BATCH DATA COMMUNICATIONS/BATCH INPUT/INBOUND-OUTBOUND

TABLE OF CONTENTS

ABOUT DATA TRANSFER

Implementing a new software system takes major effort. **New implementation requires moving data from the present system i.e., legacy system into the R3 system.** The product, components, customers and vendors have to be available in the new system. Initial data transfer is the process of populating your R3 database with data from your legacy system.

To prepare for the data transfer there are certain tasks you need to perform.

- First, understand your SAP system to know which data needs to be transferred, e.g., you would not transfer any sales order if you do not use the Sales and distribution module.
- Second, you need to know the contents of existing data in your legacy system.

Data transfer program, an effective and efficient way of transferring large amount of data into your new system, saves time and resources. **But more importantly it ensures that accurate data is transferred into R/3.**

Two steps involved in data transfer are CONVERSION and SAP DATA TRANSFER.

- **CONVERSION,** data is converted from your legacy system into the required flat file format.
- **SAP DATA TRANSFER,** data is automatically entered into the SAP system. A SAP data transfer program reads the prepared data from the flat file and moves it into R/3.

Steps for any data transfer

Preparation for legacy database

This is the first step of transfer though not associated with SAP, plays very important role in data transfer.

Before data is extracted, delete obsolete data in the legacy system and fix inconsistencies. It is easier this way, than doing it during conversion.

The two steps involved in this are:

Data purging: Before transferring data from legacy system, delete all the old and obsolete data, e.g.: To save conversion time and disk space, you may delete all one-time customers, vendors and all unused materials.
Data cleansing: This process corrects data inconsistencies and ensures the integrity of the existing data during the transfer process. Mistakes must be fixed before the transfer.

Converting legacy data to the flat file

For this second step, SAP provides no specific tools.

- In the ABAP/4 development workbench, write an ABAP/4 program to convert a file from your legacy system into the required flat file structure.
- Use other programming language to write conversion program. For C, COBOL, you can easily download the table definition for specific flat file structure.
- Use third party tools, such as format editors and code generators, which support mapping and conversion between different file formats.
- For other RDBMS like oracle, Sybase, MS access, you have EXPORT utility. By which, you can directly export data to flat file.

Files will be discussed in detail in later part of the topic.

Getting the data into R/3

This step actually transfers data to SAP database. After converting the data into the flat file you are ready to begin the third step of the data transfer.

Data transfer is an interactive process. You may often feel like you are taking two steps forward and one step back.

For example, short steps involved in whole process are as follows:

- Convert the data from the legacy system into the flat file format.
- Run the data transfer program.
- Check data for error.
- Is the transfer working as it was designed?
- If not, adjust the data/conversion program and start with step 1. Go back to step one, if you don't get the desired result.

You have three different options to enter your data into R/3.

- Automatically, with SAP standard data transfer programs.
- Automatically, by creating your own branch input programs
- Manually, by entering the data via the corresponding online transaction.

Automatic transfer with a standard data transfer program

This can be done if:

- ♦ A standard program exists for the data transfer of a business object in R/3.
- ♦ The data is available in electronic form.
- ♦ There is a significant number of records you want to transfer.
- ♦ The cost of converting the legacy data into the required flat file format is acceptable.

Manually transferring business objects

You should manually transfer data, if:

- ♦ You have no legacy system.
- ♦ There is only small number of records to enter.
- ♦ Translating legacy data into the R/3 structure is more an effort than manually entering the data.

Using customer specific batch input to transfer business objects

Create batch input program to transfer data if:

- ♦ No standard program exists to transfer the business object in R/3.
- ♦ The data is available in electronic form.
- ♦ There is a significant number of records you want to transfer.
- ♦ Translating your legacy data into the structure required by your custom program is easier than manually entering data.

Batch input is a standard procedure for transferring large amount of data into the R/3 system. It simulates manual data entry. Data consistency is ensured because batch input uses all the checks conducted on the normal screen. Using batch input is like entering the data online. Another advantage to batch input is that you do not have to check the data in advance.

Batch input is a two-step procedure. It involves a program that creates the batch input session. This session is the data file that includes everything to begin the transaction and the data to be entered on the appropriate screens. The data is not yet in the database tables of R/3 application. The second step is to process the session, which then actually transfers the data to database table. You can transfer data directly to database table by using CALL TRANSACTION method also. Another method - Direct Input, is done for high volume of data for the standard application. All these methods are discussed in detail in later part of the topic.

