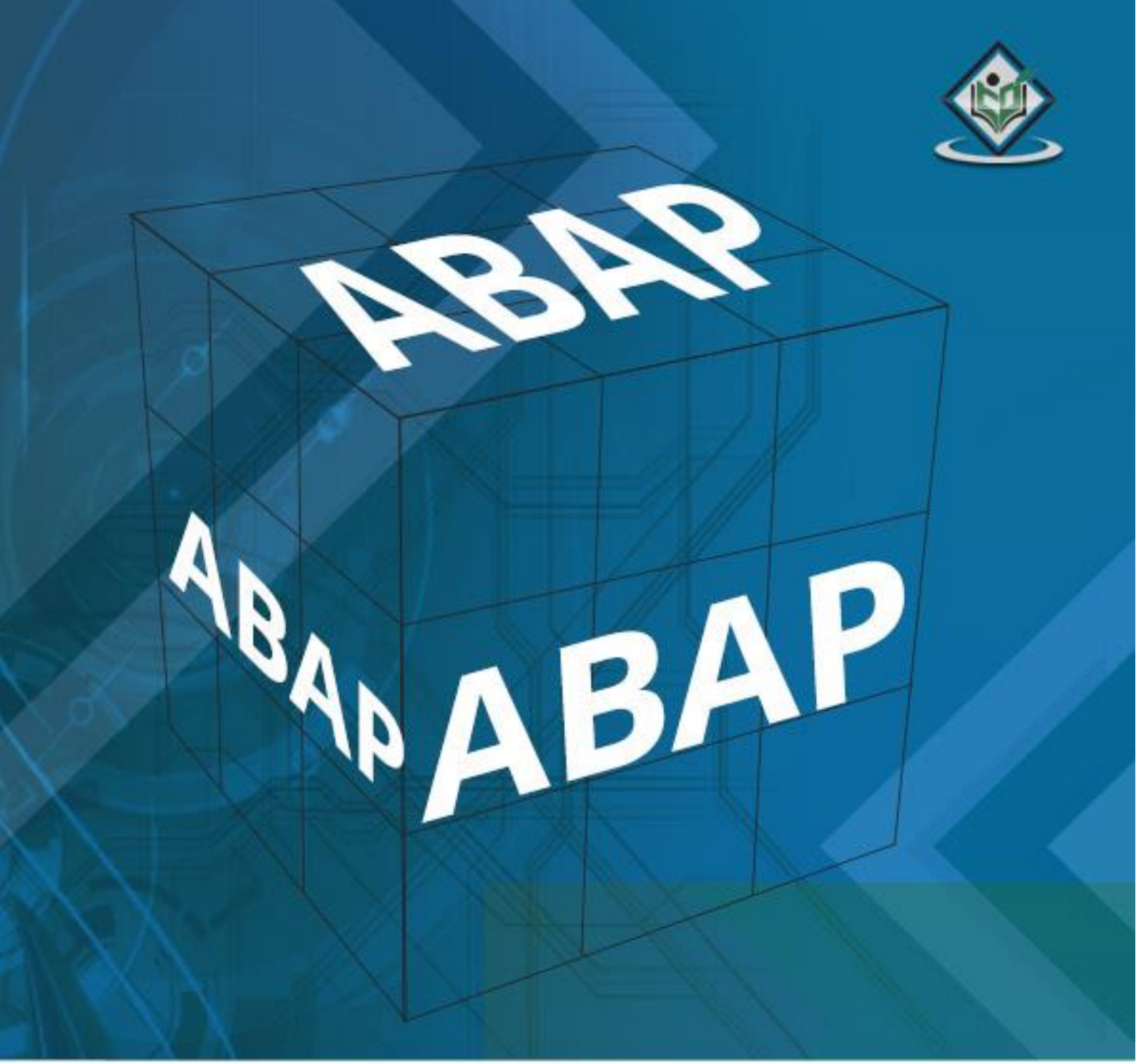


tutorialspoint

SIMPLYEASYLEARNING

About the Tutorial

ABAP (Advanced Business Application Programming), is a fourth-generation programming language, used for development and customization purposes in the SAP software. Currently positioned along with Java, as the main language for SAP application server programming, most of the programs are executed under the control of the run-time system. This tutorial explains the key concepts of SAP ABAP.

Audience

SAP ABAP is a high level language that is primarily used to develop enterprise application for large business and financial institution on SAP platform. This tutorial is designed for those who want to learn the basics of SAP ABAP and advance in the field of software development.

Prerequisites

You need to have a basic understanding of Java programming and Database technologies like PL/SQL to make the most of this tutorial.

