot shows a software window titled "Diffuser Sample: Flight Report". It contains several sections for configuration:

- Instance Settings:** A text field labeled "Label" with the value "Lufthansa Flights".
- Interval Settings:** Two dropdown menus. The first is "Perform processing using intervals of" with the value "Sample: Flight Customers". The second is "Interval variant" with the value "INT:100 : 99 intervals".
- Distribution:** Four radio buttons. The first, "Number of batch jobs across all servers", is selected and has a text field with the value "5" next to it. The other three options are "Distribution according to server group", "Manual Distribution", and "Run online as a single process (debugging mode)".
- Other settings:** Two checkboxes, "Wait for run to complete" and "Launch Transformation Program after completed run", both of which are unchecked. Below them is a "Distribution List" text field with a dropdown arrow icon to its right. At the bottom of this section is a "Message log level" dropdown menu with the value "Other".

At the bottom right of the window is a toolbar with icons for "Check", "Save", and "Close".

These “Technical Settings” are important when executing an MDR program be it in a production environment, or when performing unit testing of your MDR program.


Note that you can setup [Default Technical Settings](#) and setup user authorizations to control the users ability to change these settings, for more information refer to the Mass Data Runtime – Developers Guide [Authority Checks](#).

An explanation for the function of each field on the “Technical Settings” screen is as below:

- Label – The first is a label that can be specified to identify this particular execution
- Perform processing using intervals of – This is the interval object confirmed in the [Program Definition](#)
- Interval Variant – The Interval variant provides you with a list of different pre-generated Intervals. As detailed in the section [Interval Generation](#), the interval variants are pre-generated using program /BTR/MDR_INTERVAL_GENERATION
- Number of batch jobs across all servers –This specifies the number of processes with which to run the MDR program

- Distribution according to server grouping –This allows the distribution of jobs over one server group to control the number of processors available to this MDR instance
- Manual Distribution –The server grouping above can also be distributed manually
- Run online as a single process (debugging mode) –This is only used when debugging MDR programs and ensures the whole program runs sequentially
- Wait for run to complete before finishing –This is often used when running MDR programs on-line or when executing them via a job scheduler. It will ensure the parent process waits until all child processes have completed. Once all child processes have finished, control is returned to the parent for completion
- Distribution List –After a program completes within the MDR framework it is able to send a SAP office document or external email to a set of recipients that can be specified here
- Message log level – Lower limit for the priority of messages output to the application log by MDR. For example, you can restrict output of informational application log messages by increasing the log level via this parameter

When using Dynamic Intervals as set out in the Mass Data Runtime – Developers Guide [Dynamic Interval Generation](#) the “Interval Settings” section will change and be replaced with the two options Interval Count and Interval Size introduced, this looks as per the screenshot below.

Instance Settings	
Label	<input type="text"/>
Interval Settings	
Interval Count	<input type="text"/>
Interval Size	<input type="text"/>
Distribution	
<input checked="" type="radio"/> Number of batch jobs across all servers	<input type="text" value="3"/>
<input type="radio"/> Distribution according to server group	<input type="text"/> <input type="text"/>
<input type="radio"/> Manual Distribution	<input type="text"/> <input type="text"/>
<input type="radio"/> Run online as a single process (debugging mode)	
Other settings	
<input type="checkbox"/> Wait for run to complete	
Distribution List	<input type="text"/> 
Message log level	<input type="text" value="Other"/>

The impact of the two fields is as below:

- Interval Count –This specifies the number of intervals (chunks) that the total amount of work to be done is to be broken up into
- Interval Size –This specifies the “number of objects” to be put into each interval to be then worked upon independently

Developer Notes

As a developer, your first step will be to ensure the program runs correctly with a single job, but with more than one interval. You should have the first run option selected for this, with 1 job in the corresponding field. When your interval processing, collation and presentation all work correctly with a single job (but more than one interval) then most likely your program is ready for running in parallel mode on large volumes of data.

This can be done using the same run option but with a higher number of jobs specified.

When you execute your MDR program and you specify the first run option, the execution will be in the background; MDR will immediately start the jobs and take you into the Program Monitoring transaction. Here you are able to refresh the display in order to determine the current status and progress of the execution.

The setting “Wait for run to complete before finishing” ensures that the master job stays active until all N sub jobs have completed. This may be useful if you require another process to begin when this one completes.

The “Distribution List” parameter is described in the following section Email Notification.

After the program has completed you will be able to view the results. You can access the run history via transaction /BTR/MDRH, or via transaction /BTR/MDR and the “Run History” button. You then view the results by selecting the Program run, and selecting the Transform button (or F5). This will display your results as you have implemented in the Transformation program.

Instance List Edit Goto Settings System Help

MDR: Run History

Transform Application Log Results Intervals Variants Jobs App. Servers

MDR Program	Report title	Run Count
/BTR/MDR_SAMPLE_FLIGHT_REPORT	MDR Sample: Flight Report	1

Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
American Airlines Flights	TENGLAND	16.01.2015	19:52:56		00:00:00	In Process	45%	0:00:59	4 (9)	◀	▶

Interval	Low	High	Status	Results
1	00000001	00000005	Completed	1
2	00000006	00000010	Completed	1
3	00000011	00000015	Completed	1
4	00000016	00000020	Completed	1
5	00000021	00000025	Completed	1
6	00000026	00000030	Completed	1
7	00000031	00000035	Completed	1
8	00000036	00000040	Completed	1
9	00000041	00000045	Completed	1
10	00000046	00000050	Completed	1
11	00000051	00000056	Completed	1
12	00000057	00000062	Completed	1
13	00000063	00000067	Completed	1
14	00000068	00000072	Completed	1
15	00000073	00000077	Completed	1
16	00000078	00000082	Completed	1
17	00000083	00000088	Completed	1
18	00000089	00000094	Completed	1

Administering MDR Programs

MDR provides the advanced user a number of powerful administrative capabilities via the “Run History” report accessible via transaction /BTR/MDRH (see screen below). These capabilities provide a powerful way of managing your MDR programs.

MDR: Run History											
MDR Program		Report title									Run Count
/BTR/MDR_SAMPLE_FLIGHT_REPORT		MDR Sample: Flight Report									1
Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
test	CWAGNER	25.09.2013	13:05:16		00:00:00	In Process	14%	00:02:12	2	◀	▶
Interval	Low	High	Status	Results							
1	00000001	00000047	Completed	1							
2	00000048	00000098	Completed	1							
3	00000099	00000173	Completed	1							
4	00000174	00000221	Completed	1							
5	00000223	00000273	Completed	1							
6	00000274	00000332	Completed	1							
7	00000333	00000379	Completed	1							
8	00000380	00000426	Completed	1							
9	00000427	00000473	Completed	1							
10	00000474	00000520	Completed	1							

- [Expert Mode](#)
- [Results](#)
- [Intervals](#)
- [Variants](#)
- [App Servers](#)
- [Increase Jobs](#)
- [Decrease Jobs](#)
- [Stop](#)
- [Resume](#)
- [Delete](#)
- [Force Error](#)

Expert Mode

To access the expert mode click the expert mode button as below.

