SEQUENTIAL FILES & BDC

You can process sequential files with the statements READ DATASET (for read operations) and TRANSFER (for write operations).

The sequential file must be visible to the application server. Files on the current presentation server cannot be processed with TRANSFER or READ DATASET

Sequential files are the medium for importing data from a customer system to the R/3 System and for exporting data from the R/3 System to a customer system.

Before data records can be written to a or read from a sequential file, the file must be opened.

After processing, the file is closed.

To open a file, you use the statement ***OPEN DATASET <file name> FOR <options> IN <TEXT / BINARY MODE.>***

FOR OPTIONS :

FOR OUTPUT

Opens the file for writing. If the file already exists, its contents are deleted, but this applies only after the end of the program. If the statement occurs in a program, the cursor is simply positioned at the beginning of the file. You must then specify **CLOSE DATASET** in order to delete the file. If the file does not exist, it is generated.

FOR INPUT

Opens an existing file for reading. If the file is already open, the cursor is positioned at the beginning of the file. You do not have to specify FOR INPUT explicitly.

FOR APPENDING

Opens the file for writing at the end of the file. If the file does not exist, it is generated. If the file is already open, the cursor is positioned at the end.

If you do not specify any further options, the file is opened for reading

IN OPTIONS :

IN BINARY MODE

The data is not interpreted by the read and write operations READ DATASET and TRANSFER. The data areas specified for these key words are input or output directly. You do not have to specify IN BINARY MODE explicitly.

IN TEXT MODE

If you open a file with this option, the system assumes that the file has a line structure. Each time READ DATASET or TRANSFER is executed, one line is always input or output and the data is always processed up to the end-of-line selection. If the data area is too large for the line that is read, the remaining area is padded with blanks. If it is too small, the end of the line is lost.

Blanks at the end of a data record are not suppressed in text mode

You can transfer the records into a sequential file using TRANSFER command

Each TRANSFER statement transfers one data record to a sequential file.

Before TRANSFER, you place the data record in a field or a structure.

Possible structures are field strings or table work areas.

The execution of the TRANSFER statement depends on the mode:

- Binary mode: Writes the length of the field or structure.
- Text mode: Writes a line.

You can read the data from a sequential file into a Table or structure using the READ DATASET statement.

Each READ DATASET statement reads one record from a sequential file into a field or structure. Possible structures are field strings or table work areas.

The execution of the READ DATASET statement depends on the mode:

- Binary mode: Reads the length of the structure.
- Text mode: Reads a line.

READ DATASET, like TRANSFER, does not perform conversions implicitly. The data is read in as it was written

You use the CLOSE DATASET <file name> statement to close a sequential file explicitly.

In a program, all opened files are implicitly closed each time the screen changes, and then reopened exactly as they were before when processing resumes.

At the end of a program, all files are closed.

Opening and closing files explicitly aids readability

You can display the current state of the file in debugging by choosing *Goto -> System -> System areas* and then entering (or selecting) DATASETS.

STEPS IN CREATING A SEQUENTIAL FILE

First you define the structure to be transferred. In this example, it is a field string, but it could also be a help field, the header liner of an internal table or an ABAP/4 Dictionary work area.

The field string can also be transferred from an include report.

Next, you fill the structure. In principle, all statements that perform a value assignment can be used here. Examples include MOVE, COMPUTE, MOVE-CORRESPONDING, MOVE TO and WRITE TO.

Finally, you transfer the structure with the TRANSFER statement.

STEPS IN READING A SEQUENTIAL FILE :

First you define the structure to be read. In this example, it is a field string. It can also be transferred from an include report.

The structure used for reading with READ DATASET must match the structure used for writing with TRANSFER.

You read the data records into your structure within a loop and process them.

READ DATASET does not require a logical database. You can therefore analyze your extract data along with data from any logical database (which you then specify in the report attributes).

In a file processing program, you first define the structures required for the data records with the TABLES or DATA statement. The program uses these structures as internal buffers for the data records.

You then open the sequential file for reading or writing with the OPEN statement.

When writing to the file, you transfer the filled structures to the file with TRANSFER statements.

When reading the file, you fill the structures set up for the data records with READ DATASET for further processing in the program.

