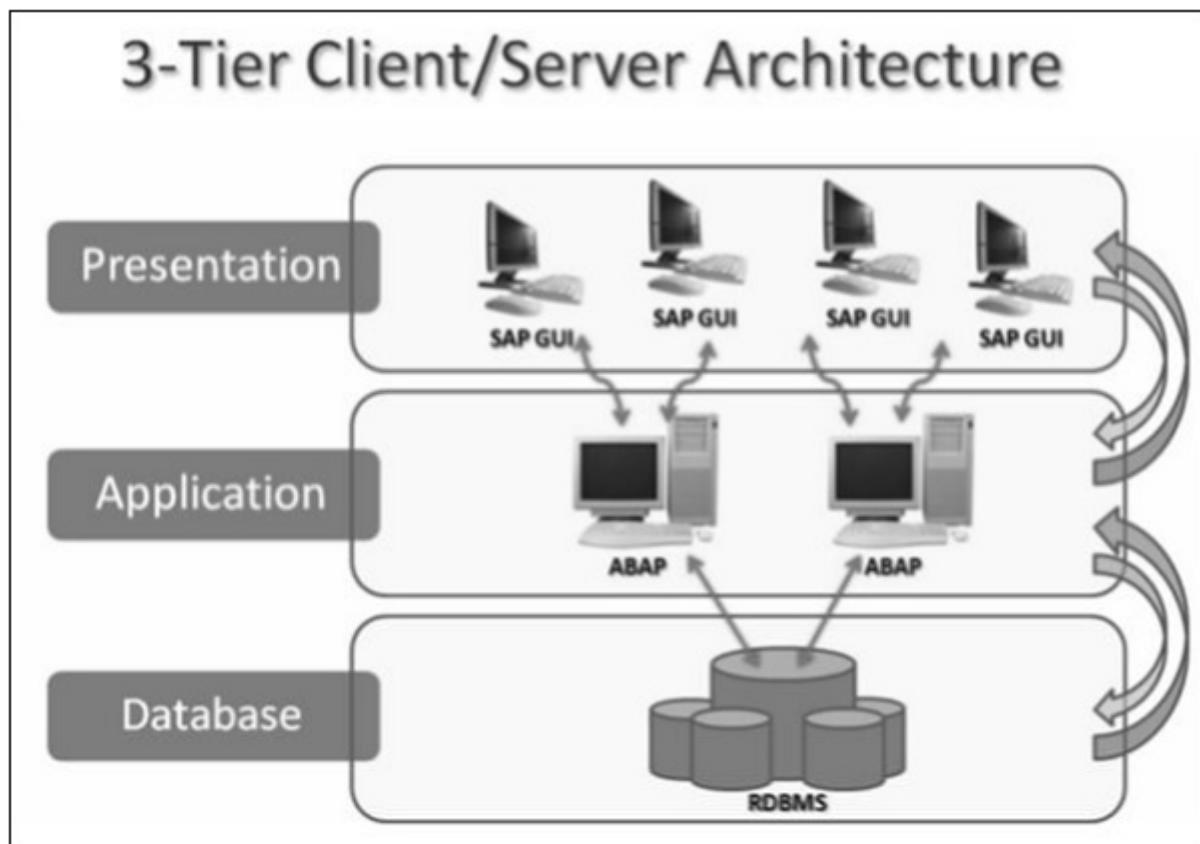


SAP ABAP - Guia rápido

SAP ABAP - Visão geral

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.



A **camada de apresentação** consiste em qualquer dispositivo de entrada que pode ser usado para controlar o sistema SAP. Pode ser um navegador da web, um dispositivo móvel e assim por diante. Todo o processamento central ocorre no **servidor de aplicativos**. O servidor de aplicativos não é apenas um sistema em si, mas pode ser várias instâncias do sistema de processamento. O servidor se comunica com a **camada de Banco de Dados** que normalmente é mantida em um servidor separado, principalmente por questões de desempenho e também ---

segurança. A comunicação acontece entre cada camada do sistema, desde a camada de Apresentação até o Banco de Dados e depois subindo na cadeia.

Observação - Os programas ABAP são executados no nível do servidor de aplicativos. A distribuição técnica de software é independente de sua localização física. Isso significa que basicamente todos os três níveis podem ser instalados uns sobre os outros em um computador ou cada nível pode ser instalado em um computador ou servidor diferente.

Os programas ABAP residem no banco de dados SAP. Eles são executados sob o controle do sistema de tempo de execução que faz parte do kernel SAP. O sistema em tempo de execução processa todas as instruções ABAP, controlando a lógica do fluxo e respondendo aos eventos do usuário.

Portanto, diferentemente de C++ e Java, os programas ABAP não são armazenados em arquivos externos separados. Dentro do banco de dados, o código ABAP existe em duas formas -

- **Código fonte** que pode ser visualizado e editado com as ferramentas do ambiente de trabalho ABAP.
- **Código gerado**, que é uma representação binária. Se você estiver familiarizado com Java, esse código gerado é comparável ao código de bytes Java.

O sistema de tempo de execução pode ser considerado uma máquina virtual, semelhante à máquina virtual Java. Um componente chave do sistema de tempo de execução ABAP é a interface do banco de dados que transforma instruções independentes do banco de dados (Open SQL) em instruções compreendidas pelo banco de dados subjacente (Native SQL). SAP pode trabalhar com uma ampla variedade de bancos de dados e o mesmo programa ABAP pode ser executado em todos eles.

SAP ABAP - Ambiente

Os relatórios são um bom ponto de partida para se familiarizar com os princípios e ferramentas gerais do ABAP. Os relatórios ABAP são usados em muitas áreas. Neste capítulo, veremos como é fácil escrever um relatório ABAP simples.

Olá ABAP

Vamos começar com o exemplo comum “Hello World”.

Cada instrução ABAP começa com uma palavra-chave ABAP e termina com um ponto final. As palavras-chave devem ser separadas por pelo menos um espaço. Não importa se você usa ou não uma ou várias linhas para uma instrução ABAP.

Você precisa inserir seu código usando o ABAP Editor que faz parte das ferramentas ABAP fornecidas com o SAP NetWeaver Application Server ABAP (também conhecido como 'AS ABAP').

'AS ABAP' é um servidor de aplicativos com seu próprio banco de dados, ambiente de execução ABAP e ferramentas de desenvolvimento ABAP, como ABAP Editor. O AS ABAP oferece uma plataforma de desenvolvimento independente de hardware, sistema operacional e banco de dados.

Usando o editor ABAP

Step 1 - Inicie a transação SE38 para navegar até o ABAP Editor (discutido no próximo capítulo). Vamos começar a criar um relatório que seja um dos muitos objetos ABAP.

Step 2 - Na tela inicial do editor, especifique o nome do seu relatório no campo de entrada PROGRAMA. Você pode especificar o nome como ZHELLO1. O Z anterior é importante para o nome. Z garante que seu relatório resida no namespace do cliente.

O namespace do cliente inclui todos os objetos com o prefixo Y ou Z. Ele é sempre usado quando clientes ou parceiros criam objetos (como um relatório) para diferenciar esses objetos dos objetos do SAP e para evitar conflitos de nomes com objetos.