Basically there are two steps involved in any transfer of data from legacy system to SAP system.

- Creation of file and transferring file into SAP system
- Transferring data to database file

Whenever, you create flat file following points should be considered:

- Provide the data in an ASCII/Text file format.
- Know how each line of the file is structured.
- Know how the required flat file for the business object must be structured.
- Once your flat file is ready, the data should be transferred into SAP system.

FILE HANDLING IN SAP

Introduction

- ♦ Files on application server are **sequential files.**
- ♦ Files on presentation server / workstation are **local files.**
- ♦ A sequential file is also called a dataset.

Handling of Sequential file

Three steps are involved in sequential file handling

- ♦ OPEN
- ♦ PROCESS
- ♦ CLOSE

Here processing of file can be READING a file or WRITING on to a file.

OPEN FILE

Before data can be processed, a file needs to be opened.
After processing file is closed.

Syntax:

```
OPEN DATASET <file name> FOR {OUTPUT/INPUT/APPENDING}
IN  {TEXT/BINARY} MODE
```

This statement returns SY_SUBRC as 0 for successful opening of file or 8, if unsuccessful.

OUTPUT: Opens the file for writing. If the dataset already exists, this will place the cursor at the start of the dataset, the old contents get deleted at the end of the program or when the CLOSE DATASET is encountered.

INPUT: Opens a file for READ and places the cursor at the beginning of the file.

FOR APPENDING: Opens the file for writing and places the cursor at the end of file. If the file does not exist, it is generated.

BINARY MODE: The READ or TRANSFER will be character wise. Each time 'n" characters are READ or transferred. The next READ or TRANSFER will start from the next character position and not on the next line.

IN TEXT MODE: The READ or TRANSFER will start at the beginning of a new line each time. If for READ, the destination is shorter than the source, it gets truncated. If destination is longer, then it is padded with spaces.

Defaults: If nothing is mentioned, then defaults are FOR INPUT and in BINARY MODE.

PROCESS FILE:

Processing a file involves READing the file or Writing on to file TRANSFER.

TRANSFER Statement
Syntax:

```
TRANSFER <field> TO <file name>.
<Field> can also be a field string / work area / DDIC structure.
```

Each transfer statement writes a statement to the dataset. In binary mode, it writes the length of the field to the dataset. In text mode, it writes one line to the dataset.

If the file is not already open, **TRANSFER tries to OPEN file FOR OUTPUT (IN BINARY MODE) or using the last OPEN DATASET statement for this file.**

IF FILE HANDLING, **TRANSFER** IS THE ONLY STATEMENT WHICH DOES NOT RETURN SY-SUBRC

READ Statement
Syntax:

```
READ DATASET <file name> INTO <field>.
<Field> can also be a field string / work area / DDIC structure.
```

Each READ will get one record from the dataset. In binary mode it reads the length of the field and in text mode it reads each line.

CLOSE FILE:
The program will close all sequential files, which are open at the end of the program. However, it is a good programming practice to explicitly close all the datasets that were opened.
Syntax:

```
CLOSE DATASET <file name>.
```

SY-SUBRC will be set to 0 or 8 depending on whether the CLOSE is successful or not.

DELETE FILE:
A dataset can be deleted.

Syntax:

```
DELETE DATASET <file name>.
```

SY-SUBRC will be set to 0 or 8 depending on whether the DELETE is successful or not.

Pseudo logic for processing the sequential files:

For reading:

```
Open dataset for input in a particular mode.
        Start DO loop.
          Read dataset into a field.
          If READ is not successful.
             Exit the loop.
          Endif.
Do relevant processing for that record.
       End the do loop.
Close the dataset.
```

For writing:

```
Open dataset for output / Appending in a particular mode.
Populate the field that is to be transferred.
TRANSFER the filed to a dataset.
Close the dataset.
```

Handling of local files

Introduction

Files on presentation server / workstation are LOCAL FILES.

Local files are processed using UPLOAD and DOWNLOAD functions. The local files are brought into ABAP/4 memory using these functions. Unlike dataset, all the records of the file are UPLOADED into an internal table in one shot. Similarly, all records are DOWNLOADED in one shot from an internal table to a local file.

DOWNLOAD function:

Important EXPORTING parameters for this function are:

Filename = name of the local file to which the internal table is to be downloaded.
Filetype = file type, default values are ASC, DAT, BIN
Mode = Write mode, overwrite (' ') or append ('A')

Important IMPORTING parameters are:

Filename = actual file name entered

Tables to be passed to the function:

Data_tab = the internal table that is to be downloaded.