MDR: Run History									
Transform Application Log									
Diffuser Program		Report title							Run Count
/BTR/SAMPLE_FLIGHT_REPORT		Diffuser Sample: Flight Report							26
Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	
Qantas Airways Flights	TENGLAND	19.12.2014	15:56:53	19.12.2014	15:58:36	Finished	100%		
American Airlines Flights	TENGLAND	19.12.2014	15:38:18	19.12.2014	15:40:22	Finished	100%		
All Flights	TENGLAND	19.12.2014	15:37:45	19.12.2014	15:39:26	Finished	100%		
BA Flights	TENGLAND	19.12.2014	15:36:11	19.12.2014	15:36:15	Finished	100%		
Run Name	TENGLAND	13.12.2014	01:11:16	13.12.2014	01:12:21	Finished	100%		

If authorized a number of other functions will be revealed.

MDR: Run History											
Transform Application Log Results Intervals Variants App. Servers											
Diffuser Program		Report title							Run Count		
/BTR/SAMPLE_FLIGHT_REPORT		Diffuser Sample: Flight Report							26		
Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
Qantas Airways Flights	TENGLAND	19.12.2014	15:56:53	19.12.2014	15:58:36	Finished	100%				
American Airlines Flights	TENGLAND	19.12.2014	15:38:18	19.12.2014	15:40:22	Finished	100%				
All Flights	TENGLAND	19.12.2014	15:37:45	19.12.2014	15:39:26	Finished	100%				
BA Flights	TENGLAND	19.12.2014	15:36:11	19.12.2014	15:36:15	Finished	100%				
Run Name	TENGLAND	13.12.2014	01:11:16	13.12.2014	01:12:21	Finished	100%				

Results

To access the raw results stored against the instance click the results button as below.

MDR: Run History

Transform Application Log **Results** Intervals Variants App. Servers

Diffuser Program	Report title	Run Count
/BTR/SAMPLE_FLIGHT_REPORT	Diffuser Sample: Flight Report	26

Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
Qantas Airways Flights	TENGLAND	19.12.2014	15:56:53	19.12.2014	15:58:36	Finished	100%				
American Airlines Flights	TENGLAND	19.12.2014	15:38:18	19.12.2014	15:40:22	Finished	100%				
All Flights	TENGLAND	19.12.2014	15:37:45	19.12.2014	15:39:26	Finished	100%				
BA Flights	TENGLAND	19.12.2014	15:36:11	19.12.2014	15:36:15	Finished	100%				
Run Name	TENGLAND	13.12.2014	01:11:16	13.12.2014	01:12:21	Finished	100%				

You can also select an interval and view the raw results stored against each interval.

Intervals

By drilling down on the program name the user will access the programs instance runs. Select an instance and click “Intervals” to display the intervals to that specific instance run.

MDR: Run History											
Transform Application Log Results Intervals Variants App. Servers Print Export Help Refresh Close											
Diffuser Program		Report title									Run Count
/BTR/SAMPLE_FLIGHT_REPORT		Diffuser Sample: Flight Report									26
Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
Qantas Airways Flights	TENGLAND	19.12.2014	15:56:53	19.12.2014	15:58:36	Finished	100%				
American Airlines Flights	TENGLAND	19.12.2014	15:38:18	19.12.2014	15:40:22	Finished	100%				
All Flights	TENGLAND	19.12.2014	15:37:45	19.12.2014	15:39:26	Finished	100%				
BA Flights	TENGLAND	19.12.2014	15:36:11	19.12.2014	15:36:15	Finished	100%				
Run Name	TENGLAND	13.12.2014	01:11:16	13.12.2014	01:12:21	Finished	100%				

The details of each intervals are then displayed as below.

MDR: Run History											
Transform Application Log Results Intervals Variants App. Servers Print Export Help Refresh Close											
Diffuser Program		Report title									Run Count
/BTR/SAMPLE_FLIGHT_REPORT		Diffuser Sample: Flight Report									26
Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
BA Flights	TENGLAND	19.12.2014	15:36:11	19.12.2014	15:36:15	Finished	100%				
Interval	Low	High	Status	Results							
1	00000001	00000047	Completed	0							
2	00000048	00000098	Completed	0							
3	00000099	00000173	Completed	0							
4	00000174	00000221	Completed	0							
5	00000223	00000273	Completed	0							
6	00000274	00000332	Completed	0							
7	00000333	00000379	Completed	0							
8	00000380	00000426	Completed	0							
9	00000427	00000473	Completed	0							
10	00000474	00000520	Completed	0							
11	00000521	00000567	Completed	0							
12	00000568	00000614	Completed	0							

Variants

To access the raw results stored against the instance click the results button as below.

MDR: Run History											
Transform Application Log Results Intervals Variants App. Servers											
Diffuser Program		Report title									Run Count
/BTR/SAMPLE_FLIGHT_REPORT		Diffuser Sample: Flight Report									26
Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
Qantas Airways Flights	TENGLAND	19.12.2014	15:56:53	19.12.2014	15:58:36	Finished	100%				
American Airlines Flights	TENGLAND	19.12.2014	15:38:18	19.12.2014	15:40:22	Finished	100%				
All Flights	TENGLAND	19.12.2014	15:37:45	19.12.2014	15:39:26	Finished	100%				
BA Flights	TENGLAND	19.12.2014	15:36:11	19.12.2014	15:36:15	Finished	100%				
Run Name	TENGLAND	13.12.2014	01:11:16	13.12.2014	01:12:21	Finished	100%				

This enables the variant details entered on the selection screen to be viewed.

ABAP: Selections of Variant MDR:0000007276											
Catalog Values Attributes											
Objects for selection screen											
Selection Scrns	Field name	Type	I/E	Option	frm	to					
1000	__MDRXX	P									
1000	Airline	S	I	EQ	BA						
1000	Connection Number	S									
1000	Flight Date	S									
2830	__MDRIL	P			BA Flights						
2830	__MDRIO	P			SCUSTOM						
2830	__MDRIV	P			INT:99						
2830	__MDRIC	P			0000000000						
2830	__MDRIS	P			0000000000						
2830	__MDRD1	P			X						
2830	__MDRNJ	P			0000000003						
2830	__MDRD2	P									
2830	__MDRNJD	P			0000000000						
2830	__MDRSG	P									
2830	__MDRD4	P									
2830	__MDRJ1	P			0000000000						

App Servers

To view the application servers click the App Server button as below.