Step 3 - Você pode digitar o nome do relatório em letras minúsculas, mas o editor irá alterá-lo para maiúsculas. Portanto, os nomes dos objetos ABAP 'Não' diferenciam maiúsculas de minúsculas.

Step 4 - Após especificar o nome do relatório, clique no botão CRIAR. Uma janela pop-up ABAP: ATRIBUTOS DO PROGRAMA aparecerá e você fornecerá mais informações sobre seu relatório.

Step 5 - Escolha "Programa Executável" como tipo de relatório, insira o título "Meu primeiro relatório ABAP" e selecione SALVAR para continuar. A janela CREATE OBJECT DIRECTORY ENTRY aparecerá a seguir. Selecione o botão LOCAL OBJECT e o pop-up será fechado.

Você pode completar seu primeiro relatório inserindo a instrução WRITE abaixo da instrução REPORT, para que o relatório completo contenha apenas duas linhas como segue -

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

Iniciando o relatório

Podemos usar o teclado (Ctrl + S) ou o ícone salvar (lado direito ao lado do campo de comando) para salvar o relatório. O desenvolvimento ABAP ocorre em AS ABAP.

Iniciar o relatório é tão simples quanto salvá-lo. Clique no botão ATIVAÇÃO (lado esquerdo próximo ao ícone iniciar) e inicie o relatório usando o ícone PROCESSAMENTO DIRETO ou a tecla de função F8. O título “Meu primeiro relatório ABAP” juntamente com a saída “Hello World” também são exibidos. Aqui está o resultado -

```
My First ABAP Report
Hello World
```

Contanto que você não ative um novo relatório ou ative uma alteração em um relatório existente, isso não será relevante para seus usuários. Isto é importante em um ambiente de desenvolvimento central onde você pode trabalhar em objetos que outros desenvolvedores usam em seus projetos.

Visualizando o código existente

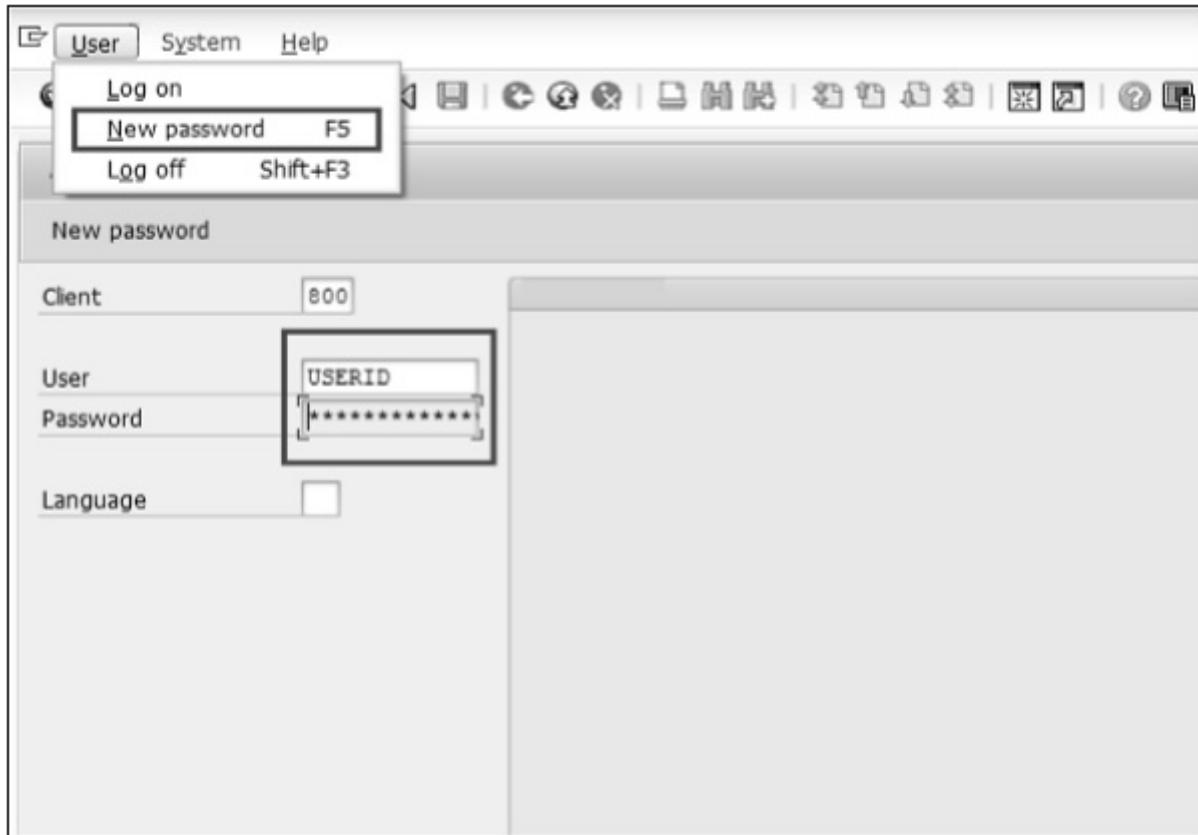
Se você olhar o campo Programa e clicar duas vezes no valor ZHELLO1, o editor ABAP exibirá o código do seu relatório. Isso é chamado de navegação direta. Clicar duas vezes no nome de um objeto abre esse objeto na ferramenta apropriada.

SAP ABAP - Navegação na tela

Para entender o SAP ABAP, você precisa ter conhecimento básico de telas como Login, Editor ABAP, Logout e assim por diante. Este capítulo se concentra na navegação na tela e na funcionalidade padrão da barra de ferramentas.

