# Table of Contents

**Audience** .................................................................................................................................................... 2

**Basic Concepts** ........................................................................................................................................ 3  
  Node5 Architecture ...................................................................................................................................... 4  
  Node5 Diffuser ........................................................................................................................................ 5  
  Node5 MiniCube ....................................................................................................................................... 6  
  Instance .................................................................................................................................................. 7  
  Intervals ................................................................................................................................................ 8  
  Interval Processing .................................................................................................................................. 10  
  Collation ............................................................................................................................................... 11  
  Transformation ...................................................................................................................................... 12

**Introduction** ............................................................................................................................................. 13

**When to use Z Accelerators** .................................................................................................................... 14

**Main Program** ........................................................................................................................................... 15  
  General ................................................................................................................................................ 16  
  Declaration .......................................................................................................................................... 17  
  Selection Screen Definition .................................................................................................................... 18  
  START-OF-SELECTION Event .................................................................................................................. 19  
  Interval Processing Subroutine ................................................................................................................ 20  
  Collating Interval Results .......................................................................................................................... 22  
  AT SELECTION-SCREEN OUTPUT ......................................................................................................... 24

**Transformation Program** .......................................................................................................................... 25  
  Regular Transformation Program .............................................................................................................. 26  
  Selection Screens and Transformation Programs ....................................................................................... 28

**Setting up a Diffuser program** ................................................................................................................... 29  
  Program Definition ................................................................................................................................ 30  
  Default Technical Settings ........................................................................................................................ 32  
  Interval Generation .................................................................................................................................. 34

**Running Diffuser Programs** ...................................................................................................................... 36  
  Technical Settings ................................................................................................................................... 37  
  Developer Notes ..................................................................................................................................... 41

**Administering Diffuser Programs** .............................................................................................................. 43  
  Diffuser Mode ....................................................................................................................................... 44
Audience

This guide has been developed for the following audience:

- ABAP™ Developers intending to use Z Accelerators with the Node5 Architecture
- Performance Specialists or Advisers
- SAP® Architects
Basic Concepts

Before starting development using Z Accelerators, a few basic concepts must be explained.

- Node5 Architecture
- Diffuser
- MiniCube
- Instance
- Intervals
- Interval Processing
- Collation
- Transformation
Node5 Architecture

The Node5 Architecture is the core of all of Basis Technologies software solutions.

For a program to be accelerated by the Node5 Diffuser, it can either be developed as a custom Z Accelerator or provided as a prepackaged program supplied by Basis Technologies (as a GT, GTi or BDi App). The key features to accelerate a program are the Node5 Diffuser and Node5 MiniCube.
Node5 Diffuser

The Node 5 Diffuser takes a large data set and splits it up into small pieces of data called **Intervals**; these can then be processed with multiple processors running. The Node 5 Diffuser allows the number of processors to be increased or decreased at runtime, including pausing and restarting a run. A group of processors can be assigned to a Capacity Group to allow processing power to be dynamically distributed across programs.

This can be utilised by custom programs as a Z Accelerator or by programs supplied prepackaged by Basis Technologies (the GT, GTi and BDi Apps).
Node5 MiniCube

When a Node5 Diffuser run completes it can store the results as a Node 5 MiniCube inside the Node5 Architecture, these can be retrieved and further selections applied to this data. The data can also be supplied in an interactive way so that for example ALV or drilldown features can be used.

The transaction /BTR/MINICUBE also allows the monitoring of runs and with the Diffuser Mode the instance resources can be increased at runtime, see Administering Diffuser Programs.
Instance

Each run of a program using the Node5 Diffuser 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 they wish to view the details of. The framework allows the saving of data against the instance.
Intervals

The Node5 Diffuser 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, might be Intervals of Business Partner Numbers, Sales Order Numbers, or Account Numbers. The Node5 Diffuser ensures that these Intervals of data are processed using multiple processors instead of the traditional sequential approach with one processor.

An interval can be thought of as a range of values that represent X number of master or transaction 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:

<table>
<thead>
<tr>
<th>Interval</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>1001</td>
<td>2000</td>
</tr>
<tr>
<td>3</td>
<td>2001</td>
<td>3000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>99001</td>
<td>100000</td>
</tr>
</tbody>
</table>

The concept of an Interval Object is used to create these Intervals for use by a Diffuser program this is where an object such as a Sales Orders are broken down before the Diffuser program is run. As part of the definition of a Diffuser 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 Diffuser are:

- Sales Order
- Business Partner
- Contract Account

The generation of Interval Objects is detailed in the Setting up a Diffuser program Interval Generation section.
When using Z Accelerators intervals can also be built at runtime. This is programmed into the Diffuser program and 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 Z Accelerators – Advanced Concepts – Dynamic Interval Generation section.
Interval Processing

For a Node5 Diffuser 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 Diffuser 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 a Diffuser program is executed, it is run with X jobs (where X is a number from 1 upwards). Node5 Diffuser achieves its performance gains by executing multiple processors at once, 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. Node5 Diffuser 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, this result is stored in the MiniCube.