MDR: Run History											
Transform Application Log Results Intervals Variants App. Servers											
Diffuser Program		Report title									Run Count
/BTR/SAMPLE_FLIGHT_REPORT		Diffuser Sample: Flight Report									26
Instance Name	Started By	Start Date	Start Time	End Date	End Time	Instance Status	Comp	Remaining	Active Job	-	+
Qantas Airways Flights	TENGLAND	19.12.2014	15:56:53	19.12.2014	15:58:36	Finished	100%				
American Airlines Flights	TENGLAND	19.12.2014	15:38:18	19.12.2014	15:40:22	Finished	100%				
All Flights	TENGLAND	19.12.2014	15:37:45	19.12.2014	15:39:26	Finished	100%				
BA Flights	TENGLAND	19.12.2014	15:36:11	19.12.2014	15:36:15	Finished	100%				
Run Name	TENGLAND	13.12.2014	01:11:16	13.12.2014	01:12:21	Finished	100%				

This then displays the available App Servers

SAP Servers			
Release Notes			
Server Name	Host Name	Message Types	Status
bti101_D01_01	bti101	Dialog Batch Update Upd2 Spool Enqueue ICM	Active

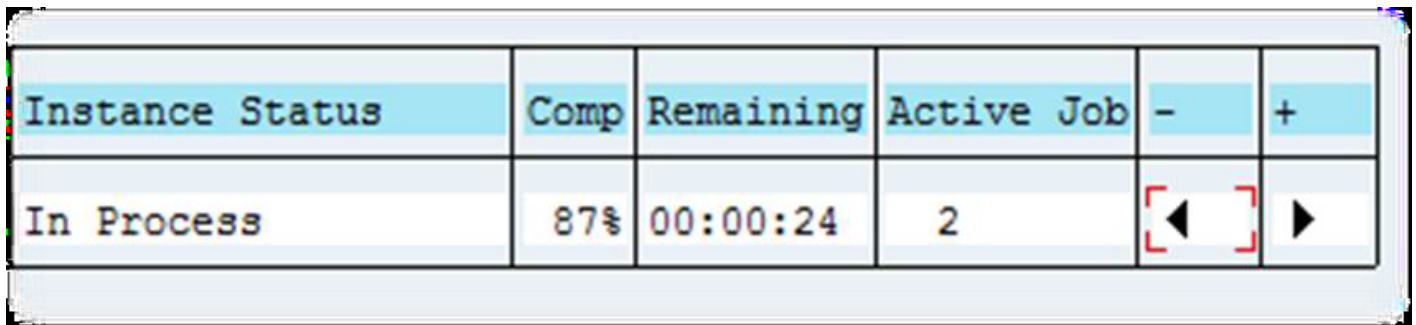
Increase Jobs

On the “Run History” report, the user can see historical instances of a program, as well as any currently executing program instances. The user can also see the number of active jobs for each program instance currently running. It is now possible to select an instance and then the “Increase Jobs” option, or alternately the right arrow icon against the instance. By selecting this option another job is scheduled immediately to assist in processing any unprocessed intervals. After a short period, and after selecting refresh the user will notice that the number of active jobs increase by 1, or by the number of times the increase jobs option is selected.

Instance Status	Comp	Remaining	Active Job	-	+
In Process	14%	00:01:30	3	◀	▶

Decrease Jobs

It is also possible to select an instance and decrease the number of jobs by selecting the “Decrease Jobs” option, or selecting the left arrow icon beside the instance. It is only possible to select this option for instances that are currently “In Progress”, and have more than one active job. MDR will prevent you decreasing the number of jobs to zero, if it is the users intention to stop the processing of the MDR report, then the user should select the “Stop” option. After a short period, and after selecting refresh the user will notice that the number of active jobs decreases by 1.

A screenshot of a web application interface showing a table with instance details. The table has six columns: 'Instance Status', 'Comp', 'Remaining', 'Active Job', and two control columns with '-' and '+' signs. The first row shows an instance in 'In Process' status, 87% complete, with 24 seconds remaining and 2 active jobs. The control columns for the first row contain a left arrow icon in a red box and a right arrow icon.

Instance Status	Comp	Remaining	Active Job	-	+
In Process	87%	00:00:24	2	◀	▶


Stop

It is possible to Stop (or Pause) a program instance using the “Stop” option. By selecting this option after selecting a program instance, the MDR framework tells the currently executing jobs to no longer process any more intervals after it completes the processing of the current interval. The status of the Instance, and unprocessed intervals changes to “Stopped”. This is a powerful option that is used typically when an MDR program needs to be stopped temporarily due to the need to free up batch resources, or stopped permanently if the report run is no longer required.

The benefit MDR has over the traditional approach to executing reports is that the MDR program does not need to start over again, execution can continue from where it left off. The intervals that have already been processed do not need to be reprocessed unless of course it is deemed necessary by the user due to perhaps a substantial amount of time passing before the program is allowed to continue. It is only possible to stop an Instance that is currently in the “In Progress” status.

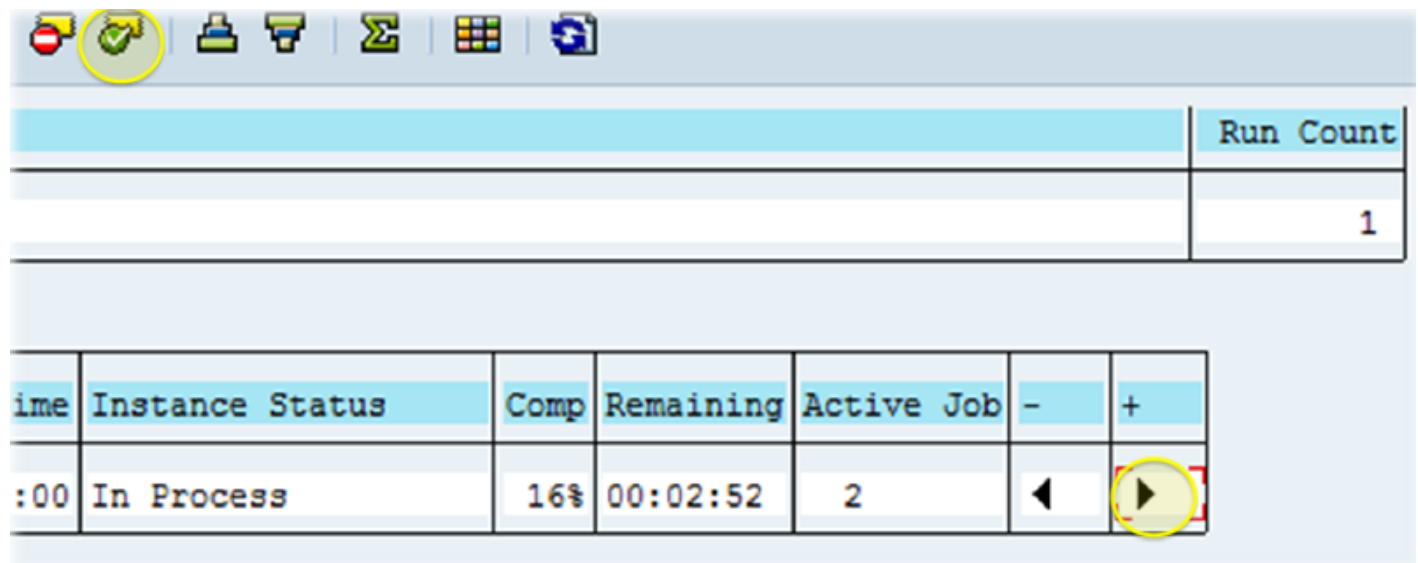
When the instance is paused, the MDR framework will not immediately stop all jobs that are currently running. It will instead prevent any new intervals from being started.

The more intervals there are the more control over the execution of the instance an administrator will have.

						
						Run Count
						1
Time	Instance Status	Comp	Remaining	Active Job	-	+
:00	Stopped	16%	00:05:40			

Resume

The “Resume” option allows the selected program instance to continue from the point it was stopped or paused. This option uses the Technical Settings of the original program instance to reschedule the report. By resuming an instance it does not reprocess any intervals that have a status of “Completed”, it changes the status of a “Paused” interval to “Ready”. The restart option can only be selected for instances with the status “Paused” or “Error”.