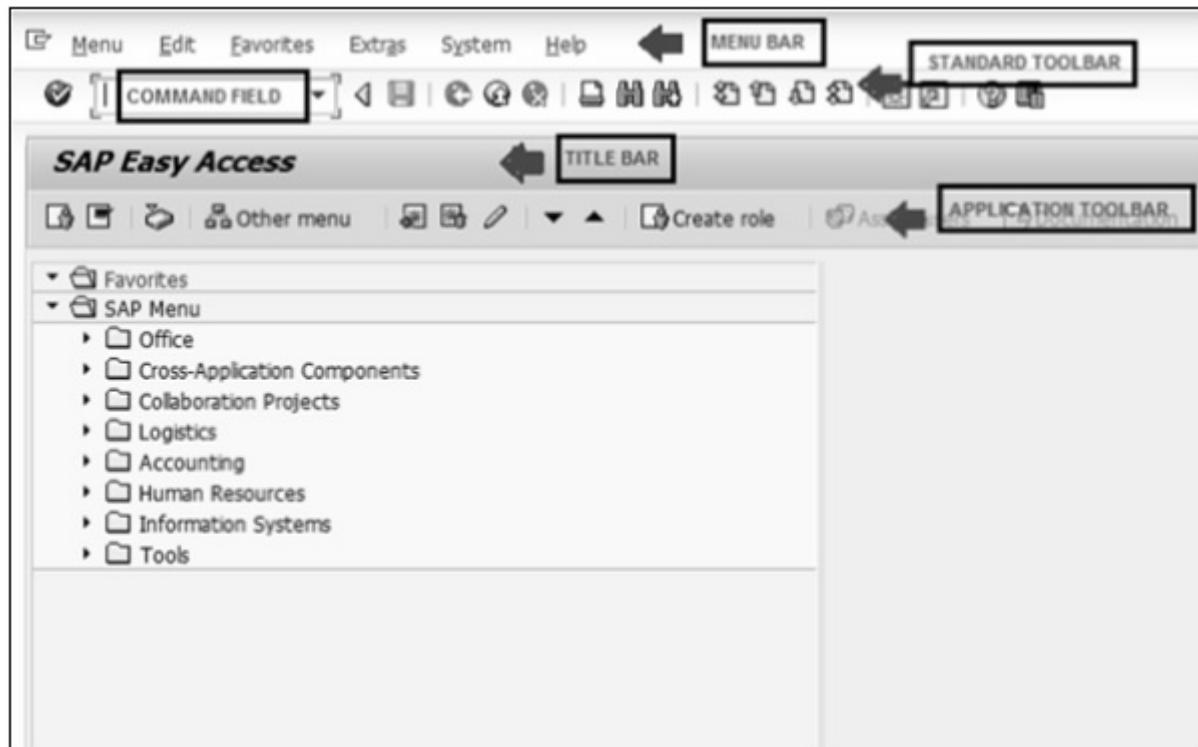
Tela de login

Depois de fazer logon no servidor SAP, a tela de login do SAP solicitará ID de usuário e senha. Você precisa fornecer um ID de usuário e senha válidos e pressionar Enter (o ID de usuário e a senha são fornecidos pelo administrador do sistema). A seguir está a tela de login.



Ícone da barra de ferramentas

A seguir está a barra de ferramentas da tela SAP.



Menu Bar - A barra de menu é a linha superior da janela de diálogo.

Standard Toolbar - A maioria das funções padrão, como Top of Page, End of Page, Page Up, Page Down e Save estão disponíveis nesta barra de ferramentas.

Title Bar - A barra de título exibe o nome do aplicativo/processo de negócios em que você está atualmente.

Application Toolbar - As opções de menu específicas do aplicativo estão disponíveis aqui.

Command Field - Podemos iniciar uma aplicação sem navegar pelas transações do menu e alguns códigos lógicos são atribuídos aos processos de negócio. Os códigos de transação são inseridos no campo de comando para iniciar o aplicativo diretamente.

Editor ABAP

Você pode simplesmente iniciar a transação SE38 (inserir SE38 no campo de comando) para navegar até o Editor ABAP.



Teclas e ícones padrão

As teclas de saída são usadas para sair do programa/módulo ou para fazer logoff. Eles também são usados para voltar à última tela acessada.

A seguir estão as chaves de saída padrão usadas no SAP, conforme mostrado na imagem.

ABAP Editor: Change Report YS_SEP_3

```

1  *->-----
2  *-> Report  YS_SEP_3
3  *->
4  *->-----
5  *->
6  *->
7  *->-----
8
9  REPORT  YS_SEP_3.
10
11  Data: a type i, b type i, c type i.
12
13
14
15

```

A seguir estão as opções para verificação, ativação e processamento dos relatórios.

ABAP Editor: Change Report YS_SEP_3

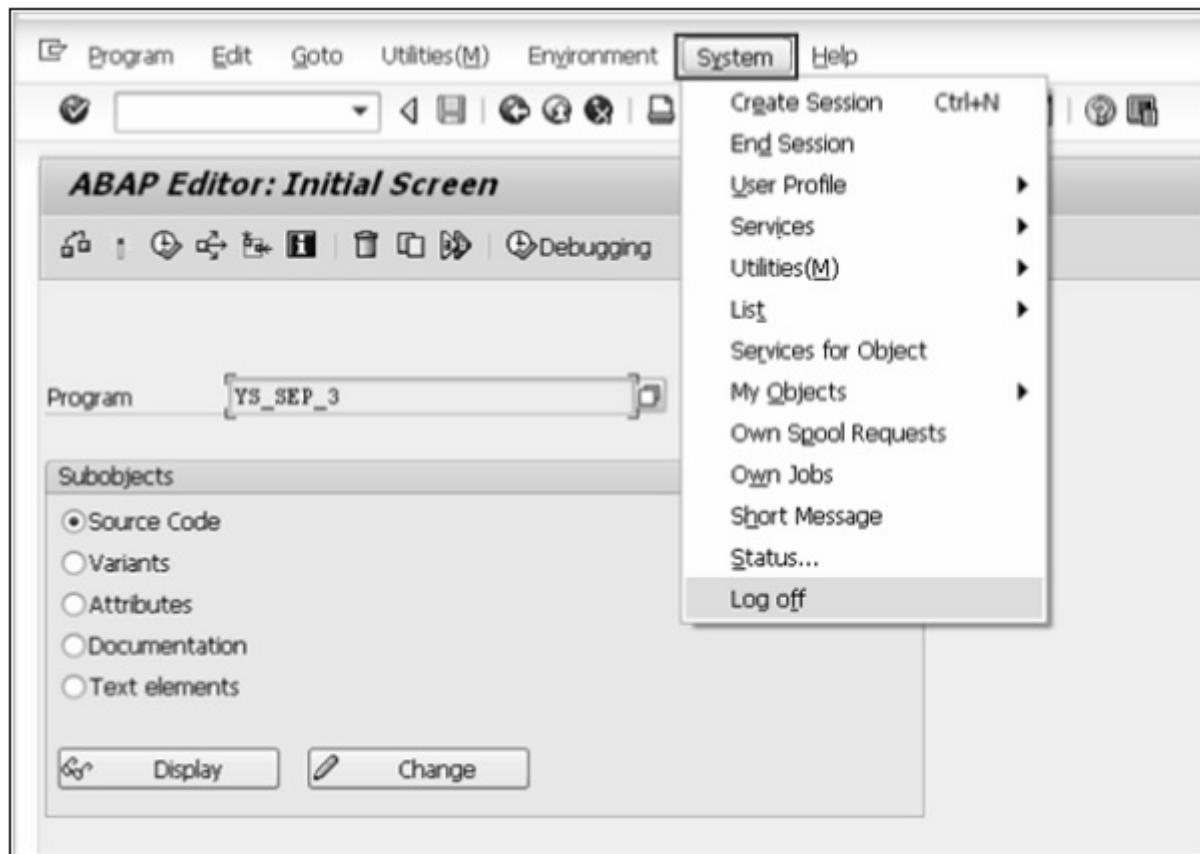
```

1  *->-----
2  *-> Report  YS_SEP_3
3  *->
4  *->-----
5  *->
6  *->
7  *->-----
8
9  REPORT  YS_SEP_3.
10
11  Data: a type i, b type i, c type i.
12
13  a = 0.
14
15  c = 0.

```

Sair

É sempre uma boa prática sair do Editor ABAP e/ou fazer logoff do sistema SAP após terminar seu trabalho.



SAP ABAP - Sintaxe Básica

Declarações

O programa fonte ABAP consiste em comentários e instruções ABAP. Cada instrução no ABAP começa com uma palavra-chave e termina com um ponto, e o ABAP diferencia maiúsculas de minúsculas.

A primeira linha sem comentários em um programa começa com a palavra REPORT. O Relatório sempre estará na primeira linha de qualquer programa executável criado. A instrução é seguida pelo nome do programa que foi criado anteriormente. A linha é então encerrada com um ponto final.