To close the sequential file, you exit file processing with the CLOSE DATASET statement.

Introduction to Batch Input Processing

The reasons for transferring data without a basic interactive dialog with the user are:

- The datasets are large and already exist on an electronic data medium
- The data is transferred to another system and only then imported into the SAP system.

External Data cannot be transffered to SAP system directly, it has to be checked before it is transferred to the SAP Database

- Large data sets are transferred to the SAP system using **batch input**.
- Batch input is an automatic procedure. It is often referred to as **BDC (Batch Data Communication)**.
- To guarantee data consistency, the **transfer data** is subject to the same checks and updates as dialog data which is entered interactively by the user. Examples include:

  - Format checks
  - Automatic value range check (against check table or fixed value range)
  - Conversion of user data to program data and vice-versa
  - Field default values

The transfer data is stored temporarily in the form of a **batch input session** in a queue file.

- Batch input or BDC (Batch Data Communication) is an automatic procedure for transferring data to the SAP system without a user dialog.
- A user dialog is simulated for this procedure so that the same checks and updates can be performed.
- A central component of this procedure is a queue file.
- This file receives the data via batch input programs and groups associated data together into ‘sessions’.
- To load the data into SAP databases, you process the sessions with the batch input function (menu options *Systems -> Services-> Batch input*, Transaction SM35).
- Data is transferred to the queue file by batch input programs.
- These programs perform the following functions:
  - They provide structured work areas in the form of an internal table (BDC table) for the data to be transferred.
  - They read in the data.
  - They place the data in the BDC table.

- They transfer the filled BDC table to the queue file.

The BDC table receives the input for the data transfer transactions

- You process these sessions with the batch input function (choose *Systems -> Services-> Batch input* or call Transaction SM35). As when you execute application functions (with add or change transactions) in the dialog interface, this sends data to the log file. When you subsequently perform an update, the data reaches the relevant SAP databases.
- The batch input function starts the application functions specified in the relevant session (indicated by their transaction codes).
- The data from the session is now copied to the screens belonging to the online transaction specified.

Function Module BDC_OPEN_GROUP

- You open a batch input session with the function module BDC_OPEN_GROUP.
- The parameters you must specify are as follows:

CLIENT	(Client)
GROUP	(Session name)
USER	(User name)

- Specification of the following parameters is optional:

HOLDDATE	(Earliest session start date)
KEEP	(ID whether session should be deleted after successful processing;
	KEEP = ' ': Delete session (default value)
	KEEP = 'X': Do not delete session)

USER must always be specified; The user type should be BDC.

Function Module BDC_INSERT

- You use function module BDC_INSERT to insert the data for **ONE** transaction into a session.
- To transfer the data, you require an internal table (BDC table).
- You must specify the following parameters:

TCODE	(transaction code),
DYNPROTAB	(BDC table).

BDC Table

- The BDC table holds the data for one transaction.
- The contents of the BDC table are then passed to the queue file.
- One table line includes the following information:
  Program name, screen number, start ID, field name, field value.

You declare the BDC table as an internal table with the ABAP/4 Dictionary structure 'BDCDATA'.

- The BDC table is set up line by line. As for every internal table, you do this with MOVE and APPEND statements.
- The fields of the table header line should be reset to their initial value with CLEAR.

Function Module BDC_CLOSE_GROUP

You close a batch input session with the function module BDC_CLOSE_GROUP

Preparations for BDC Program

- First perform (step by step) the application function to be simulated.
- Make a note of:

  - the program name and screen number for each screen
  - the table field name for each field you are filling.
  - the number for each function key pressed
  - the screen program sequence

- To do this, you use the system function *Status* and the *Technical info* help function.

- Before you write the batch input program, you have to run the transaction and note down the screen numbers, screen sequence and the screen field names to be addressed later in the program. You enter this data in the BDC table.

The field name for the OK code is the same for all screens: BDC_OKCODE. The value for the OK code consists of a slash '/' and the function key number (example: '/11').

In the declaration part of the BDC program, you define the BDC table

- In order to create sessions you need the function modules which open and close sessions.
- To insert data into the session, construct the BDC table within a subroutine and call the function module BDC_INSERT.