Similar function called WS_DOWNLOAD is used to download the information from internal table to local file. The only difference between DOWNLOAD and WS_DOWNLOAD is that, DOWNLOAD does not require the 'FILENAME' and 'FILETYPE' to be exported to the function; instead it will ask for the same at runtime. However, for WS_DOWNLOAD, these two parameters need to be passed.

UPLOAD function

Upload function is used to upload the local file to internal table into SAP system.
Parameters passed are similar to DOWNLOAD function.

For uploading, you have similar function called WS_UPLOAD.

Files with multiple record types

Many times the input files will have records with different structures. In all such cases each record will have a field, (usually the first) which identifies the record (e.g., 'H' for header, 'D' for detail or '1' & '2' etc).

Text tile with multiple record types would be as mentioned below:

```
Hxxxxyyyyyyyynnnnnnnnn
Daaaaaaaaaabbbbbbbbbcccccccc
Daaaaaaaaaabbbbbbbbbbcccccccc
Hxxxxyyyyyyyynnnnnnnnnnn
Daaaaaaaaaabbbbbbbbbbcccccccc
Daaaaaaaaaabbbbbbbbbbcccccccc
```

Processing Text file with multiple record types:

To process such files, it is necessary to first read the record into a character field that is a minimum of the lengths of the different structure in the file. If 'H' type record is 30 char long (including the record identifier) and 'D' type is 40 long (including the record identifier), then this character field should be at least 40 char long.

Pseudo logic for processing the sequential files with multiple record types (text mode):

```
Open dataset.
Do.,
 Read dataset into character field.
 If sy-subrc ne 0.
  Exit.
 Endif.
If record id of the char field (or the first of field) is 'H'
Do processing for 'H' type of records.
Else.
Do processing for d type of records.
```

```
 Endif.
Enddo.
```

Pseudo logic for processing the local files with multiple record types:

```
WS_UPLOAD into itab.
Loop at itab.
        If itab-recid is 'H'.
           Do processing for 'H' type record.
        Else.
          Do processing for 'D' type record.
        Endif.
Endloop.
```

BATCH DATA COMMUNICATION

About Data Transfer In R/3 System

When a company decides to implement the SAP R/3 to manage business-critical data, it usually does not start from a no-data situation. Normally, a SAP R/3 project comes into replace or complement existing application.

In the process of replacing current applications and transferring application data, two situations might occur:

- ♦ The first is when application data to be replaced is transferred at once, and only once.
- ♦ The second situation is to transfer data periodically from external systems to SAP and vice versa.
- ♦ There is a period of time when information has to be transferred from existing application, to SAP R/3, and often this process will be repetitive.

The SAP system offers two primary methods for transferring data into SAP systems. From non-SAP systems or legacy system. These two methods are collectively called "batch input" or "batch data communication".

1. SESSION METHOD
2. CALL TRANSACTION
3. DIRECT INPUT

Advantages offered by BATCH INPUT method:

1. Can process large data volumes in batch.
2. Can be planned and submitted in the background.
3. No manual interaction is required when data is transferred.
4. Data integrity is maintained as whatever data is transferred to the table is through transaction. Hence batch input data is submitted to all the checks and validations.

To implement one of the supported data transfers, you must often write the program that exports the data from your non-SAP system. This program, known as a "***data transfer***" program must map the data from the external system into the data structure required by the SAP batch input program.

The batch input program must build all of the input to execute the SAP transaction.

Two main steps are required:

- ♦ To build an internal table containing every screen and every field to be filled in during the execution of an SAP transaction.
- ♦ To pass the table to SAP for processing.

Prerequisite for Data Transfer Program

Writing a Data Transfer Program involves following prerequisites:

Analyzing data from local file

Analyzing transaction

Analyzing transaction involves following steps:

- ♦ The transaction code, if you do not already know it.
- ♦ Which fields require input i.e., mandatory.
- ♦ Which fields can you allow to default to standard values.
- ♦ The names, types, and lengths of the fields that are used by a transaction.
- ♦ Screen number and Name of module pool program behind a particular transaction.