A sintaxe é -

```
REPORT [Program_Name].
```

```
[Statements...].
```

Isso permite que a instrução ocupe quantas linhas forem necessárias no editor. Por exemplo, o RELATÓRIO pode ser assim -

```
REPORT Z_Test123_01.
```

As instruções consistem em um comando e quaisquer variáveis e opções, terminando com um ponto final. Desde que o ponto final apareça no final do extrato, não surgirão problemas. É este período que marca onde termina a declaração.

Vamos escrever o código.

Na linha abaixo da instrução REPORT, basta digitar esta instrução: Escreva 'ABAP Tutorial'.

```
REPORT Z_Test123_01.
```

```
Write 'This is ABAP Tutorial'.
```

Quatro coisas a considerar ao escrever declarações -

- A instrução write escreve tudo o que está entre aspas na janela de saída.
- O editor ABAP converte todo o texto em letras maiúsculas, exceto strings de texto, que são colocadas entre aspas simples.
- Ao contrário de algumas linguagens de programação mais antigas, o ABAP não se importa onde uma instrução começa em uma linha. Você pode tirar vantagem disso e melhorar a legibilidade do seu programa usando indentação para indicar blocos de código.
- ABAP não tem restrições quanto ao layout das instruções. Ou seja, diversas instruções podem ser colocadas em uma única linha ou uma única instrução pode se estender por diversas linhas.

Notação de dois pontos

Declarações consecutivas podem ser encadeadas se o início de cada afirmação for idêntico. Isso é feito com o operador dois pontos (:) e vírgulas, que são usados para encerrar as instruções individuais, da mesma forma que os pontos terminam as instruções normais.

A seguir está um exemplo de um programa que pode economizar algumas teclas -

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

Usando a notação de dois pontos, poderia ser reescrito desta forma -

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

Como qualquer outra instrução ABAP, o layout não importa. Esta é uma afirmação igualmente correta -

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

Comentários

Comentários embutidos podem ser declarados em qualquer lugar de um programa por um dos dois métodos -

- Comentários de linha completa são indicados colocando um asterisco (*) na primeira posição da linha, caso em que toda a linha é considerada pelo sistema como um comentário. Os comentários não precisam ser encerrados com ponto final porque não podem se estender por mais de uma linha -

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

- Os comentários de linha parcial são indicados inserindo aspas duplas ("") após uma instrução. Todo o texto após as aspas duplas é considerado pelo sistema como um comentário. Você não precisa encerrar os comentários de linha parcial por um ponto porque eles não podem se estender por mais de uma linha -

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

Note - O código comentado não é maiúsculo pelo editor ABAP.

Suprimindo espaços em branco

O comando NO-ZERO segue a instrução DATA. Ele suprime todos os zeros à esquerda de um campo numérico contendo espaços em branco. A saída geralmente é mais fácil de ler para os usuários.

Exemplo

```
REPORT Z_Test123_01.
```

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

O código acima produz a seguinte saída -

50

Nota - Sem o comando NO-ZERO, a saída é: 0000000050

Linhas em branco

O comando SKIP auxilia na inserção de linhas em branco na página.

Exemplo

O comando da mensagem é o seguinte -

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

O comando da mensagem acima produz a seguinte saída -

```
This is the 1st line
This is the 2nd line
```

Podemos usar o comando SKIP para inserir várias linhas em branco.

```
SKIP number_of_lines.
```

A saída seria várias linhas em branco definidas pelo número de linhas. O comando SKIP também pode posicionar o cursor em uma linha desejada na página.

```
SKIP TO LINE line_number.
```

Este comando é usado para mover dinamicamente o cursor para cima e para baixo na página. Normalmente, uma instrução WRITE ocorre após este comando para colocar a saída na linha desejada.

Inserindo Linhas

O comando ULINE insere automaticamente uma linha horizontal na saída. Também é possível controlar a posição e o comprimento da linha. A sintaxe é bem simples -

```
ULINE.
```

Exemplo

O comando da mensagem é o seguinte -

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

O código acima produz a seguinte saída -

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

Mensagens

O comando MESSAGE exibe mensagens definidas por um ID de mensagem especificado na instrução REPORT no início do programa. O ID da mensagem é um código de 2 caracteres que define qual conjunto de 1.000 mensagens o programa acessará quando o comando MESSAGE for utilizado.

As mensagens são numeradas de 000 a 999. Associado a cada número está um texto de mensagem com no máximo 80 caracteres. Quando o número da mensagem é chamado, o texto correspondente é exibido.

A seguir estão os caracteres para uso com o comando Mensagem -

Mensagem	Tipo	Consequências
E	Erro	A mensagem aparece e o aplicativo é interrompido no ponto atual. Se o programa estiver sendo executado em segundo plano, o trabalho será cancelado e a mensagem será registrada no log do trabalho.
C	Aviso	A mensagem aparece e o usuário deve pressionar Enter para que a aplicação continue. No modo de segundo plano, a mensagem é gravada no log de tarefas.
EU	Informação	Uma janela pop-up é aberta com o texto da mensagem e o usuário deve pressionar Enter para continuar. No modo de segundo plano, a mensagem é gravada no log de tarefas.
A	Encerrar	Esta classe de mensagem cancela a transação que o usuário está usando atualmente.
S	Sucesso	Isso fornece uma mensagem informativa na parte inferior da tela. As informações exibidas são de natureza positiva e destinam-se apenas ao feedback do usuário. A mensagem não atrapalha de forma alguma o programa.
X	Abortar	Esta mensagem anula o programa e gera um short dump ABAP.

Mensagens de erro normalmente são usadas para impedir que os usuários façam coisas que não deveriam fazer. Mensagens de aviso são geralmente usadas para lembrar os usuários das consequências de suas ações. As mensagens informativas fornecem aos usuários informações úteis.

Exemplo

Quando criamos uma mensagem para a mensagem ID AB, o comando MESSAGE - MESSAGE E011 fornece a seguinte saída -

EAB011 This report does not support sub-number summarization.

SAP ABAP - Tipos de dados

Ao programar em ABAP, precisamos usar uma variedade de variáveis para armazenar diversas informações. Variáveis nada mais são do que locais de memória reservados para armazenar valores. Isso significa que ao criar uma variável você reserva algum espaço na memória. Você pode querer armazenar informações de vários tipos de dados, como caractere, número inteiro, ponto flutuante, etc. Com base no tipo de dados de uma variável, o sistema operacional aloca memória e decide o que pode ser armazenado na memória reservada.

Tipos de dados elementares

ABAP oferece ao programador uma rica variedade de tipos de dados de comprimento fixo e também de comprimento variável. A tabela a seguir lista os tipos de dados elementares ABAP -

Tipo	Palavra-chave
Campo de bytes	X
Campo de texto	C
Inteiro	I
Ponto flutuante	F
Número embalado	P
Sequência de texto	CORDA

Alguns dos campos e números podem ser modificados usando um ou mais nomes como a seguir -

- byte
- numérico
- semelhante a um personagem

A tabela a seguir mostra o tipo de dados, quanta memória é necessária para armazenar o valor na memória e o valor mínimo e máximo que pode ser armazenado neste tipo de variáveis.