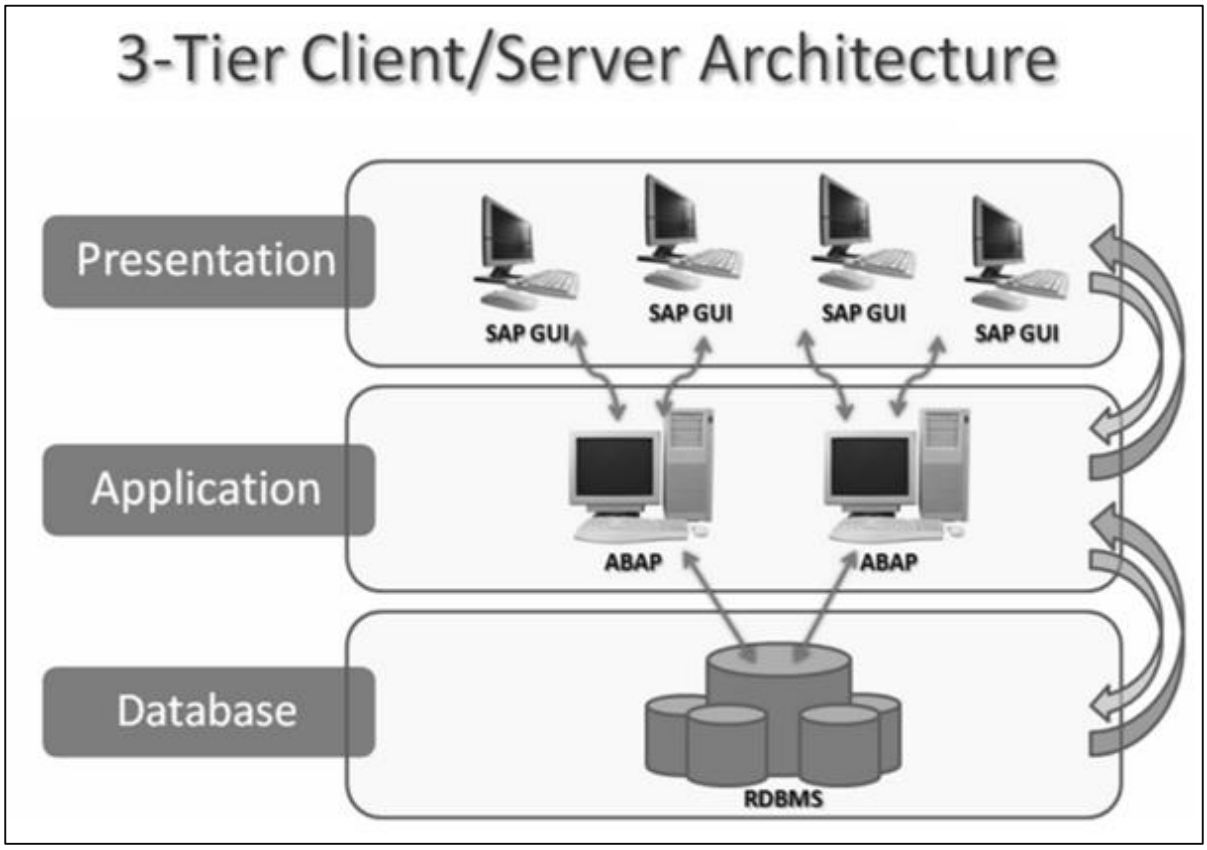
Disclaimer & Copyright

Table of Contents

1. ABAP – Overview

ABAP stands for Advanced Business Application Programming, a 4GL (4th generation) language. Currently it is positioned, along with Java, as the main language for SAP application server programming.

Let's start with the high level architecture of SAP system. The 3-tier Client/Server architecture of a typical SAP system is depicted as follows.



The **Presentation layer** consists of any input device that can be used to control SAP system. This could be a web browser, a mobile device and so on. All the central processing takes place in **Application server**. The Application server is not just one system in itself, but it can be multiple instances of the processing system. The server communicates with the **Database layer** that is usually kept on a separate server, mainly for performance reasons and also for security. Communication happens between each layer of the system, from the Presentation layer to the Database and then back up the chain.

Note: ABAP programs run at the application server level. Technical distribution of software is independent of its physical location. It means basically all three levels can be installed on top of each other on one computer or each level can be installed on a different computer or a server.

ABAP programs reside inside the SAP database. They execute under the control of the run-time system that is a part of the SAP kernel. The run-time system processes all ABAP statements, controlling the flow logic and responding to user events.

So, unlike C++ and Java, ABAP programs are not stored in separate external files. Inside the database, ABAP code exists in two forms:

- **Source code** that can be viewed and edited with the ABAP workbench tools.
- **Generated code**, which is a binary representation. If you are familiar with Java, this generated code is somewhat comparable with Java byte code.

The run-time system can be considered as a virtual machine, just similar to Java virtual machine. A key component of the ABAP run-time system is the database interface that turns database independent statements (Open SQL) into the statements understood by the underlying database (Native SQL). SAP can work with a wide variety of databases and the same ABAP program can run on all of those.

2. ABAP – Environment

Reports are a good starting point for familiarizing yourself with general ABAP principles and tools. ABAP reports are used in many areas. In this chapter, we will see how easy it is to write a simple ABAP Report.

Hello ABAP

Let's get started with the common "Hello World" example.

Each ABAP statement starts with an ABAP keyword and ends with a period. Keywords must be separated by at least one space. It does not matter whether or not you use one or several lines for an ABAP statement.

You need to enter your code using the ABAP Editor that is a part of ABAP Tools delivered with the SAP NetWeaver Application Server ABAP (also known as 'AS ABAP').

'AS ABAP' is an application server with its own database, ABAP run-time environment, and ABAP development tools such as ABAP Editor. The AS ABAP offers a development platform that is independent of hardware, operating system, and database.

Using the ABAP Editor

Step 1: Start the transaction SE38 to navigate to the ABAP Editor (discussed in the next chapter). Let's start creating a report that is one of the many ABAP objects.

Step 2: On the initial screen of the editor, specify the name of your report in the input field PROGRAM. You may specify the name as ZHELLO1. The preceding Z is important for the name. Z ensures that your report resides in the customer namespace.

The customer namespace includes all objects with the prefix Y or Z. It is always used when customers or partners create objects (like a report) to differentiate these objects from objects of SAP and to prevent name conflicts with objects.

Step 3: You may type the report name in lower case letters, but the editor will change it to upper case. So the names of ABAP objects are 'Not' case sensitive.

Step 4: After specifying the name of the report, click the CREATE button. A popup window ABAP: PROGRAM ATTRIBUTES will pop up and you will provide more information about your report.

Step 5: Choose "Executable Program" as the report type, enter the title "My First ABAP Report" and then select SAVE to continue. The CREATE OBJECT DIRECTORY ENTRY window will pop up next. Select the button LOCAL OBJECT and the popup will close.

You can complete your first report by entering the WRITE statement below the REPORT statement, so that the complete report contains just two lines as follows:

```
REPORT ZHELLO1.
WRITE 'Hello World'.
```

Starting the Report

We can use the keyboard (Ctrl + S) or the save icon (right hand side beside the command field) to save the report. ABAP development takes place in AS ABAP.

Starting the report is as simple as saving it. Click the ACTIVATION button (left hand side next to the start icon) and start the report by using the icon DIRECT PROCESSING or the F8 function key. The title "My First ABAP Report" along with the output "Hello World" is displayed as well. Here is the output:

```
My First ABAP Report

Hello World
```

As long as you do not activate a new report or activate a change to an existing report, it is not relevant to their users. This is important in a central development environment where you may work on objects that other developers use in their projects.