The screenshot shows the Mass Data Runtime interface. At the top, there is a toolbar with several icons. The 'Resume' icon, which is a green circle with a white play button, is highlighted with a yellow circle. Below the toolbar, there is a table with two columns: 'Run Count' and a value '1'. Below this, there is another table with columns: 'Time', 'Instance Status', 'Comp', 'Remaining', 'Active Job', '-', and '+'. The first row of this table shows ':00', 'In Process', '16%', '00:02:52', '2', and a play button icon (highlighted with a yellow circle) in the '-' column.

						Run Count
						1

Time	Instance Status	Comp	Remaining	Active Job	-	+
:00	In Process	16%	00:02:52	2	◀	▶

Delete

It is possible to delete a program instance by selecting the instance and then the “Delete” option, this in turn deletes all the intervals and results belonging to the program instance. After the delete option is selected the user is faced with a confirmation window to ensure the deletion was intentional. This option is particularly useful in a testing environment and with instances that have errored. It is only possible to delete instances with the status “Error” or “Completed”. The system will not allow an instance “In Progress” to be deleted due to possible data inconsistencies.

Force Error

By selecting an Instance, right-clicking and then the “Force error” option, the status of the program instance is changed to “Error”. This allows instances that have technically completed successfully to be changed to Error. This is basically an override function. It is only possible to set an instance to “Error” if there are no active jobs executing the instance.

Scheduling MDR Programs

An MDR program can be scheduled just like any other background program. Typically this is done using the standard transaction SM36. The program variants can also be saved as per normal.

MDR in most cases, however, does require another program to be scheduled for it to operate efficiently in a production environment. The program function is to regenerate the Interval Variant. The purpose of regenerating an Interval Variant is such that as the master or transactional data grows, the intervals can be recalculated to ensure that each interval is evenly spread. This then ensures the MDR program is processed as efficiently as possible.

The program /BTR/MDR_INTERVAL_REGENERATION is used for this purpose. This job should typically be scheduled nightly at the beginning of the batch window, and can be executed for individual Interval Objects, individual Interval Variants, or for all Interval Variants by adjusting the parameters on the selection screen. For the Interval Regeneration to operate, you will need to configure the table /BTR/INTVALVARC. Here you define an Interval Object, Interval Variant and the refresh age. The refresh age defines how frequently the Interval Variant is refreshed. For example if for Object SCUSTOMID, Variant SAMPLE, if the refresh age is 7, the interval variant will only be regenerated every 7 days, even if the regeneration job is scheduled nightly. This functionality allows you to avoid scheduling individual regeneration jobs in different reoccurring cycles. You can override the refresh age functionality by selecting the "Force regeneration" check-box.

Exception Handling and Application Logs

There are numerous situations where an MDR program is required to take some action. This action can range from simply outputting informational messages to extreme cases where the program is required to abend. A number of framework components exist in MDR so that your program can indicate to the framework the current processing status and the manner in which your program is reacting to a particular situation.

The interval processing subroutine is given a range of objects to process (the interval itself). During the processing of these objects, a situation may occur where a particular object in the range is invalid. An example of this may be that there is a problem with the database integrity of that particular object (e.g. an invalid enumeration or configuration is missing). The MDR program has the option to either:

1. Output a message to the application log
2. Skip the current object and move on to the next
3. Abend the program

Alternatively, the program may be required to do a combination of the above. MDR provides the developer with a number of statements to log exceptions:

The sample code below shows how to code messages raised inside MDR to be read via the application log.

```
FORM mdr_interval_processing
  USING x_interval TYPE /btr/st_interval_values.
  ...

  IF lv_error EQ gc_true.
    mdr_msg_put 1 'E' '100' 'MSGCLASS' space space space space.
  ENDIF.
  ...

ENDFORM.
```

The statement `mdr_msg_put` will output a message to the application log. It expects to be provided with the following parameters:

1. Priority – The log level represents the importance of the message. This allows the person running the program to determine what messages are outputted to the application log. The five categories that can be used are:
 - Critical (1) – Used to represent that the message is critical
 - Important (2) – Used to represent that the message is important

Normal (3) – Used to represent that the message is of normal importance

Information (4) – Used to represent that the message is an informational message

Debugging (5) – Used to represent that the message is for debugging purposes

2. Message Type – The message type is the standard SAP message type and represents the type of message. It must be one of the following five options:

Error (E) – Error message

Success (S) – Success message

Information (I) – Informational message

Warning (W) – Warning message

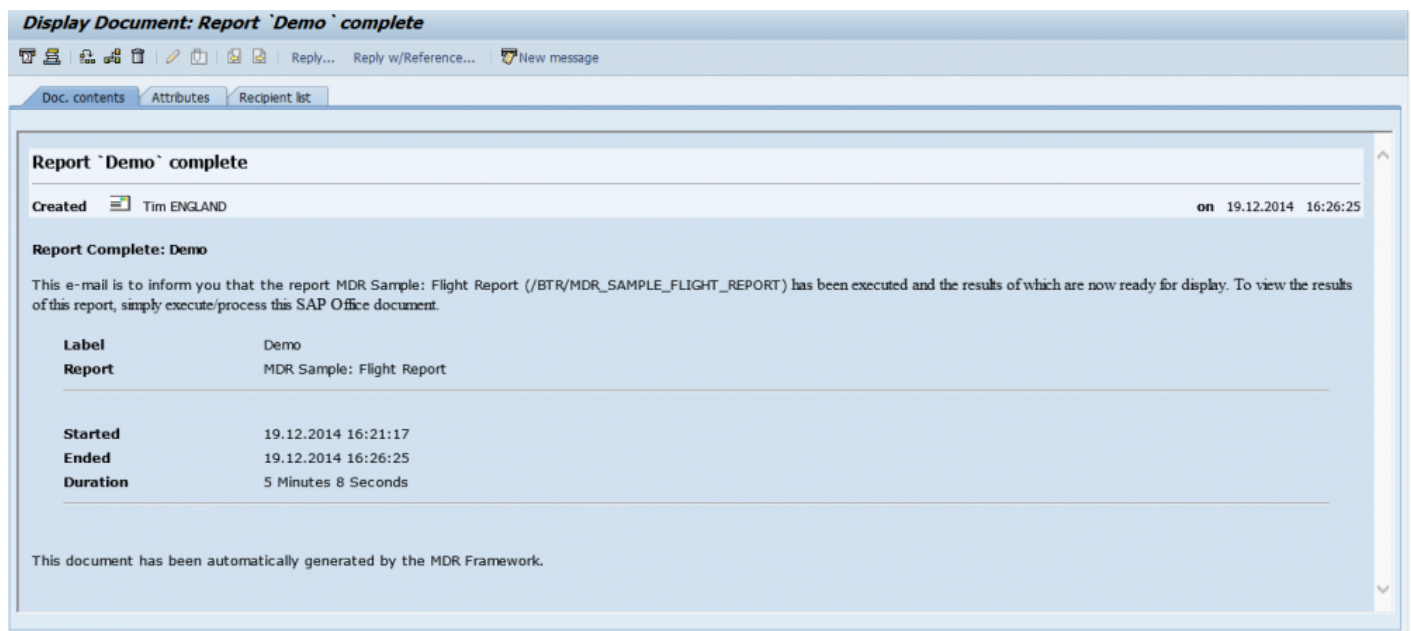
Abend (A) – Abend message – stops the program

3. Message Number – The message number is the standard SAP message number as defined in the message class.
4. Message Class – The message class represents the standard SAP collection of messages.
5. Message variables – When you define a message in the standard SAP message class, placeholders for variables may be defined. These placeholder variables can be passed in here. There may be up to four message variables. If the message has no variables or there are less than four, then you should use the “space” keyword – i.e. you must provide all four variables even if you simply use the space keyword four times.