Tipo	Comprimento típico	Faixa Típica
X	1 byte	Quaisquer valores de byte (00 a FF)
C	1 personagem	1 a 65535
N (texto numérico arquivado)	1 personagem	1 a 65535
D (data semelhante a um caractere)	8 caracteres	8 caracteres
T (tempo semelhante ao personagem)	6 caracteres	6 caracteres
EU	4 bytes	-2147483648 a 2147483647
F	8 bytes	2.2250738585072014E-308 a 1.7976931348623157E+308 positivo ou negativo
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.

S.No.	Parameter & Description
1	Line or row type Row of an internal table can be of elementary, complex or reference type.
2	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.
3	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.

SAP 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 <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 –

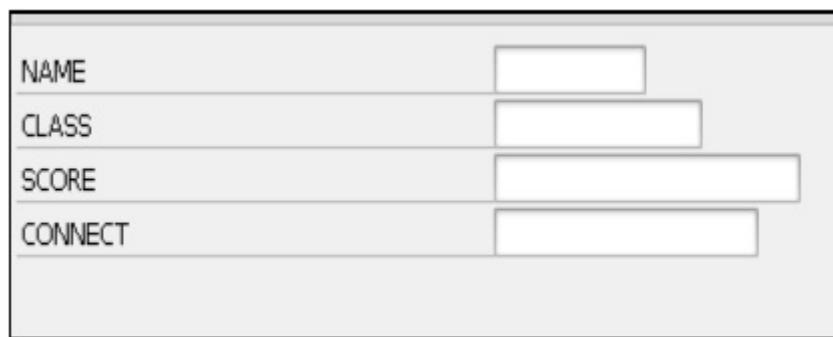
- You cannot use special characters such as "t" and "," to name variables.
- The name of the predefined data objects can't be changed.
- The name of the variable can't be the same as any ABAP keyword or clause.
- The name of the variables must convey the meaning of the variable without the need for further 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.
```

- A adição REF TO declara uma variável de referência ref.
- A especificação após REF TO especifica o tipo estático da variável de referência.
- O tipo estático restringe o conjunto de objetos aos quais <ref> pode se referir.
- O tipo dinâmico de variável de referência é o tipo de dados ou classe ao qual ela se refere atualmente.
- O tipo estático é sempre mais geral ou igual ao tipo dinâmico.
- A adição de TYPE é usada para criar um tipo de referência vinculado e como um valor inicial, e somente IS INITIAL pode ser especificado após a adição de VALUE.

Exemplo

```
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.
```

- No trecho de código acima, uma referência de objeto Oref e duas variáveis de referência de dados Dref1 e Dref2 são declaradas.
- Ambas as variáveis de referência de dados são totalmente digitadas e podem ser desreferenciadas usando o operador de desreferenciação →* nas posições dos operandos.

Variáveis do sistema

- Variáveis de sistema ABAP são acessíveis em todos os programas ABAP.
- Na verdade, esses campos são preenchidos pelo ambiente de tempo de execução.
- Os valores nestes campos indicam o estado do sistema em qualquer momento.
- Você pode encontrar a lista completa de variáveis do sistema na tabela SYST no SAP.
- Campos individuais da estrutura SYST podem ser acessados usando “SYST-” ou “SY-”.

Exemplo

```
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-OPSY', SY-OPSY,
      / 'SY-SAPRL', SY-SAPRL,
      / 'SY-SYSID', SY-SYSID,
      / 'SY-TCODE', SY-TCODE,
      / 'SY-UNAME', SY-UNAME,
      / 'SY-UZEIT', SY-UZEIT.
```

O código acima produz a seguinte saída -

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

SAP ABAP - Constantes e Literais

Literals are unnamed data objects that you create within the source code of a program. They are fully defined by their value. You can't change the value of a literal. Constants are named data objects created statically by using declarative statements. A constant is declared by assigning a value to it that is stored in the program's memory area. The value assigned to a constant can't be changed during the execution of the program. These fixed values can also be considered as literals. There are two types of literals – numeric and character.

Numeric Literals

Number literals are sequences of digits which can have a prefixed sign. In number literals, there are no decimal separators and no notation with mantissa and exponent.

Following are some examples of numeric literals –

183.
-97.
+326.

Character Literals

Literais de caracteres são sequências de caracteres alfanuméricos no código-fonte de um programa ABAP entre aspas simples. Literais de caracteres entre aspas têm o tipo ABAP predefinido C e são descritos como literais de campo de texto. Literais entre “aspas crases” têm o tipo ABAP STRING e são descritos como literais de string. O comprimento do campo é definido pelo número de caracteres.

Note - Em literais de campo de texto, os espaços em branco são ignorados, mas em literais de string eles são levados em consideração.

A seguir estão alguns exemplos de literais de caracteres.

Literais de campo de texto

```
REPORT YR_SEP_12.  
Write 'Tutorials Point'.  
Write / 'ABAP Tutorial'.
```

Literais de campo de string

```
REPORT YR_SEP_12.  
Write `Tutorials Point `.  
Write / `ABAP Tutorial `.
```

A saída é a mesma em ambos os casos acima -

```
Tutorials Point  
ABAP Tutorial
```

Note - Quando tentamos alterar o valor da constante, pode ocorrer um erro de sintaxe ou de tempo de execução. As constantes que você declara na parte de declaração de uma classe

interface pertencem aos atributos estáticos dessa classe ou interface.

Declaração CONSTANTES

Podemos declarar os objetos de dados nomeados com a ajuda da instrução CONSTANTS.

A seguir está a sintaxe -

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

A instrução CONSTANTS é semelhante à instrução DATA.

<f> especifica um nome para a constante. TYPE <type> representa uma constante chamada <f>, que herda os mesmos atributos técnicos do tipo de dados existente <type>. VALUE <val> atribui um valor inicial ao nome da constante declarada <f>.

Note - Devemos usar a cláusula VALUE na instrução CONSTANTS. A cláusula 'VALUE' é utilizada para atribuir um valor inicial à constante durante sua declaração.

Temos 3 tipos de constantes, como constantes elementares, complexas e de referência. A instrução a seguir mostra como definir constantes usando a instrução CONSTANTS -

```
REPORT YR_SEP_12.
CONSTANTS PQR TYPE P DECIMALS 4 VALUE '1.2356'.
Write: / 'The value of PQR is:', PQR.
```

A saída é -

```
The value of PQR is: 1.2356
```

Aqui se refere ao tipo de dados elementar e é conhecido como constante elementar.

A seguir está um exemplo para constantes complexas -

```
BEGIN OF EMPLOYEE,
Name(25) TYPE C VALUE 'Management Team',
Organization(40) TYPE C VALUE 'Tutorials Point Ltd',
Place(10) TYPE C VALUE 'India',
END OF EMPLOYEE.
```

No trecho de código acima, EMPLOYEE é uma constante complexa composta pelos campos Nome, Organização e Local.

A declaração a seguir declara uma referência constante -

```
CONSTANTS null_pointer TYPE REF TO object VALUE IS INITIAL.
```

Podemos usar a referência constante em comparações ou podemos repassá-la para procedimentos.

SAP ABAP - Operadores

ABAP fornece um rico conjunto de operadores para manipular variáveis. Todos os operadores ABAP são classificados em quatro categorias -

- Operadores aritméticos
- Operadores de comparação
- Operadores bit a bit
- Operadores de cadeia de caracteres

Operadores aritméticos

Os operadores aritméticos são usados em expressões matemáticas da mesma forma que são usados na álgebra. A lista a seguir descreve operadores aritméticos. Suponha que a variável inteira A contenha 20 e a variável B contenha 40.

S. Não.	Operador Aritmético e Descrição
1	+ (Adição) Adiciona valores em ambos os lados do operador. Exemplo: A + B dará 60.
2	- (Subtração) Subtrai o operando direito do operando esquerdo. Exemplo: A - B dará -20.
3	* (Multiplicação) Multiplica valores em ambos os lados do operador. Exemplo: A * B dará 800.
4	/ (Divisão) Divide o operando esquerdo pelo operando direito. Exemplo: B/A dará 2.
5	MOD (Módulo) Divide o operando esquerdo pelo operando direito e retorna o restante. Exemplo: B MOD A dará 0.

Exemplo

```

REPORT YS_SEP_08.
DATA: A TYPE I VALUE 150,
      B TYPE I VALUE 50,
      Result TYPE I.
      Result = A / B.
      WRITE / Result.
    