Viewing the Existing Code

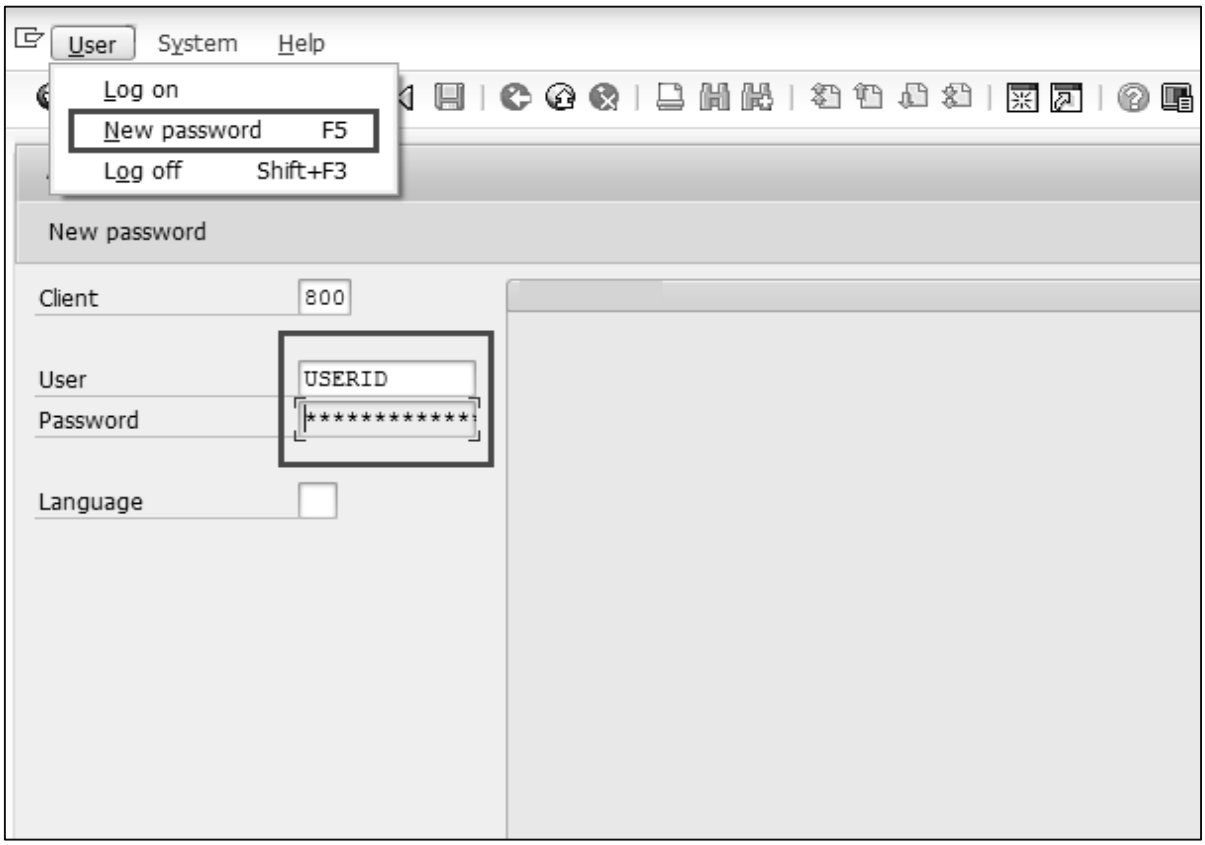
If you look at the field Program and double-click on the value ZHELLO1, the ABAP editor will display the code for your report. This is called Forward Navigation. Double clicking on an object's name opens that object in the appropriate tool.

3. ABAP – Screen Navigation

In order to understand SAP ABAP, you need to have basic knowledge of screens like Login, ABAP Editor, Logout and so on. This chapter focuses on screen navigation and the standard toolbar functionality.