Functionality of Transaction SM35

TC SM35 offers the following features :

Overview: You can have an overview of Batch Input Sessions, Batch Input Process logs, Locked Sessions
Analysis: You can display session contents, display batch input data, logs, statistics
Administration: You can select sessions for deletion, lock sessions, release sessions
Process: You do the process the session in foreground, display errors only and also process the sessions in the background

- When processing a session, the update mode is always synchronous (i.e. the next transaction is called only when the update for the preceding transaction has been completed).

Report RSBDCSUB

- You use the report RSBDCSUB to schedule session processing.
- Proceed as follows:

1. Create a variant. The selection criteria are:

   - Session name (generic entries are also possible)
   - Creation date
   - Session status
   - Target host (for the background system)
   - Extended log

2. Create a background job for the report RSBDCSUB with the variant you have defined.

- The authorization object S_BDC_MONI is used for authorization checks in batch input processing.
- You can use this authorization object to restrict processing to particular sessions (specifying the session name, or make a generic entry). You can also protect specific activities in batch input processing (by making an entry in the ‘Activities’ field).

Authorizations for Batch Input Processing

Object	Fields	Values	Meaning
S_BDC_MONI	Session name	<session names>	Specifies session names for which a user is authorized
	Activities	ABTC AONL	Process (submit) in background Process online
		DELE LOCK	Delete Sessions/logs Lock date, Change
		FREE	Release terminated sessions
		ANAL	Analyze sessions

Standard Utility Reports for Batch Input

Standard Utility Reports for Batch Input

- **RSBDCREO**
  - deletes all sessions which are flagged as successful and still in the system, together with their logs
  - physically deletes all logs for which there are no sessions
  - reorganizes the log file (file is reduced if logs have been deleted)
  - function integrated in SM35
- **RSBDCLOG**
  - generates a list of batch input logs, selected by session name
  - can display or delete logs and, if sessions exist, activate the analysis
  - function integrated in SM35
- **RSBDCDRU**
  - allows you to print out the contents of selected sessions
  - function integrated in SM35

- When the batch input procedure is used to import data from a customer system, the batch input program generally transfers the data from a sequential file.
- You create the sequential file with a customer transfer program. You then use a batch input program (the SAP transfer program) to transfer the data from the sequential file to the SAP queue file.

The SAP System offers standard batch input programs for many standard data transfer cases. This means you do not have to write you own batch input program. However, the data you provide must be in a predefined SAP format

- There is a data analysis at the start of every data transfer.
- In the data analysis, you decide which data can be transferred and how. Proceed as follows:
  - Compare your old data with the data fields in the SAP System.
  - Define which fields from the old data can be transferred directly to the SAP System.
  - Draw up rules for filling data fields in the SAP System where old data cannot be directly used.
- You require the SAP structure descriptions for the transfer program. If you write the transfer program in ABAP/4, you use the TABLES statement to transfer the structure descriptions directly from the ABAP/4 Dictionary.

- You generate the structure descriptions in the respective programming language using the ABAP/4 program RDDSRCG0. These structure descriptions are then available in a sequential file and you can include them in your transfer program.

- If batch input data is not to be assigned to particular fields in the SAP System, the standard batch input programs expect a special character (NODATA flag). The default special character is '/', but you can define your own character with the field BGR00-NODATA. You must then initialize the SAP batch input structures with this special character.
- If the fields with the old structures are of a different length or type to the fields with the SAP structures, the transfer program must perform the conversions. The fields with the SAP batch input structures are always type C.

- As an alternative to **batch input,** there are application-specific programs that use the **direct input** technique. When transferring large volumes of data (more than 10000 transactions), you can achieve considerable improvements in performance with direct input. Like **CALL TRANSACTION USING**, direct input is a means of updating data immediately (no sessions are generated). However, unlike CALL TRANSACTION USING or batch input, no screens are involved. Instead, the data is imported by calling function modules which check the data and then transfer it directly to the database tables. Since

direct input offers a restart mechanism in the case of an error, the programs can only run as background jobs. You manage and start these jobs with the report RBMVSHOW or Transaction BMV0.

Examples:

RFBIBL00 **FI** Can be switched between direct input, batch input and CALL TRANSACTION. Posts documents.