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.
Introduction

Basis Technologies' Node5 Architecture provides a framework and run-time environment for custom developed SAP reports that must work with large volumes of a program running this we call a Z Accelerator. It provides for use of multiple processors with custom reports such that these programs can run within acceptable time constraints to deliver timelier business information.

Z Accelerators can be easily developed using the standard SAP ABAP Workbench. The Z Accelerators 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 Z Accelerator, there are three main components that need to be set up. This refers to the Node5 Architecture configuration (or Diffuser definition) and also two programs in the ABAP repository for each Z Accelerator, 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. Diffuser Definition – This defines the name of the technical components of each Z Accelerator, as well as a number of other settings relevant to using the Diffuser

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

Z Accelerators are 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 Z Accelerators may not be required. However, even if your report runs within 2 hours (and this is considered acceptable) you are still able to use Z Accelerators to bring this run-time down even further.

It is our consideration that almost any report that executes in the background can gain from being run using Z Accelerators. This is because the Z Accelerator 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.
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 Diffuser Program                                                       *
*---------------------------------------------------------------------------------*
REPORT Z_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 Diffuser program, you must define the select-options and parameters of a program within the following Diffuser statements:

```diffuser
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:

```diffuser
*---------------------------------------------------------------------*
* 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 Diffuser main report. However, unlike standard ABAP reports this event does nothing except execute an Diffuser statement. This statement must immediately follow the START-OF-SELECTION keyword and nothing else should follow it. In effect, this single Diffuser 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 Diffuser 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                                      *
*----------------------------------------------------------------*

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 bukrs = p_bukrs.