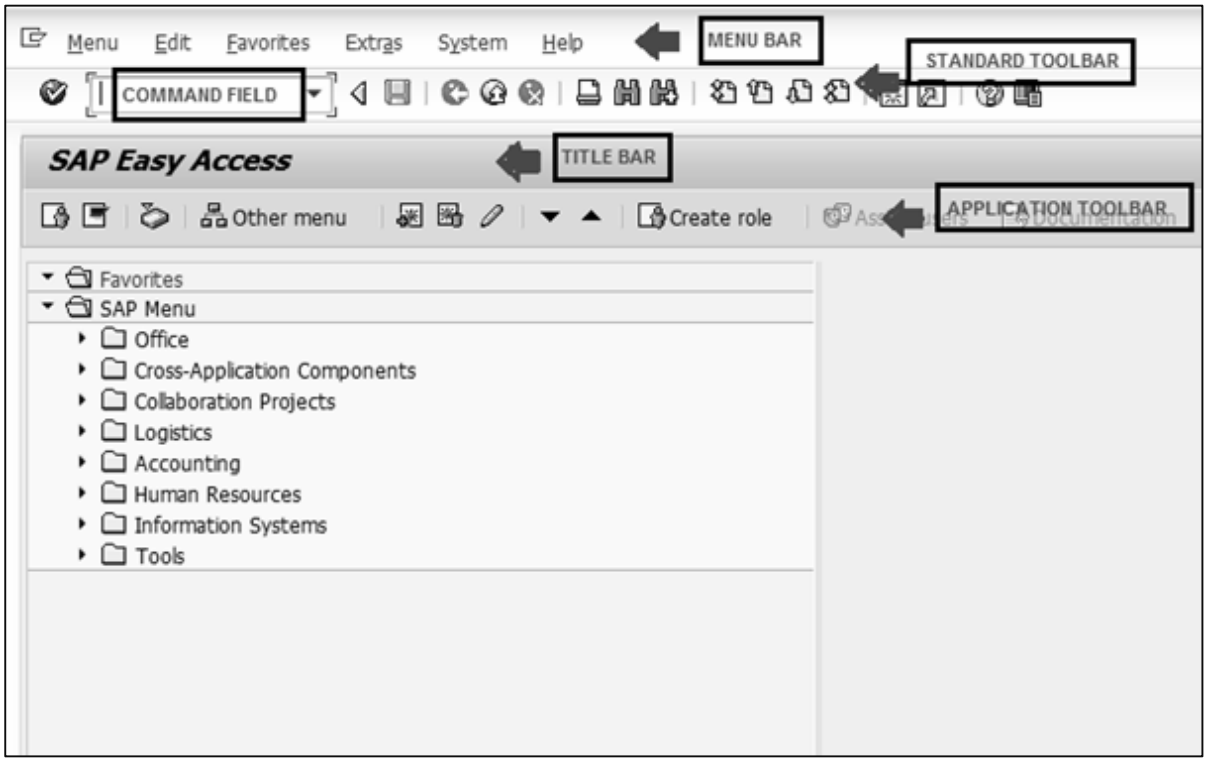
Login Screen

After you log on to SAP server, SAP login screen will prompt for User ID and Password. You need to provide a valid user ID and Password and press Enter (the user id and password is provided by system administrator). Following is the login screen.



Toolbar Icon

Following is the SAP screen toolbar.



Menu Bar: Menu bar is the top line of dialog window.

Standard Toolbar: Most standard functions such as Top of Page, End of Page, Page Up, Page Down and Save are available in this toolbar.

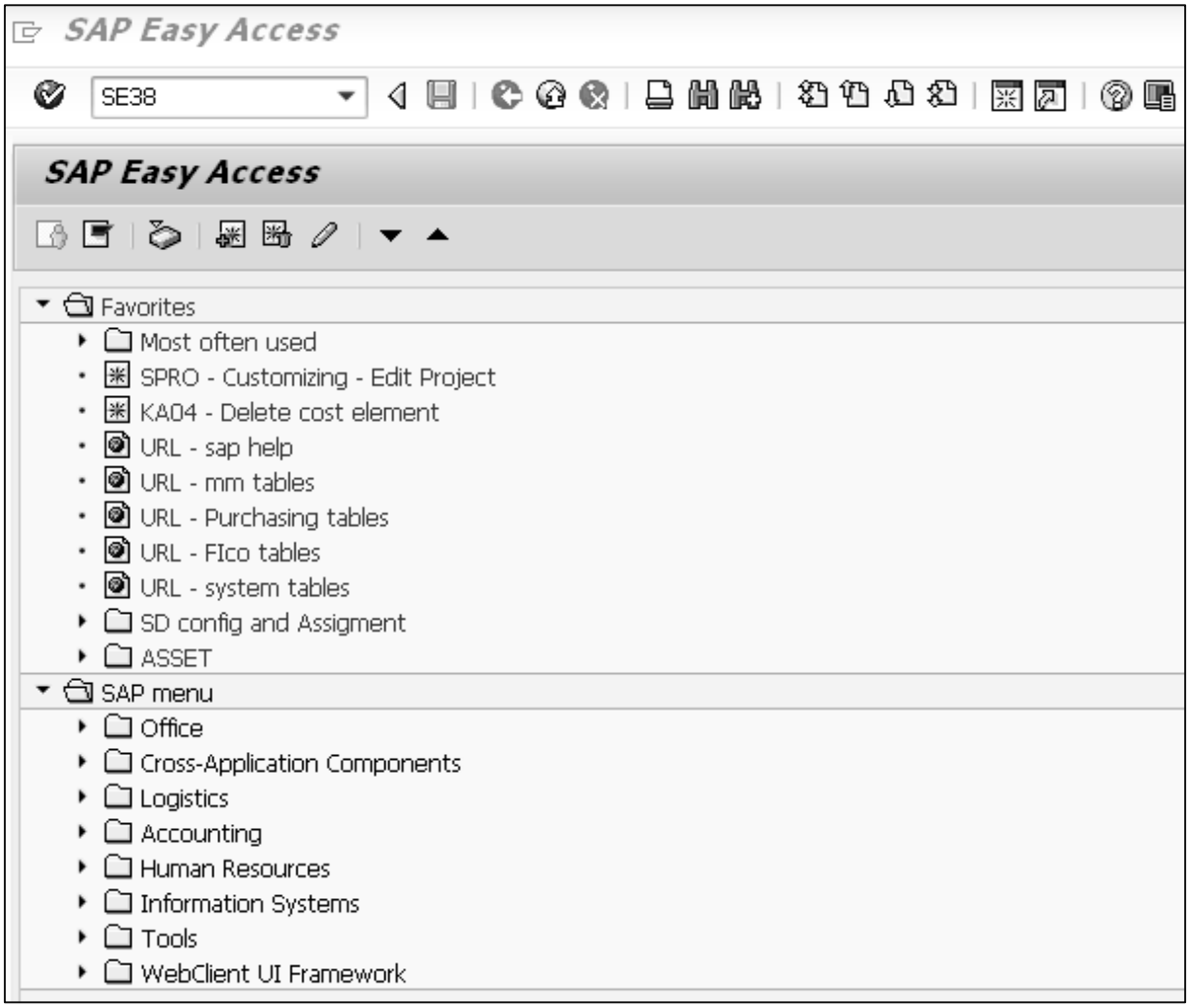
Title Bar: Title Bar displays the name of the application/business process you are currently in.

Application Toolbar: Application specific menu options are available here.

Command Field: We can start an application without navigating through the menu transactions and some logical codes are assigned to business processes. Transaction codes are entered in the command field to directly start the application.