RMDATIND **MM** Direct input. Imports material master records.

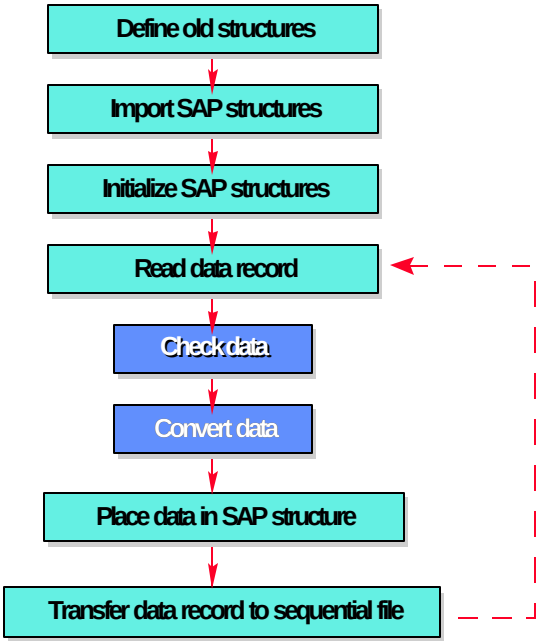
RVAFSS00 **SD** Direct input. Generates SD billing documents and FI follow-on documents from an external file.

- As an alternative to **batch input,** there are application-specific programs that use the **direct input** technique. When transferring large volumes of data (more than 10000 transactions), you can achieve considerable improvements in performance with direct input. Like **CALL TRANSACTION USING**, direct input is a means of updating data immediately (no sessions are generated). However, unlike CALL TRANSACTION USING or batch input, no screens are involved. Instead, the data is imported by calling function modules, which check the data and then transfer it directly to the database tables. Since direct input offers a restart mechanism in the case of an error, the programs can only run as background jobs. You manage and start these jobs with the report RBMVSHOW or Transaction BMV0.

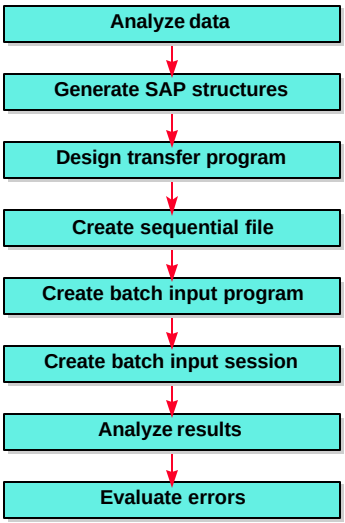
Examples:

RFBIBL00	**FI**	Can be switched between direct input, batch input and CALL TRANSACTION. Posts documents.
RMDATIND	**MM**	Direct input. Imports material master records.
RVAFSS00	**SD**	Direct input. Generates SD billing documents and FI follow-on documents from an external file.