* 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.
  ENDDO.
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:

```plaintext
*---------------------------------------------------------------------*
* 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 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 a Diffuser program

For a program to use the Node5 Diffuser, it can either be developed as a custom Z Accelerator or provided as a prepackaged program supplied by Basis Technologies (as a GT, GTi or BDi App).

To setup a program to use Node5 Diffuser use the transaction /BTR/DIFFUSER here as a minimum define the Diffuser program in the Program Definition, then choose to setup the technical settings and generate intervals where required.

- Program Definition
- Default Technical Settings
- Interval Generation
The definition of an Node5 Diffuser program is set up via the transaction /n/BTR/DIFFUSER.

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.
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/ and DEFAULT respectively.

Furthermore, the transaction N/BTR/DIFFUSER 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 relevant subroutines.
Default Technical Settings

The second sub-object managed through the transaction /BTR/DIFFUSER is “Defaults for Technical Settings”. This screen contains two main sections.

1. “Defaults for Technical Settings” allows to set default values for a specific Diffuser program. Once set, these values will always appear on the Technical Settings’ sub screen for that program see Technical Settings under Running Diffuser Programs for more details.

1. “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 a Diffuser program to be called from another program without interrupting the latter with the MiniCube 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 Diffuser 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 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 Diffuser Mode in the MiniCube 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, see the section Authority Checks in the Z Accelerators Guide.

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 section Authority Checks in the Z Accelerators Guide.
Interval Generation

When a Node5 Diffuser program uses Interval Objects, an Interval Variant needs to be created from the Interval Object, before the Diffuser program is run. An Interval Variant can be thought of as the set of Intervals that the Diffuser 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 Diffuser Interval Objects. They have similar operation except for the generation of the Interval Variants. The Intervals (or Interval Variant) are created before the Diffuser 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 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.
### Diffuser: Generate Interval Variant

<table>
<thead>
<tr>
<th>Int.Object</th>
<th>Variant</th>
<th>Interval</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>1</td>
<td>1</td>
<td>000000514</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>2</td>
<td>000000514</td>
<td>000000978</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>3</td>
<td>000000978</td>
<td>00001442</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>4</td>
<td>00001442</td>
<td>00001906</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>5</td>
<td>00001906</td>
<td>00002370</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>6</td>
<td>00002370</td>
<td>00002834</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>7</td>
<td>00002834</td>
<td>00003298</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>8</td>
<td>00003298</td>
<td>00003762</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>9</td>
<td>00003762</td>
<td>00004226</td>
</tr>
<tr>
<td>SCUSTOM</td>
<td>INT:10</td>
<td>10</td>
<td>00004226</td>
<td>999999999</td>
</tr>
</tbody>
</table>
Running Diffuser Programs
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 Node5 Diffuser specific technical settings.
These “Technical Settings” are important when executing an Diffuser program be it in a production environment, or when performing unit testing of your Diffuser program.

Note that you can set up Default Technical Settings and set up user authorizations to control the users ability to change these settings, for more information refer to the Z Accelerators – 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 Diffuser 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 Diffuser 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 Diffuser programs and ensures the whole program runs sequentially, in a foreground process.

• Wait for run to complete before finishing – This is often used when running Diffuser 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.

• Launch Transformation Program after completed run – This means the MiniCube screen is skipped so the user gets straight to their results when the run completes

• Distribution List – After a Diffuser program completes 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. 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 Z Accelerators – 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.
The impact of the two fields is as below:

- **Interval Count** – This specifies the number of intervals (pieces) 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 Diffuser program and you specify the first run option, the execution will be in the background; Diffuser will immediately start the jobs and take you into the MiniCube 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/MINICUBE, or via transaction /BTR/DIFFUSER and the “MiniCube” 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.
### MDR: Run History

<table>
<thead>
<tr>
<th>MDR Program</th>
<th>Report title</th>
<th>Run Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>/STR/MDR_SAMPLE_FLIGHT_REPORT</td>
<td>MDR Sample! Flight Report</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Started By</th>
<th>Start Date</th>
<th>Start Time</th>
<th>End Date</th>
<th>End Time</th>
<th>Instance Status</th>
<th>Comp Remaining</th>
<th>Active Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>American Airlines Flights</td>
<td>TENGLAND</td>
<td>16.01.2018</td>
<td>19:52:56</td>
<td>00:10:00</td>
<td>In Process</td>
<td>48%</td>
<td>0:00:159</td>
<td>4 (9)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interval</th>
<th>Low</th>
<th>High</th>
<th>Status</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000000001</td>
<td>000000005</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>000000006</td>
<td>000000010</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>000000011</td>
<td>000000015</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>000000016</td>
<td>000000020</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>000000021</td>
<td>000000025</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>000000026</td>
<td>000000030</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>000000031</td>
<td>000000035</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>000000036</td>
<td>000000040</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>000000041</td>
<td>000000045</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>000000046</td>
<td>000000050</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>000000051</td>
<td>000000055</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>000000056</td>
<td>000000060</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>000000061</td>
<td>000000065</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>000000066</td>
<td>000000070</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>000000071</td>
<td>000000075</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>000000076</td>
<td>000000080</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>000000081</td>
<td>000000085</td>
<td>Completed</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>000000086</td>
<td>000000090</td>
<td>Completed</td>
<td>1</td>
</tr>
</tbody>
</table>
Administering Diffuser Programs

The Node5 Architecture provides the advanced user a number of powerful administrative capabilities via the MiniCube transaction /N/BTR/MINICUBE (see screen below). These capabilities provide a powerful way of managing your Diffuser programs.

- Diffuser Mode
- Intervals
- Results
- Variants
- App Servers
- Increase or Decrease Jobs
- Pause
- Resume
- Delete
- Force Error
- Reprocess Error
- Debug an Interval
- Rename Instance
Diffuser Mode

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

If authorized a number of other functions will be revealed.