Email Notification

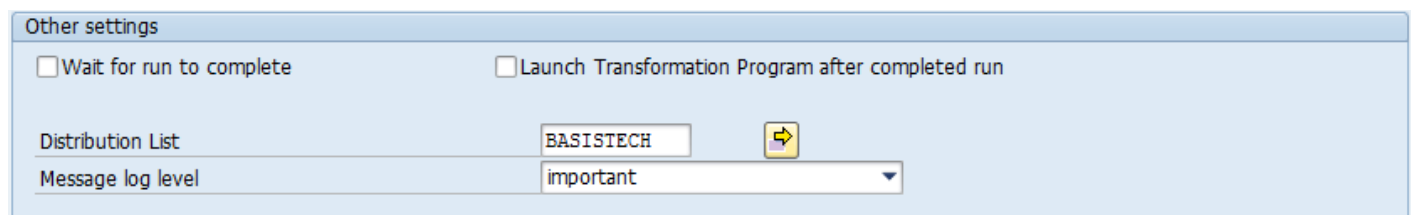
On the technical settings tab of an MDR program there is a parameter “Distribution List”.

MDR provides the functionality to automatically send a notification to users SAP inbox as defined on the Distribution List once the report results are complete. This improves the flow of information by ensuring that results are passed onto the business users at the soonest possible moment. It is no longer necessary to rely on business processes to notify and pass on a reports output to the business, instead a link to the report output is sent automatically to the users inbox. The results can be accessed by this link.



Providing SAP Connect is configured in your system, these results can also be sent directly to the corresponding users external email address. The link provides direct access to the report output, even if the user is not initially logged into your SAP system.

This configuration is extremely simple. All it involves is configuring the appropriate users on the Technical Settings screen, under Distribution List.



For consistency, MDR provides the ability to send email notification of results of reports not migrated to MDR format. MDR includes a program used to monitor background reports that have been scheduled and determine whether a notification is to be sent to a user or a group of users indicating the completion of the reports.

The notification is delivered in the same manner as MDR delivers the notification. This has the added advantage that a link is sent to a user rather than the actual spool attachment (which may be very large). Furthermore, it is possible to configure the email template used for the delivery of the notification. In order to deliver notifications for standard/custom-built non-MDR reports and programs, the following is required:

1. Initially, the configuration for the notification must be maintained in the view cluster (/BTR/CREPNTF) to determine whom the notification should be sent to and with what template. This is done via transaction SM34. The job name is critical in the configuration and will be used to determine if an entry has been executed in the job schedule. The job (when configured to run in background) must have the same name as entered in the configuration. If it doesn't have the same name – it will not be picked up and hence no notifications will be sent.

For each job you setup, you must also specify who will be notified when the report is finished.

Recipients are classified in various categories including:

1. Internal SAP Username
2. External Address
3. Organizational Unit (As defined in the HR Org Structure)
4. Internet Address

You can configure one or more of the above for each job setup. For each recipient, you may also specify the template that is used when the notification is sent. This can be defaulted to /BTR/MDR_REPORT_NOTIFICATION. If you want to customize the notification template at an individual level, this can also be specified against the recipient. The recipient level template will always take precedence over what has been specified at the job level.

2. During the normal batch schedule, the reports that have configured in the first step are executed as per normal (in background).
3. The report /BTR/MDR_REPORT_NOTIFICATION must then be scheduled daily at the end of the batch schedule. This will find all background jobs that have been executed (restricted on selection criteria if required) and send the appropriate notification to the configured user(s).

Notifications are generally delivered to a user's SAP Inbox. The document that is sent contains the particulars about when the report was started, when it completed and how long it took. It also specifies the report title and technical name.

The inbox item must be "Executed" by the user in order for them to be taken to the report output. This is a simple matter of right-clicking on the Inbox item and selecting "Execute". The user will then be taken to the spool output immediately.

Debugging and Troubleshooting

Debugging MDR Programs

One of the major benefits provided with MDR is that the developer is able to debug their program in much the same way as they would when they develop a normal program.

When you first execute your MDR program you will be presented with the selection-screen. On the “Technical Settings” Tab, the option “Run online as a single process (debugging mode)”, allows the program to be run online. This means that any break points that the developer has in their code, either hard-coded or dynamically set from the editor, will be stopped at.

The screenshot shows a configuration window titled "Diffuser Sample: Flight Report". It contains several sections for setting up the program execution:

- Instance Settings:** A text field labeled "Label" with the value "Lufthansa Flights".
- Interval Settings:** Two dropdown menus. The first is labeled "Perform processing using intervals of" and has the value "Sample: Flight Customers". The second is labeled "Interval variant" and has the value "INT:100 : 99 intervals".
- Distribution:** Three radio button options: "Number of batch jobs across all servers" (set to 2), "Distribution according to server group", and "Manual Distribution". Below these is a red-bordered box containing the selected option: "Run online as a single process (debugging mode)".
- Other settings:** Two checkboxes: "Wait for run to complete" and "Launch Transformation Program after completed run". Below these are two more fields: "Distribution List" (with a refresh icon) and "Message log level" (set to "Other").

At the bottom right of the window, there is a toolbar with icons for a clock, a checkmark, a document, and a close button, along with the text "Check".

Troubleshooting

When converting an existing program to use the MDR framework the obvious way to help find problems is to run the original program and MDR program side by side to check the results, to remove the multiple processing and intervals as possible issues keep the MDR program you can run it with just one large interval.

Typical bugs introduced from converting a program to use MDR are:

- Global variables not being cleared at the end of the interval processing subroutine, if you have no problem with one large interval the problem will most likely be around clearing global variables
- Repeating code in interval processing that is only required once per background job
- Variables not being stored for the transformation program, don't forget anything that needs to be displayed in the results needs to be stored into the results and retrieved by the transformation program

Advanced Concepts

The details of advanced programming concepts are available here:

- [Dynamic Interval Generation](#)
- [Authority Checks](#)

Dynamic Interval Generation

In some situations it may be necessary that you want the Intervals to dynamically change each time you execute an MDR program. This is an alternate method of Interval Generation to the more commonly used Interval Object method. MDR uses the concept of an Interval Generator to do this. An Interval Generator allows you to write ABAP code to do this dynamic interval creation. A custom Interval Generator can be implemented using the subroutine `mdr_interval_generator` in the Main program of the MDR program.

The impact on the selection screen is that the interval size and interval count options are displayed on the “Technical Setting” screen, see the [Technical Settings](#) section for details on this.