ABAP Editor

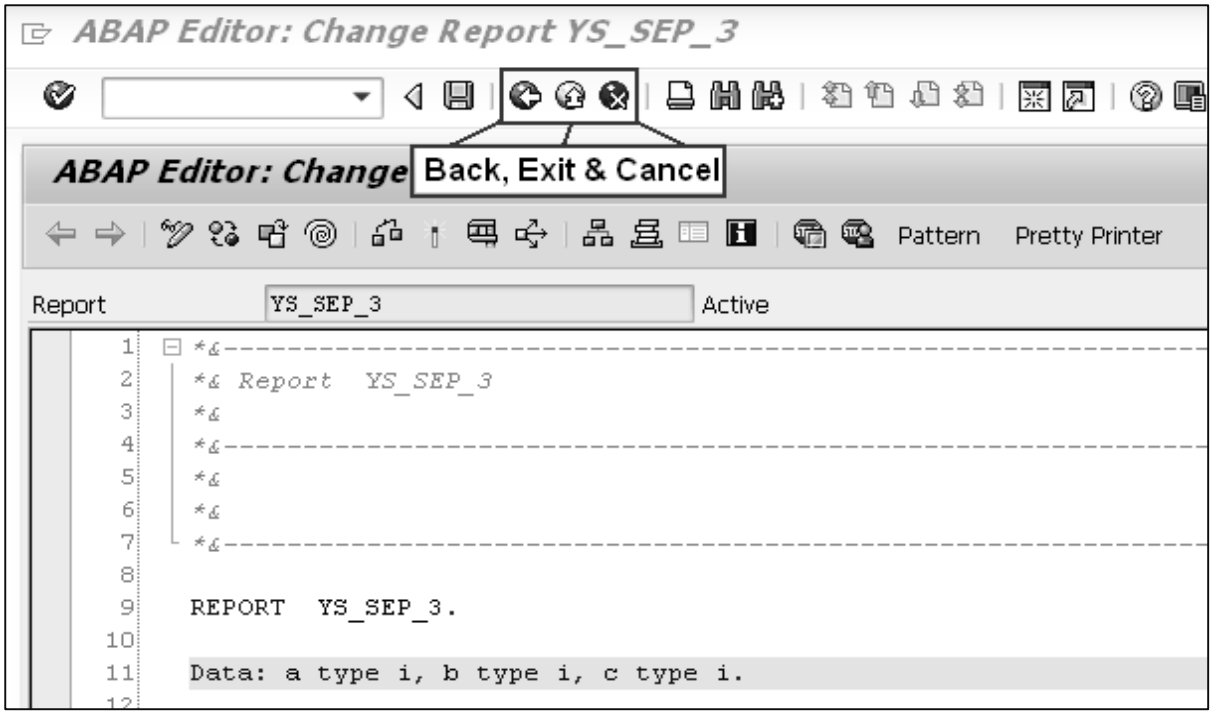
You may just start the transaction SE38 (enter SE38 in Command Field) to navigate to the ABAP Editor.



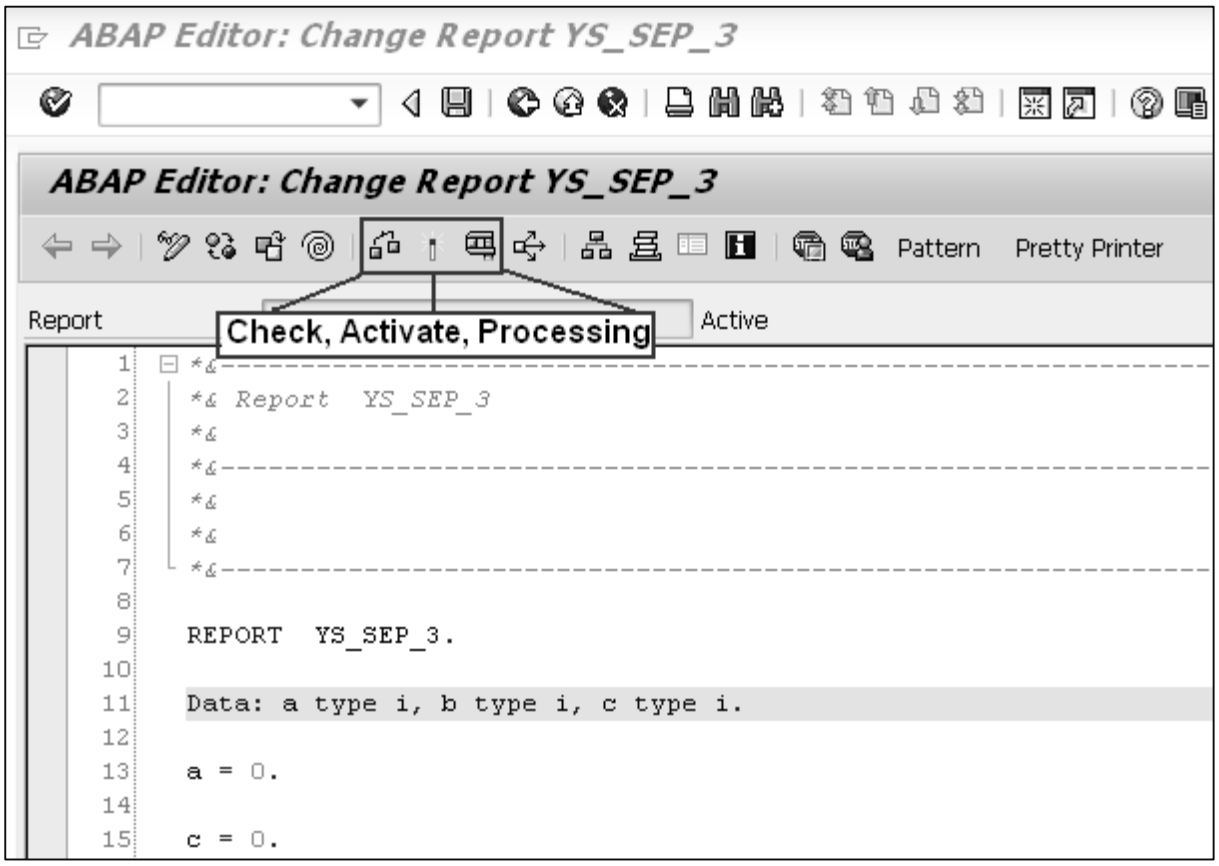
Standard Keys and Icons

Exit keys are used to exit the program/module or to log off. They are also used to go back to the last accessed screen.

Following are the standard exit keys used in SAP as shown in the image.