Results

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

You can also select an interval and view the raw results stored against each interval, by double-clicking the number of results.
Intervals

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

The details of all the intervals are then displayed as below.
Variants

To access the details entered on the selection screen for an instance click the variant button as below.

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

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

This then displays the available App Servers.

<table>
<thead>
<tr>
<th>Server Name</th>
<th>Host Name</th>
<th>Message Types</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>bu101_D01_01</td>
<td>bu101</td>
<td>Dialog Batch Update Upd2 Spool Enqueue JCM</td>
<td>Active</td>
</tr>
</tbody>
</table>
Increase or Decrease Jobs

Through the MiniCube transaction, you can see historical instances of a program as well as any currently executing program instances. You can also see the number of active jobs for each program instance currently running, and it is possible to change the number of jobs running for a particular active job.

Adding more jobs can help decrease the run time.

To change the number of jobs click the number of jobs currently running and a popup appears where you can enter the new number of jobs you want the instance to run. Note the top of the popup box shows the number of unused background jobs in the system at that point in time, in this case 22.

Instance is now running 10 jobs.

Alternatively you can click the arrow buttons to increase or decrease the jobs one at a time.
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.
Pause

It is possible to Pause (or Stop) a program instance using this option. By selecting this option after selecting a program instance, the Diffuser tells the currently executing jobs to no-longer process any more intervals after it completes the processing of the current intervals. The status of the Instance, and unprocessed intervals changes to “Stopped”. You will need to click Refresh to update the status. This is a powerful option that is used typically when a Diffuser 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. When the instance is paused, the Diffuser 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.

The benefit Diffuser has over the traditional approach to executing reports is that the Diffuser 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 pause an Instance that is currently in the “In Progress” status.

To pause a program simply select the instance and hit the pause button, you will be asked to confirm that you want to.

Once all the intervals have completed the status of the instance changes as below.
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 “Stopped” interval to “Available”. The restart option can only be selected for instances with the status “Paused” or “Error”.

To resume a program select the instance and press the resume button, a pop appears for the number of jobs required to start processing intervals.
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.

To delete an instance select it right-click and select the Delete option as below.
**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.

To delete an instance select it right mouse click and select the force error option as below.
Reprocess Error

On finding an interval in error as below there is an option to reprocess where you have been able to fix the cause of the error, such as updating some master data.

Bear in mind the impact that running the interval out of sequence or at a later date may have on your report or processing of data.

To reprocess the error select the instance in the status of error and right-click for the "Reprocess Error" option as below.
The same as resuming a Diffuser instance the popup for the number of processors you want to utilize appears.

In this example the error is successfully reprocessed.
### ZERROR_FLIGHT_REPORT

<table>
<thead>
<tr>
<th>Status</th>
<th>Instance Name</th>
<th>Started By</th>
<th>Start Date</th>
<th>Start Time</th>
<th>End Date</th>
<th>End Time</th>
<th>Status</th>
<th>Comp Remaining</th>
<th>Active Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>☢️</td>
<td>ZERROR_FLIGHT_REPORT</td>
<td>TONGLAND</td>
<td>12.06.2015</td>
<td>17:33:02</td>
<td>12.06.20</td>
<td>17:56:50</td>
<td>100%</td>
<td></td>
<td>- + CG</td>
</tr>
</tbody>
</table>

#### ZERROR_FLIGHT_REPORT

<table>
<thead>
<tr>
<th>Status</th>
<th>Inter</th>
<th>Low Value</th>
<th>High</th>
<th>Status</th>
<th>Runetime</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>☢️</td>
<td>1</td>
<td>000000001</td>
<td>00000047</td>
<td>Completed</td>
<td>0:00:09</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>2</td>
<td>00000048</td>
<td>00000098</td>
<td>Completed</td>
<td>0:00:10</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>3</td>
<td>00000099</td>
<td>0000173</td>
<td>Completed</td>
<td>0:00:13</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>4</td>
<td>0000174</td>
<td>0000221</td>
<td>Completed</td>
<td>0:00:04</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>5</td>
<td>0000223</td>
<td>0000273</td>
<td>Completed</td>
<td>0:00:03</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>6</td>
<td>00000274</td>
<td>0000332</td>
<td>Completed</td>
<td>0:00:06</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>7</td>
<td>00000333</td>
<td>0000379</td>
<td>Completed</td>
<td>0:00:04</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>8</td>
<td>00000368</td>
<td>0000426</td>
<td>Completed</td>
<td>0:00:04</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>9</td>
<td>00000427</td>
<td>0000473</td>
<td>Completed</td>
<td>0:00:04</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>10</td>
<td>00000474</td>
<td>0000520</td>
<td>Completed</td>
<td>0:00:03</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>11</td>
<td>00000521</td>
<td>0000567</td>
<td>Completed</td>
<td>0:00:03</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>12</td>
<td>00000568</td>
<td>0000614</td>
<td>Completed</td>
<td>0:00:03</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>13</td>
<td>00000615</td>
<td>0000661</td>
<td>Completed</td>
<td>0:00:04</td>
<td></td>
</tr>
<tr>
<td>☢️</td>
<td>14</td>
<td>00000662</td>
<td>0000706</td>
<td>Completed</td>
<td>0:00:03</td>
<td></td>
</tr>
</tbody>
</table>
Rename Instance

To rename an instance select the instance and right-click.

Enter the new name.

The new name is updated as below:
Debug an Interval

On finding an interval in error you also have the option of debugging the interval to try and work out what went wrong.

Firstly ensure you have positioned your break point in the code, then select the interval and right-click for the option to “Debug an Interval”

The debugger will then open at your break point.
* Type-cast the interval low and high values
  lv_customid_low  = x_interval-low.
  lv_customid_high = x_interval-high.

  if x_interval-sequence  = 5.
    message  e000(/BTR/MDR).
  endif.

* Write a message to the MDR application log
**Scheduling Diffuser Programs**

A Diffuser 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.

Node5 Diffuser 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 Diffuser 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.
  ENDF;
  ...
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 a Diffuser program there is a parameter “Distribution List”. Diffuser 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, Diffuser provides the ability to send email notification of results of reports not migrated to Diffuser format. Diffuser 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 Diffuser 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-Diffuser 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:
   a. Internal SAP Username
   b. External Address
   c. Organizational Unit (As defined in the HR Org Structure)
   d. 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 Programs

One of the major benefits provided 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 first executing a program with Diffuser at the top of the selection screen is the “Technical Settings” button, 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.

In addition on finding an interval in error you also have the option of debugging the interval to try and work out what went wrong see “Debug an Interval” for details.
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

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.