The MDR subroutine `mdr_interval_generator` is called at the beginning of the MDR program. This subroutine is used to define the intervals that will be processed. The subroutine provides a single input structure that contains both an interval count and an interval size. These are values that are set at run-time, and provide information to the program on how it should break up the processing. The `CHANGING` parameter is a list of Intervals to be processed. The developer writing the MDR program needs to implement logic to populate the `LOW` and `HIGH` interval values of `yt_intervals`.

```
*-----*
* FORM mdr_interval_generation
*-----*
* This form is called by the MDR architecture to generate
* interval ranges that can be coded as per your own requirements
*-----*
FORM mdr_interval_generation
  USING      x_input          TYPE /btr/st_intgen_input
  CHANGING   yt_intervals     TYPE /btr/tt_mdr_interval_values.

DATA :
  lv_interval      LIKE LINE OF yt_intervals.

DATA:
  lv_interval_size      TYPE i,
  lv_interval_index     TYPE i,
  lv_interval_index_1   TYPE i,
  lv_remainder          TYPE i,
  lv_count              TYPE i,
  lv_count_numc         TYPE numc10,
  lv_count_intervals    TYPE i,
  lv_index              TYPE syindex,
  lv_low_index          TYPE syindex,
  lv_flag               TYPE c.
```



```
DATA: lt_sbook TYPE sbook,
      ls_sbook TYPE sbook.

* Select bookings
SELECT *
  FROM sbook
  INTO TABLE lt_sbook
 WHERE custid in s_custid

SORT lt_sbook.

DELETE ADJACENT DUPLICATES FROM lt_sbook.

DESCRIBE TABLE lt_sbook LINES lv_count.

IF x_input-interval_size IS INITIAL.

*   Determine the size of each interval
    lv_interval_size = lv_count DIV x_input-interval_count.
    lv_remainder      = lv_count MOD x_input-interval_count.

    IF NOT lv_remainder IS INITIAL.

        ADD 1 TO lv_interval_size.
    ENDIF.

ELSE.

    lv_interval_size = x_input-interval_size.

ENDIF.

lv_interval_index = lv_interval_size.

* Find the first low for the interval
CLEAR ls_sbook.
READ TABLE lt_sbook INTO ls_sbook INDEX 1.

lv_interval-low = ls_sbook-custid.

DO.

* Get the last number in this package of data
  CLEAR ls_sbook.
  READ TABLE lt_sbook INTO ls_sbook INDEX lv_interval_index.

  IF sy-subrc EQ 0.

*   Provided something was found store this in the high
```

```
lv_interval-high = ls_sbook-custid.

APPEND lv_interval TO yt_intervals.
CLEAR lv_interval.

* Add one to the interval index to find the new low for the next interval
  lv_interval_index_1 = lv_interval_index + 1.

  CLEAR ls_sbook.
  READ TABLE lt_sbook INTO ls_sbook INDEX lv_interval_index_1.
  IF sy-subrc = 0.
*   If a record is found we have a new interval to create so create the low
*   else interval creation is complete so exit
    lv_interval-low = ls_sbook-custid.
  ELSE.
    EXIT.
  ENDIF.
  lv_interval_index = lv_interval_index + lv_interval_size.

ELSE.

*   Fill the last entry of the intervals with the last entry
  CLEAR ls_sbook.
  READ TABLE lt_sbook INTO ls_sbook INDEX lv_count.

  IF ls_sbook-custid >= lv_interval-low .
    lv_interval-high = ls_sbook-custid.
    APPEND lv_interval TO yt_intervals.
  ENDIF.
  EXIT.

ENDIF.

ENDDO.

ENDFORM.
```

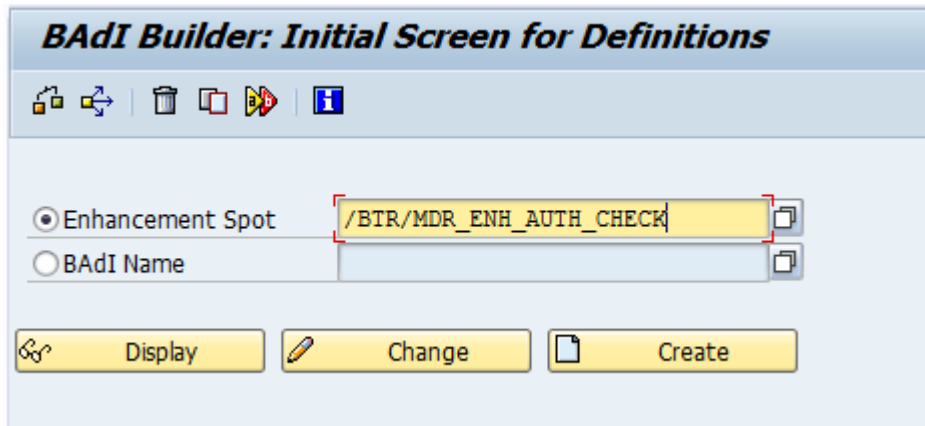
Authority Checks

To implement authorizations into MDR the following steps allow you to control which users can control the technical parts of MDR, either in the technical settings popup or the expert mode in the MDR Run History, using normal SAP authorization objects.

- [Implementation](#)
- [Technical Settings](#)
- [Expert Mode](#)
- [Individual Actions](#)

Implementation

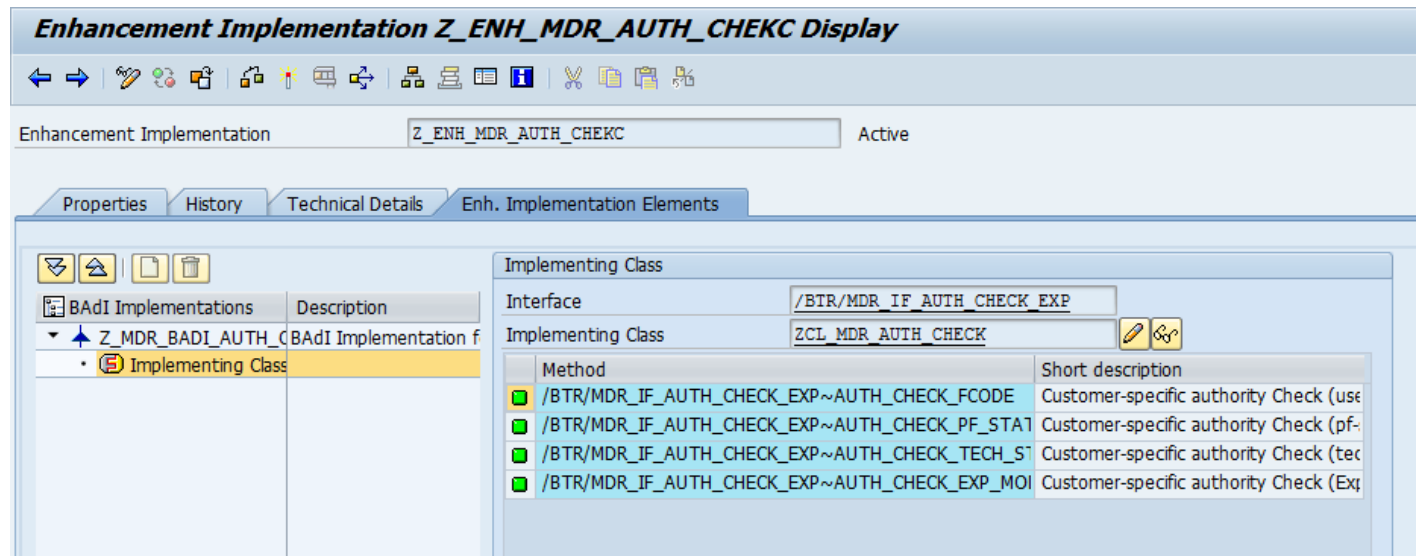
The MDR Framework contains enhancement spot /BTR/MDR_ENH_AUTH_CHECK. Go to SAP standard transaction SE18 to access it.