To analyze a transaction::

- ♦ Start the transaction by menu or by entering the transaction code in the command box.
  (You can determine the transaction name by choosing System – Status.)
- ♦ Step through the transaction, entering the data will be required for processing your batch input data.
- ♦ On each screen, note the program name and screen (dynpro) number.
  (dynpro = dyn + pro. Dyn = screen, pro = number)
- ♦ Display these by choosing System – Status. The relevant fields are Program (dynpro) and Dynpro number. If pop-up windows occur during execution, you can get the program name and screen number by pressing F1 on any field or button on the screen.
  The technical info pop-up shows not only the field information but also the program and screen.
- ♦ For each field, check box, and radio button on each screen, press F1 (help) and then choose Technical Info.

Note the following information:

- The field name for batch input, which you'll find in its own box.
- The length and data type of the field. You can display this information by double clicking on the Data Element field.

- Find out the identification code for each function (button or menu) that you must execute to process the batch-input data (or to go to new screen).

  Place the cursor on the button or menu entry while holding down the left mouse button. Then press F1.
  In the pop-up window that follows, choose Technical info and note the code that is shown in the Function field.
  You can also run any function that is assigned to a function key by way of the function key number. To display the list of available function keys, click on the right mouse button. Note the key number that is assigned to the functions you want to run.

  Once you have program name, screen number, field name (screen field name), you can start writing.
  DATA TRANSFER program.

Declaring internal table

First Integral Table similar to structure like local file.

Declaring internal table like BDCDATA

The data from internal table is not transferred directly to database table, it has to go through transaction. You need to pass data to particular screen and to particular screen-field. Data is passed to transaction in particular format, hence there is a need for batch input structure.

The batch input structure stores the data that is to be entered into SAP system and the actions that are necessary to process the data. The batch input structure is used by all of the batch input methods. You can use the same structure for all types of batch input, regardless of whether you are creating a session in the batch input queue or using CALL TRANSACTION.

This structure is BDCDATA, which can contain the batch input data for only a single run of a transaction. The typical processing loop in a program is as follows:

- Create a BDCDATA structure
- Write the structure out to a session or process it with CALL TRANSACTION USING; and then
- Create a BDCDATA structure for the next transaction that is to be processed.

Within a BDCDATA structure, organize the data of screens in a transaction. Each screen that is processed in the course of a transaction must be identified with a BDCDATA record. This record uses the Program, Dynpro, and Dynbegin fields of the structure.

The screen identifier record is followed by a separate BDCDATA record for each value, to be entered into a field. These records use the FNAM and FVAL fields of the BDCDATA structure. Values to be entered in a field can be any of the following:

- ♦ Data that is entered into screen fields.
- ♦ Function codes that are entered into the command field. Such function codes execute functions in a transaction, such as Save or Enter.

The **BDCDATA** structure contains the following fields:

- ♦ **PROGRAM:** Name of module pool program associated with the screen. Set this field only for the first record for the screen.
- ♦ **DYNPRO:** Screen Number. Set this field only in the first record for the screen.
- ♦ **DYNBEGIN:** Indicates the first record for the screen. Set this field to X, only for the first record for the screen. (Reset to ‘ ‘ (blank) for all other records.)
- ♦ **FNAM:** Field Name. The FNAM field is not case-sensitive.
- ♦ **FVAL:** Value for the field named in FNAM. The FVAL field is case-sensitive. Values assigned to this field are always padded on the right, if they are less than 132 characters. Values must be in character format.

Transferring data from local file to internal table

Data is uploaded to internal table by UPLOAD of WS_UPLOAD function.

Population of BDCDATA

For each record of internal table, you need to populate Internal table, which is similar to BDCDATA structure.

All these five initial steps are necessary for any type of BDC interface.

DATA TRANSFER program can call SESSION METHOD or CALL TRANSACTION. The initial steps for both the methods are same.

First step for both the methods is to upload the data to internal table. From Internal Table, the data is transferred to database table by two ways i.e., Session method and Call transaction.

SESSION METHOD

About Session method

In this method you transfer data from internal table to database table through sessions.

In this method, an ABAP/4 program reads the external data that is to be entered in the SAP System and stores the data in session. A session stores the actions that are required to enter your data using normal SAP transaction i.e., Data is transferred to session which in turn transfers data to database table.

Session is intermediate step between internal table and database table. Data along with its action is stored in session i.e., data for screen fields, to which screen it is passed, the program name behind it, and how the next screen is processed.

When the program has finished generating the session, you can run the session to execute the SAP transactions in it. You can either explicitly start and monitor a session or have the session run in the background processing system.

Unless session is processed, the data is not transferred to database table.