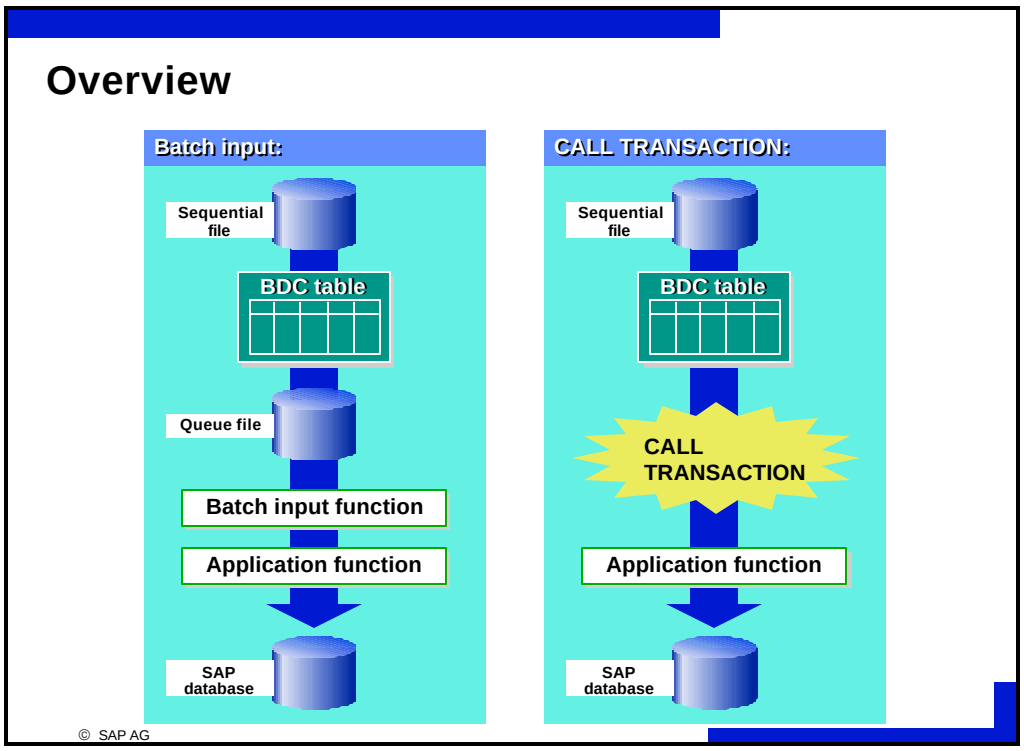
Tasks of a Transfer Program



Data Transfer with Batch Input



CALL TRANSACTION



- In contrast to batch input, the CALL TRANSACTION statement allows you to pass data directly to the dialog interface (without using the queue file). To store screen data temporarily, you use an internal table (a BDC table which has the same structure as with batch input). Then, you call the desired transaction in your program and the system copies the data temporarily stored the BDC table to the screens of the transaction.

The CALL TRANSACTION Statement

```
CALL TRANSACTION        <transaction code>
        USING           <BDC table>
        MODE            <display mode>
        UPDATE          <update mode>
        MESSAGES INTO   <messtab>
```

<display mode>:

A	Display all
E	Display only if there are errors
N	Display nothing

<update mode>:

S	Do not continue processing until update has finished (synchronous)
A	Continue processing immediately

In contrast to batch input, there is no error logging. Whether the called transaction is processed depends on the authorizations of the calling user. The BDC table can only accept data for a single transaction. This means that before the next transaction call, ***you must perform a REFRESH on the BDC table and refill it with data.***

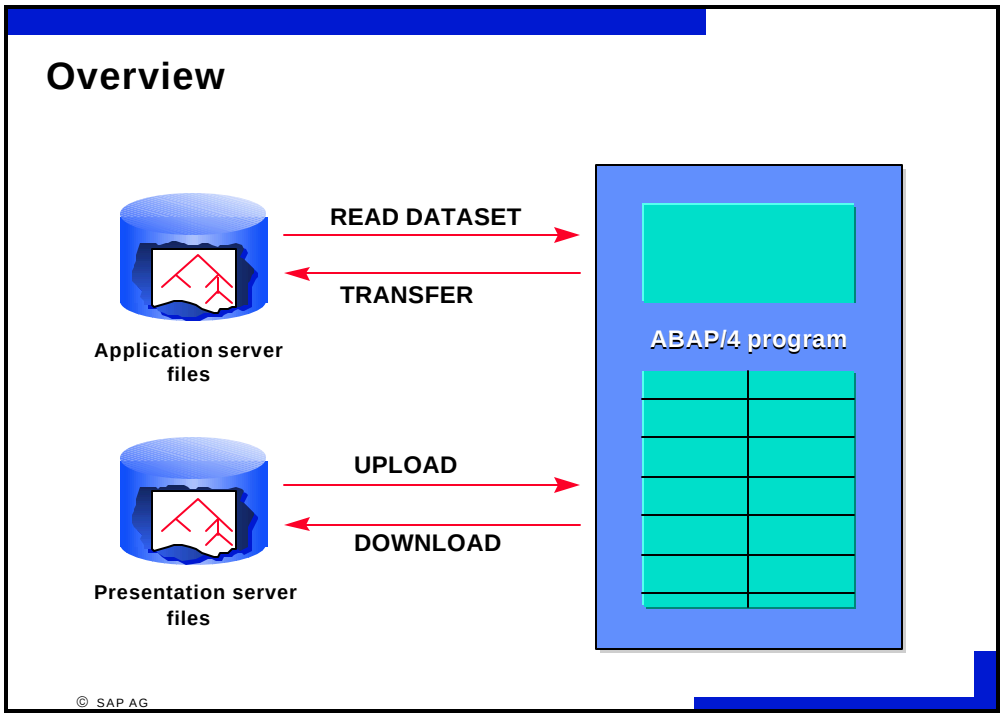
Batch Input / CALL TRANSACTION - Summary