This enhancement spot contains BAdI definition /BTR/MDR_BADI_AUTH_CHECK_EXP which can be implemented to restrict access to both the Expert Mode functionality and the Technical Settings of MDR programs.

To implement BAdI definition /BTR/MDR_BADI_AUTH_CHECK_EXP create a custom class using interface /BTR/MDR_IF_AUTH_CHECK_EXP in SE18 (More information on how to implement a BAdI can be found in SAP standard documentation).

Once your class implementation of BAdI definition /BTR/MDR_BADI_AUTH_CHECK_EXP is in place, implement interface method AUTH_CHECK_TECH_STTGS.



The example above uses custom implementing class ZCL_MDR_AUTH_CHECK. Click on the method name to drill into its implementation and add your custom code. The signature of this method imports the program ID so you can set restrictions at program level.

The following sections will demonstrate how to implement controls around Technical Settings, Expert Mode and Individual Actions.

Technical Settings

There are two options to control the authorizations for Technical Settings.

When the changing parameter LCK_EXPT_MODE is set to true only the job distribution section of the Technical Settings screen is grayed out (see below).

Class Builder: Class ZCL_MDR_AUTH_CHECK Display

Navigation icons: Back, Forward, Undo, Redo, Copy, Paste, Find, Replace, Print, Run, Stop, Breakpoint, Comment, Uncomment, Zoom In, Zoom Out, Full Screen, Help, Pattern, Pretty Printer, Signature.

Ty.	Parameter	Type spec.	Description
▶	PROGRAMOID	TYPE /BTR/OID OPTIONAL	MDR: Internal Object ID
▶	LCK_EXPT_MODE	TYPE /BTR/LCKEXMD OPTIONAL	MDR: Lock Expert Mode
▶	LCK_TECH_SETT	TYPE /BTR/LCKTSETT OPTIONAL	MDR: Lock Technical Settings

Method: /BTR/MDR_IF_AUTH_CHECK_EXP~AUTH_CHECK_TECH_STTGS Active

```

1  METHOD /btr/mdr_if_auth_check_exp~auth_check_tech_sttgs.
2
3      AUTHORITY-CHECK OBJECT 'S_TCH_S'
4          ID 'PROID' FIELD programoid
5          ID 'ACTVT' FIELD '03'.
6
7      IF sy-subrc <> 0.
8          lck_expt_mode = 'X'.
9      ENDIF.
10
11  ENDMETHOD.
  
```

Technical Settings will look like this:

The screenshot shows a configuration window titled "Diffuser Sample: Flight Report". It is divided into four main sections: "Instance Settings", "Interval Settings", "Distribution", and "Other settings".

- Instance Settings:** Contains a "Label" field with the value "American Airlines Flights".
- Interval Settings:** Contains two dropdown menus. The first, "Perform processing using intervals of", is set to "Sample: Flight Customers". The second, "Interval variant", is set to "INT:100 : 99 intervals".
- Distribution:** Contains three radio buttons and two input fields. The first radio button, "Number of batch jobs across all servers", is selected and has an input field with the value "2". The second radio button, "Distribution according to server group", has an input field with the value "0" and a dropdown menu. The third radio button, "Manual Distribution", has an input field with the value "0". The fourth radio button, "Run online as a single process (debugging mode)", is unselected.
- Other settings:** Contains two checkboxes: "Wait for run to complete" and "Launch Transformation Program after completed run", both of which are unselected. Below these are a "Distribution List" field with an empty input box and a button with a right-pointing arrow, and a "Message log level" dropdown menu set to "Other".

At the bottom right of the window, there is a toolbar with icons for a clock, a lock, a "Check" button, a save icon, and a close icon.

When the changing parameter LCK_TECH_SETT is set to true all input fields of the Technical Settings screen is grayed out (see below).

Class Builder: Class ZCL_MDR_AUTH_CHECK Display

Ty.	Parameter	Type spec.	Description
▶	PROGRAMOID	TYPE /BTR/OID OPTIONAL	MDR: Internal Object ID
▶▶	LCK_EXPT_MODE	TYPE /BTR/LCKEXMD OPTIONAL	MDR: Lock Expert Mode
▶▶	LCK_TECH_SETT	TYPE /BTR/LCKTSETT OPTIONAL	MDR: Lock Technical Settings

Method

/BTR/MDR_IF_AUTH_CHECK_EXP~AUTH_CHECK_TECH_STGS

Active

```

1  METHOD /btr/mdr_if_auth_check_exp~auth_check_tech_stgs.
2
3      AUTHORITY-CHECK OBJECT 'S_TCH_S'
4          ID 'PROID' FIELD programoid
5          ID 'ACTVT' FIELD '03'.
6
7      IF sy-subrc <> 0.
8          lck_tech_sett = 'X'.
9      ENDIF.
10
11  ENDMETHOD.

```

Technical Settings will look like this:

Diffuser Sample: Flight Report

Instance Settings

Label American Airlines Flights

Interval Settings

Perform processing using intervals of Sample: Flight Customers

Interval variant INT:100 : 99 intervals

Distribution

☒ Number of batch jobs across all servers 2

☐ Distribution according to server group 0

☐ Manual Distribution

0

☐ Run online as a single process (debugging mode)