BDC_OPEN_GROUP

You create the session through program by BDC_OPEN_GROUP function.

Parameters to this function are:

♦ User Name:	User name
♦ Group:	Name of the session
♦ Lock Date:	The date on which you want to process the session.
♦ Keep:	This parameter is passed as 'X' when you want to retain session after processing it or ' ' to delete it after processing.

BDC_INSERT

This function creates the session & data is transferred to Session.
Parameters to this function are:

♦ Tcode:	Transaction Name
♦ Dynprotab:	BDC Data

BDC_CLOSE_GROUP

This function closes the BDC Group. No Parameters.

Some additional information for session processing

When the session is generated using the KEEP option within the BDC_OPEN_GROUP, the system always keeps the sessions in the queue, whether it has been processed successfully or not.

However, if the session is processed, you have to delete it manually. When session processing is completed successfully while KEEP option was not set, it will be removed automatically from the session queue. Log is not removed for that session.

If the batch-input session is terminated with errors, then it appears in the list of INCORRECT session and it can be processed again. To correct incorrect session, you can analyze the session. The Analysis function allows to determine which screen and value has produced the error. If you find small errors in data, you can correct them interactively, otherwise you need to modify batch input program, which has generated the session or many times even the data file.

CALL TRANSACTION

About CALL TRANSACTION

A technique similar to SESSION method, while batch input is a two-step procedure, Call Transaction does both steps online, one after the other. In this method, you call a transaction from your program by

```
Call transaction  <tcode> using <BDCTAB>
                  Mode <A/N/E>
                  Update <S/A>
                  Messages into <MSGTAB>.
```

Parameter – 1	is transaction code.
Parameter – 2	is name of BDCTAB table.
Parameter – 3	here you are specifying mode in which you execute transaction **A** is all screen mode. All the screen of transaction are displayed. **N** is no screen mode. No screen is displayed when you execute the transaction. **E** is error screen. Only those screens are displayed wherein you have error record.
Parameter – 4	here you are specifying update type by which database table is updated. **S** is for Synchronous update in which if you change data of one table then all the related Tables gets updated. And sy-subrc is returned i.e., sy-subrc is returned for once and all. **A** is for Asynchronous update. When you change data of one table, the sy-subrc is returned. And then updating of other affected tables takes place. So if system fails to update other tables, still sy-subrc returned is 0 (i.e., when first table gets updated).
Parameter – 5	when you update database table, operation is either successful or unsuccessful or operation is successful with some warning. These messages are stored in internal table, which you specify along with MESSAGE statement. This internal table should be declared like **BDCMSGCOLL**, a structure available in ABAP/4. It contains the following fields:

1. **Tcode:** Transaction code
2. **Dyname:** Batch point module name
3. **Dynumb:** Batch input Dyn number
4. **Msgtyp:** Batch input message type (A/E/W/I/S)
5. **Msgspra:** Batch input Lang, id of message
6. **Msgid:** Message id
7. **MsgvN:** Message variables (N = 1 - 4)

For each entry, which is updated in database, table message is available in BDCMSGCOLL. As BDCMSGCOLL is structure, you need to declare a internal table which can contain multiple records (unlike structure).

Steps for CALL TRANSACTION method

1.	Internal table for the data (structure similar to your local file)
2.	BDCTAB like BDCDATA
3.	UPLOAD or WS_UPLOAD function to upload the data from local file to itab. (Considering file is local file)
4.	Loop at itab. Populate BDCTAB table. Call transaction <tcode> using <BDCTAB> Mode <A/N/E> Update <S/A>. Refresh BDCTAB. Endloop. (To populate BDCTAB, You need to transfer each and every field)

The major differences between Session method and Call transaction are as follows:

	SESSION METHOD	**CALL TRANSACTION**
1.	**Data is not updated in database table unless Session is processed.**	**Immediate updation in database table.**
2.	**No sy-subrc is returned.**	**Sy-subrc is returned.**
3.	**Error log is created for error records.**	**Errors need to be handled explicitly**
4.	**Updation in database table is always synchronous**	**Updation in database table can be synchronous Or Asynchronous.**

Error Handling in CALL TRANSACTION

When Session Method updates the records in database table, error records are stored in the log file. In Call transaction there is no such log file available and error record is lost unless handled. Usually you need to give report of all the error records i.e., records which are not inserted or updated in the database table. This can be done by the following method:

Steps for the error handling in CALL TRANSACTION

1.	Internal table for the data (structure similar to your local file)
2.	BDCTAB like BDCDATA
3.	Internal table BDCMSG like BDCMSGCOLL
4.	Internal table similar to Ist internal table (Third and fourth steps are for error handling)
5.	UPLOAD or WS_UPLOAD function to upload the data from the local file to itab. (Considering file is local file)

6.	Loop at itab.
	Populate BDCTAB table.
	Call transaction <tr.code> using <Bdctab>
	        Mode <A/N/E>
	        Update <S/A>
	        Messages <BDCMSG>.
	  Perform check.
	  Refresh BDCTAB.
	  Endloop.