```

O código acima produz a seguinte saída -

3

Operadores de comparação

Vamos discutir os vários tipos de operadores de comparação para diferentes operandos.

S. Não.	Operador de comparação e descrição
1	<p>= (teste de igualdade). A forma alternativa é EQ.</p> <p>Verifica se os valores de dois operandos são iguais ou não; se sim, a condição torna-se verdadeira. O exemplo ($A = B$) não é verdadeiro.</p>
2	<p><> (Teste de desigualdade). A forma alternativa é NE.</p> <p>Verifica se os valores de dois operandos são iguais ou não. Se os valores não forem iguais, a condição se torna verdadeira. Exemplo ($A <> B$) é verdadeiro.</p>
3	<p>> (Maior que teste). A forma alternativa é GT.</p> <p>Verifica se o valor do operando esquerdo é maior que o valor do operando direito. Se sim, a condição se torna verdadeira. O exemplo ($A > B$) não é verdadeiro.</p>
4	<p>< (Menos que teste). A forma alternativa é LT.</p> <p>Verifica se o valor do operando esquerdo é menor que o valor do operando direito. Se sim, então a condição se torna verdadeira. O exemplo ($A < B$) é verdadeiro.</p>
5	<p>>= (Maior ou igual) A forma alternativa é GE.</p> <p>Verifica se o valor do operando esquerdo é maior ou igual ao valor do operando direito. Se sim, então a condição se torna verdadeira. O exemplo ($A >= B$) não é verdadeiro.</p>
6	<p><= (teste menor ou igual). A forma alternativa é LE.</p> <p>Verifica se o valor do operando esquerdo é menor ou igual ao valor do operando direito. Se sim, então a condição se torna verdadeira. Exemplo ($A <= B$) é verdadeiro.</p>
7	<p>a1 ENTRE a2 E a3 (teste de intervalo)</p> <p>Verifica se a1 está entre a2 e a3 (inclusive). Se sim, então a condição se torna verdadeira. O exemplo ($A \text{ ENTRE } B \text{ E } C$) é verdadeiro.</p>
8	<p>É INICIAL</p> <p>A condição torna-se verdadeira se o conteúdo da variável não tiver sido alterado e seu valor inicial tiver sido atribuído automaticamente. Exemplo ($A \text{ IS INITIAL}$) não</p>

	verdadeiro
9	<p>NÃO É INICIAL</p> <p>A condição se torna verdadeira se o conteúdo da variável for alterado. O exemplo (A NÃO É INICIAL) é verdadeiro.</p>

Note - Se o tipo de dados ou comprimento das variáveis não corresponderem, a conversão automática será executada. O ajuste automático de tipo é executado para um ou ambos os valores ao comparar dois valores de tipos de dados diferentes. O tipo de conversão é decidido pelo tipo de dados e pela ordem de preferência do tipo de dados.

A seguir está a ordem de preferência -

- Se um campo for do tipo I, o outro será convertido para o tipo I.
- Se um campo for do tipo P, o outro será convertido para o tipo P.
- Se um campo for do tipo D, o outro será convertido para o tipo D. Mas os tipos C e N não são convertidos e são comparados diretamente. Semelhante é o caso do tipo T.
- Se um campo for do tipo N e o outro for do tipo C ou X, ambos os campos serão convertidos para o tipo P.
- Se um campo for do tipo C e o outro for do tipo X, o tipo X será convertido para o tipo C.

Exemplo 1

```
REPORT YS_SEP_08.
```

```
DATA: A TYPE I VALUE 115,
      B TYPE I VALUE 119.
IF A LT B.
  WRITE: / 'A is less than B'.
ENDIF
```

O código acima produz a seguinte saída -

A is less than B

Exemplo 2

```
REPORT YS_SEP_08.
```

```
DATA: A TYPE I.
IF A IS INITIAL.
```

```
WRITE: / 'A is assigned'.
ENDIF.
```

O código acima produz a seguinte saída -

A is assigned.

Operadores bit a bit

ABAP também fornece uma série de operadores lógicos bit a bit que podem ser usados para construir expressões algébricas booleanas. Os operadores bit a bit podem ser combinados em expressões complexas usando parênteses e assim por diante.

S. Não.	Operador e descrição bit a bit
1	POUCO NÃO Operador unário que inverte todos os bits de um número hexadecimal para o valor oposto. Por exemplo, aplicar este operador a um número hexadecimal com o valor de nível de bit 10101010 (por exemplo, 'AA') daria 01010101.
2	BIT-E Este operador binário compara cada campo bit a bit usando o operador booleano AND.
3	BIT-XOR Operador binário que compara cada campo bit a bit usando o operador booleano XOR (OR exclusivo).
4	BIT-OU Operador binário que compara cada campo bit a bit usando o operador booleano OR.

Por exemplo, a seguir está a tabela verdade que mostra os valores gerados ao aplicar os operadores booleanos AND, OR ou XOR aos dois valores de bits contidos no campo A e no campo B.

Campo A	Campo B	E	OU	XOR
---------	---------	---	----	-----

0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Operadores de cadeia de caracteres

A seguir está uma lista de operadores de strings de caracteres -

S. Não.	Operador e descrição de string de caracteres
1	CO (contém apenas) Verifica se A é composto apenas pelos caracteres de B.
2	CN (não contém APENAS) Verifica se A contém caracteres que não estão em B.
3	CA (contém QUALQUER) Verifica se A contém pelo menos um caractere de B.
4	NA (NÃO contém nenhum) Verifica se A não contém nenhum caractere de B.
5	CS (contém uma string) Verifica se A contém a sequência de caracteres B.
6	NS (NÃO contém uma string) Verifica se A não contém a sequência de caracteres B.
7	CP (contém um padrão) Ele verifica se A contém o padrão em B.
8	NP (NÃO contém um padrão) Ele verifica se A não contém o padrão em B.

Exemplo