	Batch input	CALL TRANSACTION
Return code	No	Yes
Error logging	Yes	No
Processing	Time-delayed	Immediately

LOCAL FILES

ABAP/4 programs run on the application server and have access to its sequential files.

Overview - Sequential Files



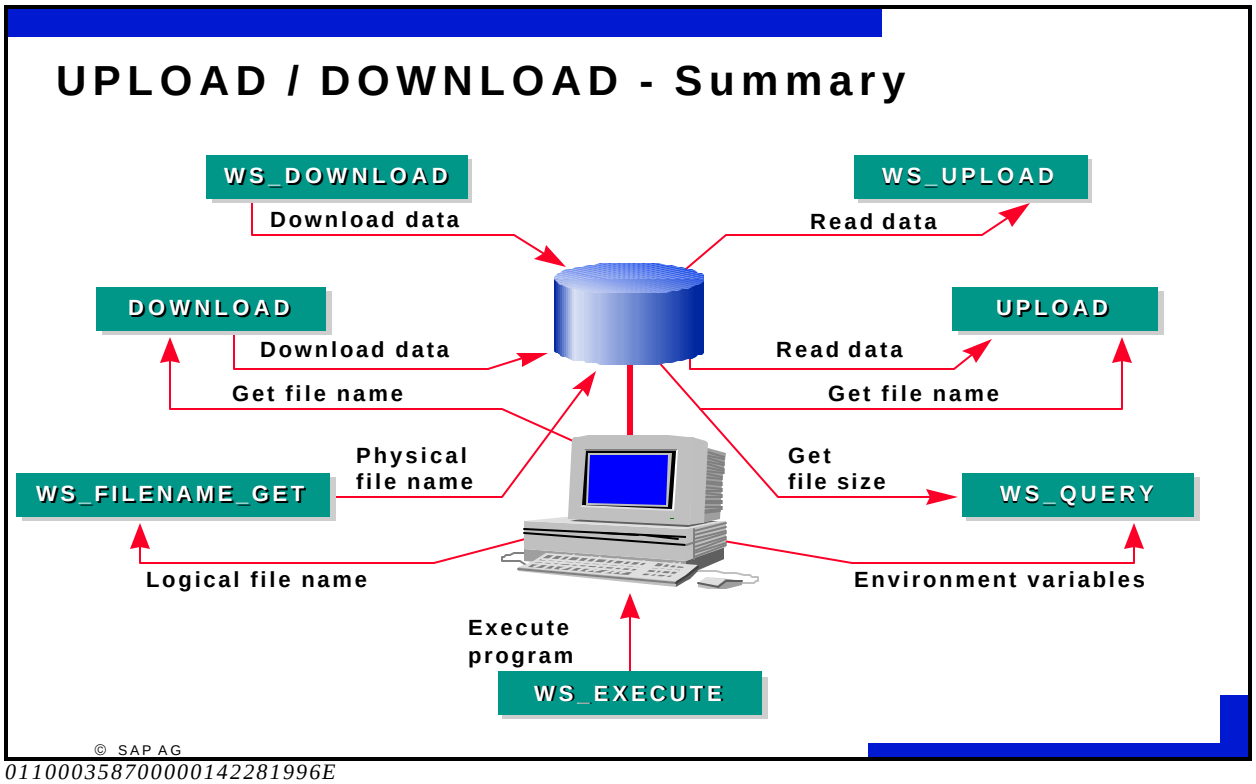
- You process sequential files on the application server in ABAP/4 programs with the READ DATASET and TRANSFER statements which allow you to copy data from/to a field string.
- You process local sequential files on the presentation server with (UPLOAD and DOWNLOAD) function modules. These copy data from/to an internal table.