7	Form check. IF sy-subrc <> 0. (Call transaction returns the sy-subrc if updating is not successful). Call function Format_message. (This function is called to store the message given by system and to display it along with record) Append itab2. Display the record and message.

DIRECT INPUT

About Direct Input

In contrast to batch input, this technique does not create sessions, but stores the data directly. It does not simulate the online transaction. To enter the data into the corresponding database tables directly, the system calls a number of function modules that execute any necessary checks. In case of errors, the direct input technique provides a restart mechanism. However, to be able to activate the restart mechanism, direct input programs must be executed in the background only. Direct input checks the data thoroughly and then updates the database directly.

You can start a Direct Input program in two ways;

Start the program directly

This is the quickest way to see if the program works with your flat file. This option is possible with all direct input programs. If the program ends abnormally, you will not have any logs telling you what has or has not been posted. To minimize the chance of this happening, always use the check file option for the first run with your flat file. This allows you to detect format errors before transfer.

Starting the program via the DI administration transaction

This transaction restarts the processing, if the data transfer program aborts. Since DI document are immediately posted into the SAP D/B, the restart option prevents the duplicate document posting that occurs during a program restart (i.e., without adjusting your flat file).

Direct input is usually done for standard data like material master, FI accounting document, SD sales order and Classification for which SAP has provided standard programs.

First time you work with the Direct Input administration program, you will need to do some preparation before you can transfer data:

- Create variant
- Define job
- Start job
- Restart job

Common batch input errors

- The batch input BDCDATA structure tries to assign values to fields which do not exist in the current transaction screen.

- The screen in the BDCDATA structure does not match the right sequence, or an intermediate screen is missing.
- On exceptional occasions, the logic flow of batch input session does not exactly match that of manual online processing. Testing the sessions online can discover by this.
- The BDCDATA structure contains fields, which are longer than the actual definition.
- Authorization problems.

RECORDING A BATCH INPUT

A B recording allows you to record a R/3 transaction and generate a program that contains all screens and field information in the required BDC-DATA format.

You can either use SHDB transaction for recording or

SYSTEM → SERVICES → BATCH INPUT → EDIT
And from here click recording.

Enter name for the recording.
(Dates are optional)
Click recording.
Enter transaction code.
Enter.
Click Save button.

You finally come to a screen where, you have all the information for each screen including BDC_OKCODE.

- Click Get Transaction.
- Return to BI.
- Click overview.
- Position the cursor on the just recorded entry and click generate program.
- Enter program name.
- Click enter

The program is generated for the particular transaction.

BACKGROUND PROCESSING

Need for Background processing

When a large volume of data is involved, usually all batch inputs are done in background.

The R/3 system includes functions that allow users to work non-interactively or offline. The background processing systems handle these functions.

Non-interactively means that instead of executing the ABAP/4 programs and waiting for an answer, user can submit those programs for execution at a more convenient planned time.

There are several reasons to submit programs for background execution.

- The maximum time allowed for online execution should not exceed 300 seconds. User gets TIMEOUT error and an aborted transaction, if time for execution exceeds 300 seconds. To avoid these types of error, you can submit jobs for background processing.
- You can use the system while your program is executing.

This does not mean that interactive or online work is not useful. Both type of processing have their own purposes. Online work is the most common one entering business data, displaying information, printing small reports, managing the system and so on. Background jobs are mainly used for the following tasks; to process large amount of data, to execute periodic jobs without human intervention, to run program at a more convenient, planned time other than during normal working hours i.e., Nights or weekends.

The transaction for background processing is **SM36.**
Or
Tools → Administration → Jobs → Define jobs
Or
System → services → Jobs

Components of the background jobs

A job in Background processing is a series of steps that can be scheduled and step is a program for background processing.