```
REPORT YS_SEP_08.
DATA: P(10) TYPE C VALUE 'APPLE',
      Q(10) TYPE C VALUE 'CHAIR'.
IF P CA Q.
```

```
WRITE: / 'P contains at least one character of Q'.
ENDIF.
```

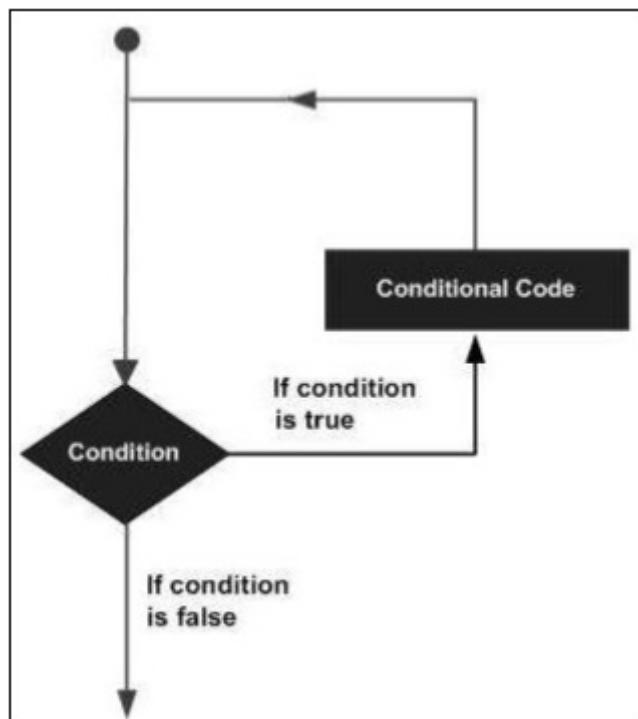
O código acima produz a seguinte saída -

```
P contains at least one character of Q.
```

SAP ABAP - Controle de Loop

Pode haver uma situação em que você precise executar um bloco de código várias vezes. Em geral, as instruções são executadas sequencialmente: a primeira instrução em uma função é executada primeiro, seguida pela segunda e assim por diante.

As linguagens de programação fornecem várias estruturas de controle que permitem caminhos de execução mais complicados. Uma **instrução de loop** nos permite executar uma instrução ou grupo de instruções várias vezes e a seguir está a forma geral de uma instrução de loop na maioria das linguagens de programação.



A linguagem de programação ABAP fornece os seguintes tipos de loop para lidar com os requisitos de loop.

S. Não.	Tipo e descrição do loop
1	<p>Ciclo ENQUANTO</p> <p>Repete uma declaração ou grupo de declarações quando uma determinada condição é verdadeira. Ele testa a condição antes de executar o corpo do loop.</p>
2	<p>Faça um loop</p> <p>A instrução DO é útil para repetir uma tarefa específica um número específico de vezes.</p>
3	<p>Loop aninhado</p> <p>Você pode usar um ou mais loops dentro de qualquer outro loop WHILE ou DO.</p>

Declarações de controle de loop

As instruções de controle de loop alteram a execução de sua sequência normal. ABAP inclui instruções de controle que permitem que os loops sejam encerrados prematuramente. Ele oferece suporte às seguintes instruções de controle.

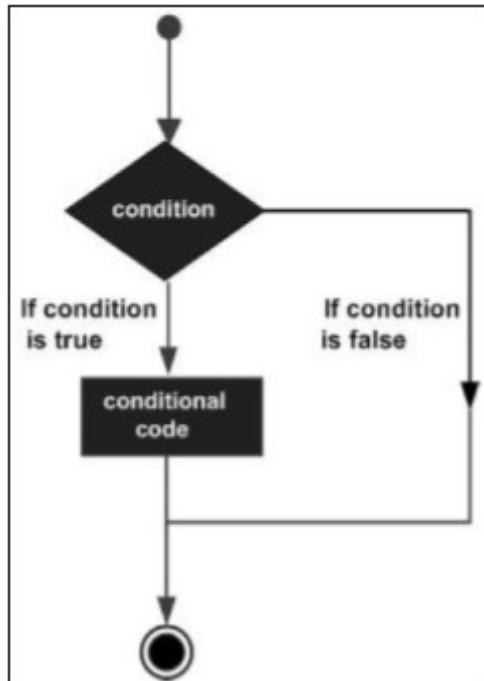
S. Não.	Declaração e descrição de controle
1	<p>CONTINUAR</p> <p>Faz com que o loop pule o restante do seu corpo e inicia a próxima passagem do loop.</p>
2	<p>VERIFICAR</p> <p>Se a condição for falsa, as instruções restantes após CHECK serão simplesmente ignoradas e o sistema iniciará a próxima passagem do loop.</p>
3	<p>SAÍDA</p> <p>Encerra totalmente o loop e transfere a execução para a instrução imediatamente após o loop.</p>

SAP ABAP - Decisões

As estruturas de tomada de decisão têm uma ou mais condições a serem avaliadas ou testadas pelo programa, juntamente com uma instrução ou instruções que devem ser executadas, s

condição for determinada como verdadeira e, opcionalmente, outras instruções a serem executadas, se a condição é determinado como falso.

A seguir está a forma geral de uma estrutura típica de tomada de decisão encontrada na maioria das linguagens de programação -



A linguagem de programação ABAP fornece os seguintes tipos de declarações de tomada de decisão.

S. Não.	Declaração e Descrição
1	<p>Declaração SE</p> <p>Uma instrução IF consiste em uma expressão lógica seguida por uma ou mais instruções.</p>
2	<p>SE.. Declaração Else</p> <p>Uma instrução IF pode ser seguida por uma instrução ELSE opcional que é executada quando a expressão é falsa.</p>
3	<p>Instrução IF aninhada</p> <p>Você pode usar uma instrução IF ou ELSEIF dentro de outra instrução IF ou ELSEIF.</p>
4	<p>Declaração de controle CASE</p> <p>A instrução CASE é usada quando precisamos comparar dois ou mais campos ou variáveis.</p>

SAP ABAP - Sequências

Strings, amplamente utilizadas na programação ABAP, são uma sequência de caracteres.

Utilizamos variáveis do tipo C para armazenar caracteres alfanuméricos, com mínimo de 1 caractere e máximo de 65.535 caracteres. Por padrão, eles estão alinhados à esquerda.

Criando Strings

A declaração e inicialização a seguir criam uma string que consiste na palavra 'Hello'. O tamanho da string é exatamente o número de caracteres da palavra 'Hello'.

```
Data my_Char(5) VALUE 'Hello'.
```

O programa a seguir é um exemplo de criação de strings.

```
REPORT YT_SEP_15.  
DATA my_Char(5) VALUE 'Hello'.  
Write my_Char.
```

O código acima produz a seguinte saída -

```
Hello
```

Comprimento da corda

Para encontrar o comprimento das strings de caracteres, podemos usar a instrução STRLEN . A função STRLEN() retorna o número de caracteres contidos na string.

Exemplo

```
REPORT YT_SEP_15.  
DATA: title_1(10) VALUE 'Tutorials',  
      length_1 TYPE I.  
  
length_1 = STRLEN( title_1 ).  
Write: / 'The Length of the Title is:', length_1.
```

O código acima produz a seguinte saída -

```
The Length of the Title is: 9
```

ABAP oferece suporte a uma ampla variedade de instruções que manipulam strings.

S. Não.	Declaração e Propósito
1	CONCATENAR Duas cordas são unidas para formar uma terceira corda.
2	CONDENSAR Esta instrução exclui os caracteres de espaço.
3	ESTRELA Usado para encontrar o comprimento de um campo.
4	SUBSTITUIR Usado para fazer substituições de caracteres.
5	PROCURAR Para executar pesquisas em sequências de caracteres.
6	MUDANÇA Usado para mover o conteúdo de uma string para a esquerda ou para a direita.
7	DIVIDIR Usado para dividir o conteúdo de um campo em dois ou mais campos.

O exemplo a seguir faz uso de algumas das declarações mencionadas acima -

Exemplo

```

REPORT YT_SEP_15.