For the local file, you specify the absolute file name (meaning the complete path of the location of the file.

- For the function module DOWNLOAD, you need an internal table for the data transfer. You define this table according to your data structure at the beginning of the program and then fill it with data.
- You use the MODE parameter to define the write mode ('A' to extend a file, ' ' to create a new file).

- With UPLOAD and DOWNLOAD, another format available for conversions apart from ASC (ASCII) and BIN (binary) is DAT for Excel. With DOWNLOAD, WK1 is also available for Excel and Lotus.
- For the function module UPLOAD, you need an internal table for the data transfer. You define this table according to your data structure at the beginning of the program.
- The specification of default values for the file name, file type and a header for the file dialog is optional.

UPLOAD / DOWNLOAD - Summary



011000358700000142281996E

- The function modules UPLOAD and DOWNLOAD are a convenient way of processing local files because they perform all the necessary activities automatically. The user is requested to specify the file name and type interactively.
- There are also other function modules you can use for file processing. These include the following:

- WS_DOWNLOAD and WS_UPLOAD:

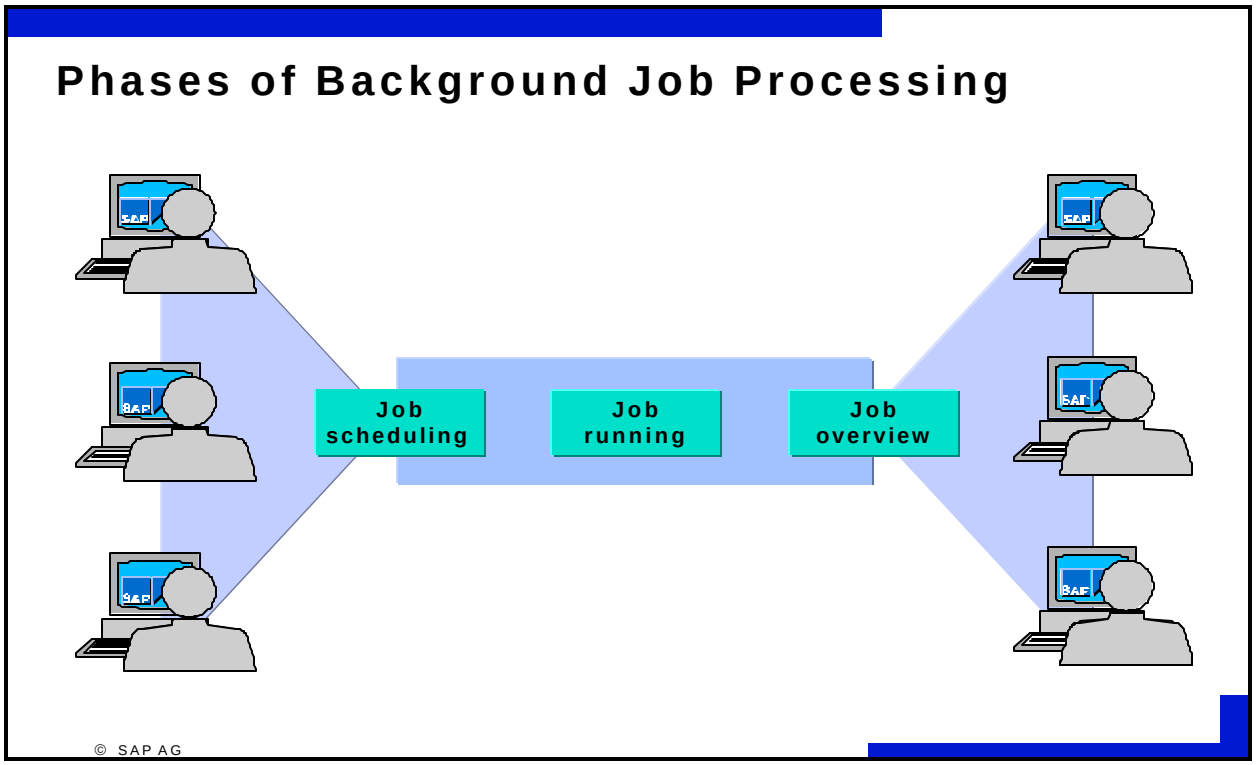
  Like DOWNLOAD, except that the file name and type are not specified interactively and exceptions must be handled by the calling program.