- Job name. Define the name of assigned to the job. It identifies the job. You can specify up to 32 characters for the name.
- Job class. Indicates the type of background processing priority assigned to the job. The job class determines the priority of a job. The background system admits three types of job classes: A B & C, which correspond to job priority.
- Job steps. Parameters to be passed for this screen are as follows:
  - Program name.
  - Variant if it is report program
  - Start criteria for the job: Option available for this are as follows:
    - Immediate - allows you to start a job immediately.
    - Date/Time - allows you to start a job at a specific name.
    - After job - you can start a job after a particular job.
    - After event - allows you to start a job after a particular event.
    - At operation mode - allows you to start a job when the system switches to a particular operation mode.

Defining Background jobs

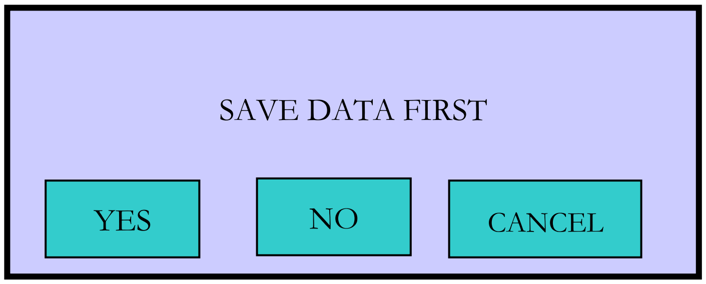
It is two step process: Firstly, you define the job and then release it.

When users define a job and save it, they are actually **scheduling** the report i.e., specifying the job components, the steps, the start time.

When users schedule program for background processing, they are instructing the system to execute an ABAP/4 report or an external program in the background. Scheduled jobs are not executed until they are released. When jobs are released, they are sent for execution to the background processing system at the specified start time. Both scheduling and releasing of jobs require authorizations.

HANDLING OF POP UP SCREEN IN BDC

Many times in transaction pop up screen appears and for this screen you don't pass any record but some indication to system telling it to proceed further. For example: The following screen



To handle such screen, system has provided a variable called BDC_CURSOR. You pass this variable to BDCDATA and process the screen.

Usually such screen appears in many transactions, in this case you are just passing information, that YES you want to save the information, that means YES should be clicked. So you are transferring this information to BDCDATA i.e., field name of YES which is usually SPOT_OPTION. Instead of BDC_OKCODE, you are passing BDC_CURSOR.

BDC_CURSOR is also used to place cursor on particular field.

AN EXAMPLE WITH SESSION METHOD

Following program demonstrates how data is passed from flat file to SAP transaction and further to database table by using SESSION method.

The transaction is TFBA (to change customer).

A simple transaction where you are entering customer number on first screen and on next screen data is displayed for the particular customer number. Field, which we are changing here, are name and city. When you click on save, the changed record gets saved.

Prerequisite to write this BDC interface as indicated earlier is:
1. To find screen number
2. To find screen field names, type of the field and length of the field.
3. To find BDC_OKCODE for each screen
4. Create flat file.

Flat file can be created in your hard disk as follows:

1	Vinod Krishna	Hyderabad
2	Kavitha	Secunderabad
3	Kishore	Hyderabad

(Where 1st character field is Customer number, 2nd field is Customer name and 3rd field is City.)