Other settings

☐ Wait for run to complete ☐ Launch Transformation Program after completed run

Distribution List

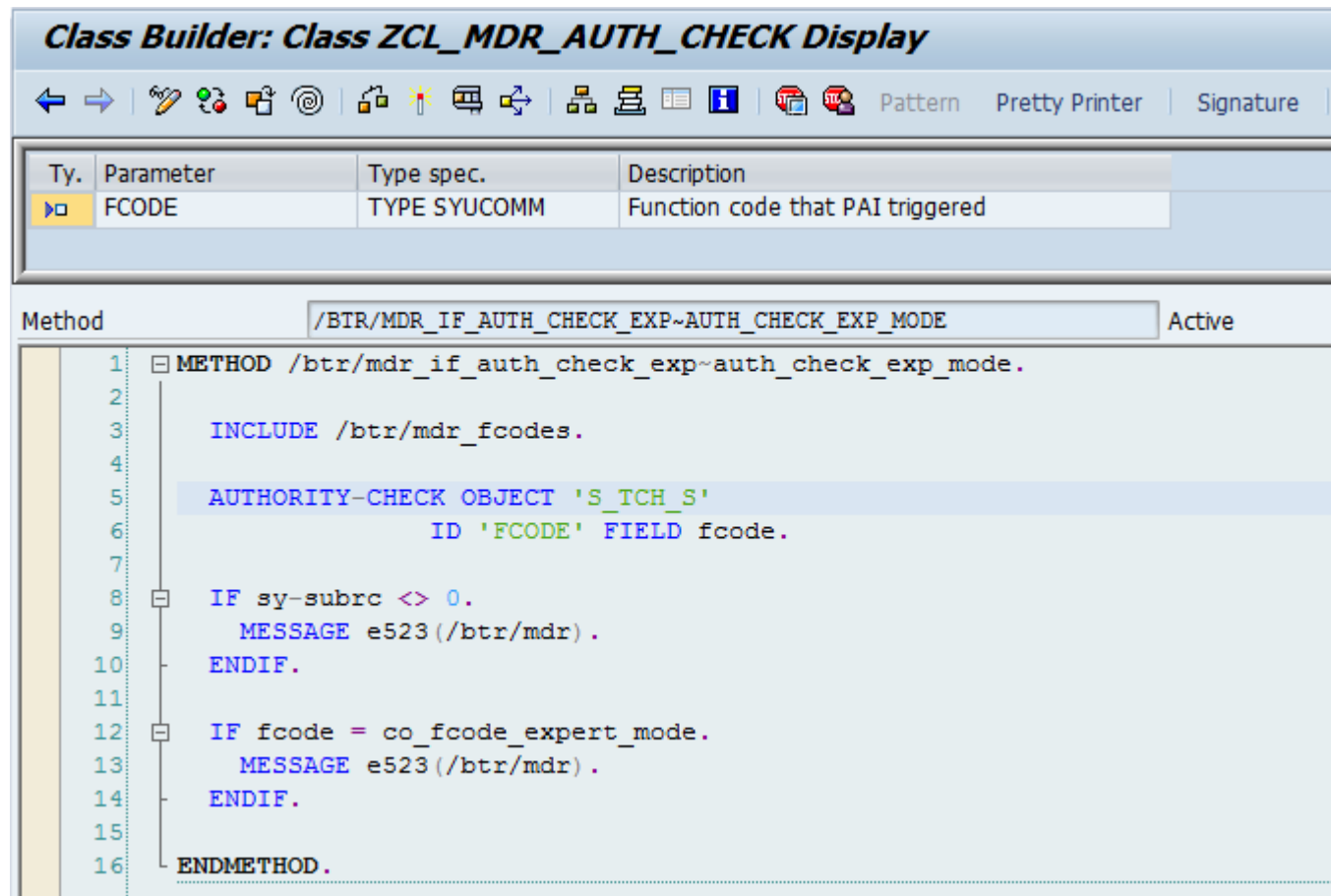
Message log level Other

Check

Expert Mode







Restricting access to Expert Mode on Run history transaction /BTR/MDRH can be done either by issue of error messages (Type E) or by hiding the Expert Mode button from the tool bar. Note: Both options can be done only at user level and not at program level.

To restrict access to Expert Mode using error messages (Type E) use BAdI implementing class method AUTH_CHECK_EXP_MODE. An example can be seen below:





Attempting to press the expert mode button results in an error message:

MDR: Run History

Transform | Application Log |      

Diffuser Program	Report title
/BTR/MDR_SAMPLE_FLIGHT_REPORT	Diffuser Sample: Flight Report

Instance Name	Started By	Start Date	Start Time	End Date	End Time
Demo	TENGLAND	17.01.2015	01:29:01	17.01.2015	01:29:14

 No authorization for the chosen action 

To restrict access to Expert Mode at user level by hiding the button on the Tool bar use BAdI implementing class method AUTH_CHECK_PF_STATUS. An example can be seen below:

Class Builder: Class ZCL_MDR_AUTH_CHECK Display

← → | | Pattern | Pretty Printer | Signature

Ty.	Parameter	Type spec.	Description
▶▶	VALUE(XYT_EXCLUDING)	TYPE /BTR/SLIS_EXTAB OPTIONAL	Table with excluded fcodes

Method: /BTR/MDR_IF_AUTH_CHECK_EXP~AUTH_CHECK_PF_STATUS Active








```

1  method /btr/mdr_if_auth_check_exp~auth_check_pf_status.
2
3      include /btr/mdr_fcodes.
4      clear xyt_excluding.
5
6      append co_fcode_show_variant      to xyt_excluding.
7      append co_fcode_show_intervals    to xyt_excluding.
8      append co_fcode_show_resultset    to xyt_excluding.
9      append co_fcode_action_stop       to xyt_excluding.
10     append co_fcode_action_resume      to xyt_excluding.
11     append co_fcode_action_adjust      to xyt_excluding.
12     append co_fcode_action_delete      to xyt_excluding.
13     append co_fcode_action_archive     to xyt_excluding.
14     append co_fcode_action_force_err   to xyt_excluding.
15     append co_fcode_view_jobs          to xyt_excluding.
16     append co_fcode_appserver          to xyt_excluding.
17     append co_fcode_expert_mode        to xyt_excluding.
18
19  endmethod.

```

Table XYT_EXCLUDING holds the function code values to be excluded from the Run History screen's pf-status. Make sure to exclude all function codes under Expert Mode as well as Expert Mode itself (see above). The button will then disappear as seen below:

MDR: Run History

 Transform |  Application Log |   |  |  | 

Diffuser Program	Report title
/BTR/MDR_SAMPLE_FLIGHT_REPORT	Diffuser Sample: Flight Report

Instance Name	Started By	Start Date	Start Time	End Date	End Time
Demo	TENGLAND	17.01.2015	01:29:01	17.01.2015	01:29:14

Individual Actions

BAdI definition /BTR/MDR_BADI_AUTH_CHECK_EXP offers the ability to restrict access to individual functions. For instance, actions like deleting an instance run, pausing or increasing number of batch jobs can be restricted at program level.

BAdI implementing class method AUTH_CHECK_FCODE. An example can be seen below:

Class Builder: Class ZCL_MDR_AUTH_CHECK Display

Navigation icons: Back, Forward, Find, Replace, Run, Stop, Breakpoint, Undo, Redo, Print, Signature, Pattern, Pretty Printer, Signature.

Ty.	Parameter	Type spec.	Description
▶	FCODE	TYPE SYUCOMM OPTIONAL	Function code that PAI triggered
▶	PROGRAMOID	TYPE /BTR/OID OPTIONAL	MDR: Internal Object ID
▶▶	RETURN_VALUE	TYPE SYSUBRC DEFAULT 0	Check Return Value

Method: /BTR/MDR_IF_AUTH_CHECK_EXP~AUTH_CHECK_FCODE Active

```

1  METHOD /btr/mdr_if_auth_check_exp~auth_check_fcode.
2
3      INCLUDE /btr/mdr_fcodes.
4
5      AUTHORITY-CHECK OBJECT 'S_TCH_S'
6                          ID 'PROID' FIELD programoid
7                          ID 'FCODE' FIELD fcode.
8
9      IF sy-subrc <> 0.
10         *      No authorization for the chosen action
11         MESSAGE e523 (/btr/mdr).
12     ENDIF.
13
14  ENDMETHOD.
  
```