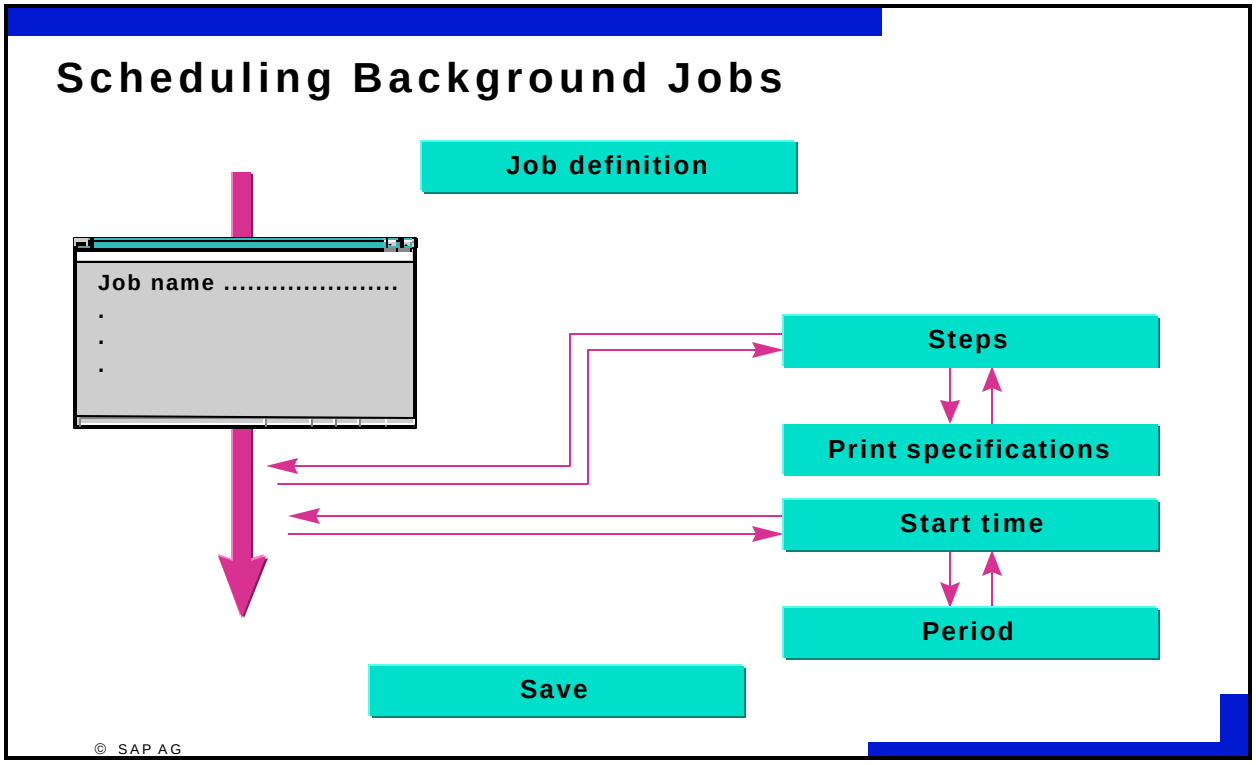
- WS_QUERY:

  checks the existence and size of a file, as well as other environment variables.

- WS_EXECUTE:

  starts a program on the presentation server.





- You create a job with the *Job definition* function.
- To proceed, choose *System -> Services -> Jobs* or *System -> Services -> Reporting -> Program -> Background*, or call Transaction SM36.
- To make the request, you need to specify a job name (any name you like), the name of the program to be started and a variant, as well as the name of a user for authorization checking purposes.
- The scheduling part ends when you *Save*. Before doing this, you can specify the start time, how often the job is to be repeated and the print parameters.
- You can extend an existing job to include more steps. When doing this, you get a step list which you can edit to suit your requirements.
- In distributed environments, you can specify a target machine where the processing is to take place (choose this by pressing **F4**).
- By specifying a job class, you can determine the priority and type of a job.