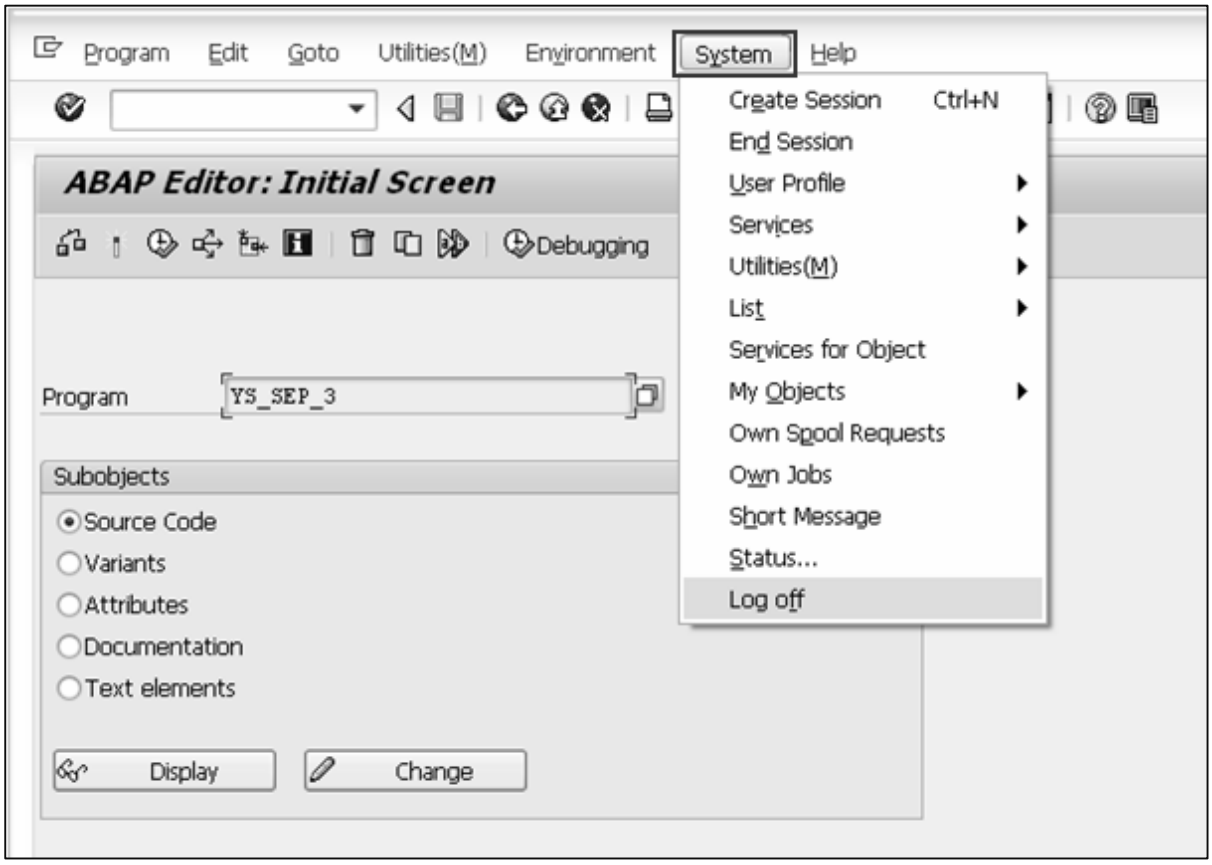


Following are the options for checking, activating and processing the reports.



Log Off

It's always a good practice to Exit from your ABAP Editor or/and logoff from the SAP system after finishing your work.



4. ABAP – Basic Syntax

Statements

ABAP source program consists of comments and ABAP statements. Every statement in ABAP begins with a keyword and ends with a period, and ABAP is 'Not' case sensitive.

The first non-comment line in a program begins with the word `REPORT`. The Report will always be the first line of any executable program created. The statement is followed by the program name which was created previously. The line is then terminated with a full stop.

The syntax is:

```
REPORT [Program_Name].


[Statements…].
```

This allows the statement to take up as many lines in the editor as it needs. For example, the REPORT may look like this:

```
REPORT Z_Test123_01.
```

Statements consist of a command and any variables and options, ending with a period. As long as the period appears at the end of the statement, no problems will arise. It is this period that marks where the statement finishes.

Let's write the code.

On the line below the REPORT statement, just type this statement: Write 'ABAP Tutorial'.

```
REPORT Z_Test123_01.


Write 'This is ABAP Tutorial'.
```

Four things to consider while writing statements:

- The write statement writes whatever is in quotes to the output window.
- The ABAP editor converts all text to uppercase except text strings, which are surrounded by single quotation marks.

- Unlike some older programming languages, ABAP does not care where a statement begins on a line. You may take advantage of this and improve the readability of your program by using indentation to indicate blocks of code.

- ABAP has no restrictions on the layout of statements. That is, multiple statements can be placed on a single line, or a single statement may stretch across multiple lines.

Colon Notation

Consecutive statements can be chained together if the beginning of each statement is identical. This is done with the colon (:) operator and commas, which are used to terminate the individual statements, much as periods end normal statements.

Following is an example of a program that could save some key stroking:

```
WRITE 'Hello'.
WRITE 'ABAP'.
WRITE 'World'.
```

Using the colon notation, it could be rewritten this way:

```
WRITE: 'Hello',
       'ABAP',
       'World'.
```

Like any other ABAP statement, the layout doesn't matter. This is an equally correct statement:

```
WRITE: 'Hello', 'ABAP', 'World'.
```

Comments

Inline comments may be declared anywhere in a program by one of the two methods:

- Full line comments are indicated by placing an asterisk (*) in the first position of the line, in which case the entire line is considered by the system to be a comment. Comments don't need to be terminated by a period because they may not extend across more than one line:

  ```
  * This is the comment line
  ```

- Partial line comments are indicated by entering a double quote (") after a statement. All text following the double quote is considered by the system to be a comment. You need not terminate partial line comments by a period because they may not extend across more than one line:

```
    WRITE 'Hello'. "Here is the partial comment
```

Note: Commented code is not capitalized by the ABAP editor.