To transfer this data to database table SCUSTOM following interface can be used.

```
REPORT DEMO1.
* Following internal table is to upload flat file.
DATA: BEGIN OF ITAB OCCURS 0,
        ID(10),
        NAME(25),
        CITY(25),
      END OF ITAB.
*Following internal table BDCDATA is to pass date from internal table to session.
DATA: BDCTAB LIKE BDCDATA OCCURS 0 WITH HEADER LINE.
* Variables
DATA: DATE1 LIKE SY-DATUM. DATE1 = SY-DATUM - 1. " This is for Hold Date
* To upload flat file to internal table.
CALL FUNCTION UPLOAD
      EXPORTING
            FILE NAME                    = 'C:\FF.TXT'
            FILE TYPE                    = 'ASC"
      TABLES
```

```
            DATA_TAB                                    = ITAB
      EXCEPTIONS
            CONVERSION_ERROR                            = 1
            INVALID_TABLE_WIDTH                         = 2
            INVALID_TYPE                                = 3
            NO_BATCH                                    = 4
            UNKNOWN_ERROR                               = 5
            OTHERS                                      = 6.
If sy-subrc = 0.
* Calling Function to Create a Session
CALL FUNCTION 'BDC_OPEN_GROUP'
      EXPORTING
            CLIENT                                      = SY-MANDT
            GROUP                                       = 'POTHURI'
            HOLDDATE                                    = DATE1
            KEEP                                        = 'X'
            USER                                        = SY-UNAME
      EXCEPTIONS
            CLIENT_INVALID                              = 1
            DESTINATION_INVALID                         = 2
            GROUP_INVALID                               = 3
            GROUP_IS_LOCKED                             = 4
            HOLDDATE_INVALID                            = 5
            INTERNAL_ERROR                              = 6
            QUEUE_ERROR                                 = 7
            RUNNING                                     = 8
            SYSTEM_LOCK_ERROR                           = 9
            USER_INVALID                                = 10
            OTHERS                                      = 11.
If sy-subrc = 0.
*------------------------- MAIN Logic-----------------------------
LOOP AT ITAB
 PERFORM GENERATE_DATA. " Populating BDCDATA Table
 CALL FUNCTION 'BDC_INSERT'
      EXPORTING
            TCODE                           = 'TFBA'
      TABLES
            DYNPROTAB                       = BDCTAB
      EXCEPTIONS
            INTERNAL_ERROR                  = 1
            NOT_OPEN                        = 2
            QUEUE_ERROR                     = 3
            TCODE_INVALID                   = 4
            PRINTING_INVALID                = 5
            POSTING_INVALID                 = 6
            OTHERS                          = 7.
 REFRESH BDCTAB
ENDLOOP.
```

*** Calling function to close the session**

```
CALL FUNCTION ‘BDC_CLOSE_GROUP’
          EXCEPTIONS
                    NOT_OPEN          = 1
                    QUEUE_ERROR       = 2
                    OTHERS            = 3.
Endif.
Endif.
*&---------------------------------------------------------------------*
*& Form GENERATE_DATA
*&---------------------------------------------------------------------*
*    Create BDC Data
*&---------------------------------------------------------------------*
FORM GENERATE_DATA
* Passing information for 1st screen on BDCDATA
 BDCTAB-PROGRAM = ‘SAPMTFBA’.
 BDCTAX-DYNPRO = 100.
 BDCTAP-DYNBEGIN = ‘X’.
 APPEND BCDTAB.CLEAR BDCTAB.
* Passing field information to BDCDATA
 BDCTAB-FNAM = ‘SCUSTOM-ID’
 BDCTAB-FVAL = ITAB-ID.
 APPEND BDCTAB.CLEAR BDCTAB.
* Passing BDC_OKCODE to BDCDATA
 BDCTAB-FNAM = ‘BDC_OKCODE’.
 BDCTAB-FVAL = ‘/5’.
 APPEND BDCTAB.CLEAR BDCTAB.
* Passing screen information for next screen to BDCDATA
 BDCTAB-PROGRAM = ‘SAPMTFBA’.
 BDCTAB-DYNPRO = 200.
 BDCTAB-DYNBEGIN = ‘X’.
 APPEND BDCTAB.CLEAR BDCTAB.
* Passing screen information to BDCDATA
 BDCTAB-FNAM = ‘SCUSTOM-NAME’.
 BDCTAB-FVAL = ITAB-NAME.
 APPEND BDCTAB.CLEAR BDCTAB.
 * Passing screen information to BDCDATA
 BDCTAB-FNAM = ‘SCUSTOM-CITY’.
 BDCTAB-FVAL = ITAB-CITY.
 APPEND BDCTAB.CLEAR BDCTAB.
* Passing BDC_OKCODE to BDCDATA
 BDCTAB-FNAM = ‘BDC_OKCODE’.
 BDCTAB-FVAL = ‘SAVE’.
 APPEND BDCTAB.CLEAR BDCTAB.
ENDFORM.                        “GENERATE_DATA
```

AN EXAMPLE WITH CALL TRANSACTION

Same steps to be repeated for CALL TRANSACTION

The only difference between the two types of interface is in Session method, you create session and store information about screen and data into session. When session is processed the data is transferred to database. While in CALL TRANSACTION, data is transferred directly to database table.

```
REPORT DEMO1.
* Follow above Code till MAIN Logic. Even the Subroutine should be copied
LOOP AT ITAB
 PERFORM GENERATE_DATA, "Populating BDCDATA Table
 Call transaction 'TFBA' using BCDDATA Mode 'A' Update 'S'.
 REFRESH BDCTAB
ENDLOOP.
```