```abap
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_l 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 customid 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.
    ENDF.
ELSE.
  lv_interval_size = x_input-interval_size.
ENDF.

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_l = lv_interval_index + 1.

CLEAR ls_sbook.
READ TABLE lt_sbook INTO ls_sbook INDEX lv_interval_index_l.
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 AUTHCHECK_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).

Technical Settings will look like this:
When the changing parameter LCK_TECH_SETT is set to true all input fields of the Technical Settings screen is grayed out (see below).
Technical Settings will look like this:
### 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
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:

![Class Builder: Class ZCL_MDR_AUTH_CHECK Display](image)

```plaintext
METHOD /btr/mdr_if_auth_check_exp-auth_check_exp_mode.

INCLUDE /btr/mdr_fcodes.

AUTHORITY-CHECK OBJECT 'S_TCH_S'
   ID 'FCODE' FIELD fcode.

IF sy-subrc <> 0.
   MESSAGE e523(/btr/mdr).
ENDIF.

IF fcode = co_fcode_expert_mode.
   MESSAGE e523(/btr/mdr).
ENDIF.

ENDMETHOD.
```

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

<table>
<thead>
<tr>
<th>Diffuser Program</th>
<th>Report title</th>
</tr>
</thead>
<tbody>
<tr>
<td>/BTR/MDR_SAMPLE_FLIGHT_REPORT</td>
<td>Diffuser Sample: Flight Report</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Started By</th>
<th>Start Date</th>
<th>Start Time</th>
<th>End Date</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo</td>
<td>TENGLAND</td>
<td>17.01.2015</td>
<td>01:29:01</td>
<td>17.01.2015</td>
<td>01:29:14</td>
</tr>
</tbody>
</table>
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:

```plaintext
<table>
<thead>
<tr>
<th>Method</th>
<th>/BTR/MDR_IF_AUTH_CHECK_EXP~AUTH_CHECK_FCODE</th>
</tr>
</thead>
</table>

1) INCLUDE /btr/mdr_if_auth_check_exp-auth_check_fcode.

2) AUTHORITY-CHECK OBJECT 'S_TCH_S'
   ID 'PROID' FIELD prooid
   ID 'FCODE' FIELD fcode.

3) IF sy-subrc <> 0.
4)   * No authorization for the chosen action
5)     MESSAGE e523('/btr/mdr).
6)     ENDF.
7)     ENDMETHOD.
```
Software Support
Online Forum

Basis Technologies have an online forum containing over 250 searchable Frequently Asked Questions relating to our products.

These FAQs cover many of the common error / warning messages that can be experienced during normal usage and also useful HOW TO guides to perform many of the common operations.

The online forum can be accessed via the following URL:

http://support.basistechnologies.com/forums

You will need to register for a username and password before you can access the forum.
Support from Basis Technologies

Raising Support Tickets

To request support from Basis Technologies on any issue relating to our product sets (Transport Expresso, DevOps, Diffuser or Utilities), a ticket should be raised via the following email address:

support@basistechnologies.com

Sending an email to this address will automatically create a ticket in Zendesk, the ticketing tool used by Basis Technologies.

Please include as much information as possible about the issue (product, version, error messages, steps to replicate, screenshot attachments) in the email. In addition, please also include your own contact details in your email.

Please reflect any high priority issues by including URGENT or HIGH PRIORITY at the start of the email subject.

Support Escalation

If you have any concerns with the service you are getting from Basis Technologies support, or wish to escalate any high priority issues please email supportescalation@basistechnologies.com

Require additional Information or Services?

If additional information or services relating to any of Basis Technologies product sets is required, you can contact us via the above support@basistechnologies.com address, or alternatively by contacting your assigned Basis Technologies Account Director.