Suppressing Blanks

The NO-ZERO command follows the DATA statement. It suppresses all leading zeros of a number field containing blanks. The output is usually easier for the users to read.

Example

```
REPORT Z_Test123_01.

DATA: W_NUR(10) TYPE N.
      MOVE 50 TO W_NUR.
      WRITE W_NUR NO-ZERO.
```

The above code produces the following output:

```
50
```

Note: Without NO-ZERO command, the output is: 0000000050

Blank Lines

The SKIP command helps in inserting blank lines on the page.

Example

The message command is as follows:

```
WRITE 'This is the 1st line'.
SKIP.
WRITE 'This is the 2nd line'.
```

The above message command produces the following output:

```
    This is the 1st line

    This is the 2nd line
```

We may use the SKIP command to insert multiple blank lines.

```
SKIP number_of_lines.
```

The output would be several blank lines defined by the `number of lines`. The SKIP command can also position the cursor on a desired line on the page.

```
SKIP TO LINE line_number.
```

This command is used to dynamically move the cursor up and down the page. Usually, a `WRITE` statement occurs after this command to put output on that desired line.

Inserting Lines

The `ULINE` command automatically inserts a horizontal line across the output. It's also possible to control the position and length of the line. The syntax is pretty simple:

```
    ULINE.
```

Example

The message command is as follows:

```
    WRITE 'This is Underlined'.
    ULINE.
```

The above code produces the following output:

```
This is Underlined (and a horizontal line below this).
```

Messages

The MESSAGE command displays messages defined by a message ID specified in the REPORT statement at the beginning of the program. The message ID is a 2 character code that defines which set of 1,000 messages the program will access when the MESSAGE command is used.

The messages are numbered from 000 to 999. Associated with each number is a message text up to a maximum of 80 characters. When message number is called, the corresponding text is displayed.

Following are the characters for use with the Message command:

Message	Type	Consequences
E	**Error**	The message appears and the application halts at its current point. If the program is running in background mode, the job is canceled and the message is recorded in the job log.
W	**Warning**	

		The message appears and the user must press Enter for the application to continue. In background mode, the message is recorded in the job log.
I	**Information**	A pop-up window opens with the message text and the user must press Enter to continue. In background mode, the message is recorded in the job log.
A	**Abend**	This message class cancels the transaction that the user is currently using.
S	**Success**	This provides an informational message at the bottom of the screen. The information displayed is positive in nature and it is just meant for user feedback. The message does not impede the program in any way.
X	**Abort**	This message aborts the program and generates an ABAP short dump.

Error messages are normally used to stop users from doing things they are not supposed to do. Warning messages are generally used to remind the users of the consequences of their actions. Information messages give the users useful information.

Example

When we create a message for message the ID AB, the MESSAGE command - MESSAGE E011 gives the following output:

```
EAB011 This report does not support sub-number summarization.
```

5. ABAP – Data Types

While programming in ABAP, we need to use a variety of variables to store various information. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. You may like to store information of various data types like character, integer, floating point, etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

Elementary Data Types

ABAP offers the programmer a rich assortment of fixed length as well as variable length data types. Following table lists down ABAP elementary data types:

Type	Keyword
Byte field	X
Text field	C
Integer	I
Floating point	F
Packed number	P
Text string	STRING

Some of the fields and numbers can be modified using one or more names as the following:

- byte
- numeric
- character-like

The following table shows the data type, how much memory it takes to store the value in memory, and the minimum and maximum value that could be stored in such type of variables.

Type	Typical Length	Typical Range
X	1 byte	Any byte values (00 to FF)
C	1 character	1 to 65535
N (numeric text filed)	1 character	1 to 65535
D (character-like date)	8 characters	8 characters
T (character-like time)	6 characters	6 characters
I	4 bytes	-2147483648 to 2147483647

F	8 bytes	2.2250738585072014E-308 to 1.7976931348623157E+308 positive or negative
P	8 bytes	[-10^(2len -1) +1] to [+10^(2len -1) -1] (where len = fixed length)
STRING	Variable	Any alphanumeric characters
XSTRING (byte string)	Variable	Any byte values (00 to FF)

Example

```
REPORT YR_SEP_12.
DATA text_line TYPE C LENGTH 40.
text_line = 'A Chapter on Data Types'.
Write text_line.
DATA text_string TYPE STRING.
text_string = 'A Program in ABAP'.
Write / text_string.
DATA d_date TYPE D.
d_date = SY-DATUM.
Write / d_date.
```

In this example, we have a character string of type C with a predefined length 40. STRING is a data type that can be used for any character string of variable length (text strings). Type STRING data objects should generally be used for character-like content where fixed length is not important.

The above code produces the following output:

```
     A Chapter on Data Types

A Program in ABAP

12092015
```

The DATE type is used for the storage of date information and can store eight digits as shown above.

Complex and Reference Types

The complex types are classified into **Structure types** and **Table types**. In the structure types, elementary types and structures (i.e. structure embedded in a structure) are grouped together. You may consider only the grouping of elementary types. But you must be aware of the availability of nesting of structures.

When the elementary types are grouped together, the data item can be accessed as a grouped data item or the individual elementary type data items (structure fields) can be accessed. The table types are better known as arrays in other programming languages. **Arrays** can be simple or structure arrays. In ABAP, arrays are called internal tables and they can be declared and operated upon in many ways when compared to other

programming languages. The following table shows the parameters according to which internal tables are characterized.

Parameter	Description
Line or row type	Row of an internal table can be of elementary, complex or reference type.
Key	Specifies a field or a group of fields as a key of an internal table that identifies the table rows. A key contains the fields of elementary types.
Access method	Describes how ABAP programs access individual table entries.

Reference types are used to refer to instances of classes, interfaces, and run-time data items. The ABAP OOP run-time type services (RTTS) enables declaration of data items at run-time.

6. ABAP – Variables

Variables are named data objects used to store values within the allotted memory area of a program. As the name suggests, users can change the content of variables with the help of ABAP statements. Each variable in ABAP has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. The basic form of a variable declaration is:

```
DATA <f> TYPE <type> VALUE <val>.
```

Here <f> specifies the name of a variable. The name of the variable can be up to 30 characters long. <type> specifies the type of variable. Any data type with fully specified technical attributes is known as <type>. The <val> specifies the initial value of the of <f> variable. In case you define an elementary fixed-length variable, the DATA statement automatically populates the value of the variable with the type-specific initial value. Other possible values for <val> can be a literal, constant, or an explicit clause, such as Is INITIAL.

Following are valid examples of variable declarations.

```
DATA d1(2) TYPE C.
DATA d2 LIKE d1.
DATA minimum_value TYPE I VALUE 10.
```

In the above code snippet, d1 is a variable of C type, d2 is a variable of d1 type, and minimum_value is a variable of ABAP integer type I.

This chapter will explain various variable types available in ABAP. There are three kinds of variables in ABAP:

- Static Variables
- Reference Variables
- System Variables

Static Variables

- Static variables are declared in subroutines, function modules, and static methods.
- The lifetime is linked to the context of the declaration.
- With 'CLASS-DATA' statement, you can declare variables within the classes.
- The 'PARAMETERS' statement can be used to declare the elementary data objects that are linked to input fields on a selection screen.

- You can also declare the internal tables that are linked to input fields on a selection screen by using 'SELECT-OPTIONS' statement.

Following are the conventions used while naming a variable:

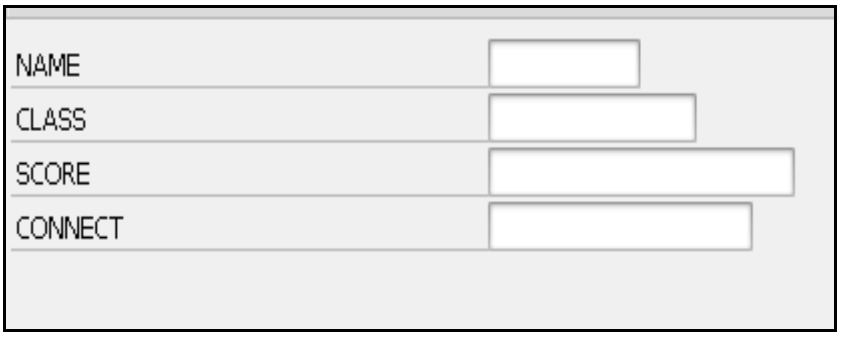
- You cannot use special characters such as "t" and "," to name variables.
- The name of the predefined data objects can't be changed.
- The name of the variable can't be the same as any ABAP keyword or clause.
- The name of the variables must convey the meaning of the variable without the need for further comments.
- Hyphens are reserved to represent the components of structures. Therefore, you are supposed to avoid hyphens in variable names.
- The underscore character can be used to separate compound words.

This program shows how to declare a variable using the PARAMETERS statement:

```
REPORT ZTest123_01.
PARAMETERS: NAME(10) TYPE C,
CLASS TYPE I,
SCORE TYPE P DECIMALS 2,
CONNECT TYPE MARA-MATNR.
```

Here, NAME represents a parameter of 10 characters, CLASS specifies a parameter of integer type with the default size in bytes, SCORE represents a packed type parameter with values up to two decimal places, and CONNECT refers to the MARA-MATNF type of ABAP Dictionary.

The above code produces the following output:



Reference Variables

The syntax for declaring reference variables is:

```
DATA <ref> TYPE REF TO <type> VALUE IS INITIAL.
```

- REF TO addition declares a reference variable ref.

- The specification after REF TO specifies the static type of the reference variable.
- The static type restricts the set of objects to which <ref> can refer.
- The dynamic type of reference variable is the data type or class to which it currently refers.
- The static type is always more general or the same as the dynamic type.
- The TYPE addition is used to create a bound reference type and as a start value, and only IS INITIAL can be specified after the VALUE addition.

Example

```
CLASS C1 DEFINITION.
PUBLIC SECTION.
DATA B1 TYPE I VALUE 1.
ENDCLASS.
DATA: Oref TYPE REF TO C1 ,
Dref1 LIKE REF TO Oref,
Dref2 TYPE REF TO I .
CREATE OBJECT Oref.
GET REFERENCE OF Oref INTO Dref1.
CREATE DATA Dref2.
Dref2->* = Dref1->*->B1.
```

- In the above code snippet, an object reference Oref and two data reference variables Dref1 and Dref2 are declared.
- Both data reference variables are fully typed and can be dereferenced using the dereferencing operator `->*` at operand positions.

System Variables

- ABAP system variables are accessible from all ABAP programs.
- These fields are actually filled by the run-time environment.
- The values in these fields indicate the state of the system at any given point of time.
- You can find the complete list of system variables in the SYST table in SAP.
- Individual fields of the SYST structure can be accessed by using either "SYST-" or "SY-".

Example

```
REPORT Z_Test123_01.
WRITE:/'SY-ABCDE', SY-ABCDE,
      /'SY-DATUM', SY-DATUM,
      /'SY-DBSYS', SY-DBSYS,
      /'SY-HOST ', SY-HOST,
      /'SY-LANGU', SY-LANGU,
      /'SY-MANDT', SY-MANDT,
      /'SY-OPSYS', SY-OPSYS,
      /'SY-SAPRL', SY-SAPRL,
      /'SY-SYSID', SY-SYSID,
      /'SY-TCODE', SY-TCODE,
      /'SY-UNAME', SY-UNAME,
      /'SY-UZEIT', SY-UZEIT.
```

The above code produces the following output:

```
SY-ABCDE    ABCDEFGHIJKLMNOPQRSTUVWXYZ
SY-DATUM    12.09.2015
SY-DBSYS    ORACLE
SY-HOST     sapserver
SY-LANGU    EN
SY-MANDT    800
SY-OPSYS    Windows NT
SY-SAPRL    700
SY-SYSID    DMO
SY-TCODE    SE38
SY-UNAME    SAPUSER
SY-UZEIT    14:25:48
```

End of ebook preview

If you liked what you saw…

Buy it from our store @ https://store.tutorialspoint.com