DATA: title_1(10) VALUE 'Tutorials',
      title_2(10) VALUE 'Point',
      spaced_title(30) VALUE 'Tutorials Point Limited',
      sep,
      dest1(30),
      dest2(30).

CONCATENATE title_1 title_2 INTO dest1.

```

```

Write: / 'Concatenation:', dest1.

CONCATENATE title_1 title_2 INTO dest2 SEPARATED BY sep.

Write: / 'Concatenation with Space:', dest2.

CONDENSE spaced_title.

Write: / 'Condense with Gaps:', spaced_title.

CONDENSE spaced_title NO-GAPS.

Write: / 'Condense with No Gaps:', spaced_title.

```

O código acima produz a seguinte saída -

```

Concatenation: TutorialsPoint
Concatenation with Space: Tutorials Point
Condense with Gaps: Tutorials Point Limited
Condense with No Gaps: TutorialsPointLimited

```

Nota -

- No caso de Concatenação, o 'sep' insere um espaço entre os campos.
- A instrução CONDENSE remove espaços em branco entre os campos, mas deixando apenas 1 caractere de espaço.
- 'NO-GAPS' é uma adição opcional à instrução CONDENSE que remove todos os espaços.

SAP ABAP - Data e Hora

ABAP faz referência implicitamente ao calendário gregoriano, válido na maior parte do mundo. Podemos converter a saída em calendários específicos de cada país. Uma data é uma hora especificada para um dia, semana ou mês preciso em relação a um calendário. Um tempo é especificado em um segundo ou minuto preciso em relação a um dia. ABAP sempre economiza tempo no formato de 24 horas. A saída pode ter um formato específico do país. As datas e a hora geralmente são interpretadas como datas locais válidas no fuso horário atual.

ABAP fornece dois tipos integrados para trabalhar com datas e horas -

- Tipo de dados D
- Tipo de dados T

A seguir está o formato básico -

```

DATA: date TYPE D,
      time TYPE T.

```

```
DATA: year TYPE I,
month TYPE I,
day TYPE I,
hour TYPE I,
minute TYPE I,
second TYPE I.
```

Ambos os tipos são tipos de caracteres de comprimento fixo que possuem o formato AAAAMMDD e HHMMSS, respectivamente.

Carimbos de data e hora

Além desses tipos integrados, os outros dois tipos **TIMESTAMP** e **TIMESTAMPL** estão sendo usados em muitas tabelas de aplicativos padrão para armazenar um carimbo de data/hora no formato UTC. A tabela a seguir mostra os tipos básicos de data e hora disponíveis no ABAP.

S. Não.	Tipo e descrição de dados
1	D Um tipo de data interno de comprimento fixo no formato AAAAMMDD. Por exemplo, o valor 20100913 representa a data 13 de setembro de 2010.
2	T Um tipo de tempo de duração fixa integrado no formato HHMMSS. Por exemplo, o valor 102305 representa o horário 10:23:05.
3	TIMESTAMP (Tipo P – Comprimento 8 Sem decimais) Este tipo é usado para representar carimbos de data/hora curtos no formato AAAAMMDDhhmmss. Por exemplo, o valor 20100913102305 representa a data 13 de setembro de 2010 às 10h23min05s.
4	TIMESTAMPL (Tipo P - Comprimento 11 Decimais 7) TIMESTAMPL representa carimbos de data/hora longos no formato AAAAMMDDhhmmss,mmmuuun. Aqui, os dígitos adicionais 'mmmuuun' representam as frações de segundo.

Data e hora atuais

Os trechos de código a seguir recuperam a data e hora atuais do sistema.

```
REPORT YR_SEP_15.
DATA: date_1 TYPE D.

date_1 = SY-DATUM.
Write: / 'Present Date is:', date_1 DD/MM/YYYY.

date_1 = date_1 + 06.
Write: / 'Date after 6 Days is:', date_1 DD/MM/YYYY.
```

O código acima produz a seguinte saída -

```
Present Date is: 21.09.2015
Date after 6 Days is: 27.09.2015
```

A variável date_1 recebe o valor da data atual do sistema SY-DATUM. A seguir, incrementamos o valor da data em 6. Em termos de cálculo de data no ABAP, isso implica que estamos aumentando o componente de dia do objeto de data em 6 dias. O ambiente de execução ABAP é inteligente o suficiente para transferir o valor da data sempre que chega ao final de um mês.

Os cálculos de tempo funcionam de forma semelhante aos cálculos de data. O código a seguir aumenta a hora atual do sistema em 75 segundos usando aritmética básica de hora.

```
REPORT YR_SEP_15.
DATA: time_1 TYPE T.
      time_1 = SY-UZEIT.

Write /(60) time_1 USING EDIT MASK
'Now the Time is: __:__:__'.
time_1 = time_1 + 75.

Write /(60) time_1 USING EDIT MASK
'A Minute and a Quarter from Now, it is: __:__:__'.
```

O código acima produz a seguinte saída -

```
Now the Time is 11:45:05
A Minute and a Quarter from Now, it is: 11:46:20
```

Trabalhando com carimbos de data/hora

Você pode recuperar a hora atual do sistema e armazená-la em uma variável timestamp usando **GET TIME STAMP** conforme mostrado no código a seguir. A instrução GET TIME STAMP armazena o carimbo de data/hora em um formato extenso ou abreviado de acordo com o tipo de objeto de dados de carimbo de data/hora usado. O valor do carimbo de data/hora é codificado usando o padrão UTC.

```
REPORT YR_SEP_12.  
DATA: stamp_1 TYPE TIMESTAMP,  
  
stamp_2 TYPE TIMESTAMPL.  
GET TIME STAMP FIELD stamp_1.  
Write: / 'The short time stamp is:', stamp_1  
  
TIME ZONE SY-ZONLO.  
GET TIME STAMP FIELD stamp_2.  
Write: / 'The long time stamp is:', stamp_2  
TIME ZONE SY-ZONLO.
```

O código acima produz a seguinte saída -

```
The short time stamp is: 18.09.2015 11:19:40  
The long time stamp is: 18.09.2015 11:19:40,9370000
```

No exemplo acima, estamos exibindo o carimbo de data/hora usando a adição TIME ZONE da instrução WRITE. Esta adição formata a saída do carimbo de data/hora de acordo com as regras do fuso horário especificado. O campo do sistema SY-ZONLO é utilizado para exibir o fuso horário local configurado nas preferências do usuário.

SAP ABAP - Formatando Dados

ABAP oferece vários tipos de opções de formatação para formatar a saída dos programas. Por exemplo, você pode criar uma lista que inclua vários itens em cores ou estilos de formatação diferentes.

A instrução WRITE é uma instrução de formatação usada para exibir dados em uma tela. Existem diferentes opções de formatação para a instrução WRITE. A sintaxe da instrução WRITE é -

```
WRITE <format> <f> <options>.
```

Nesta sintaxe, <format> representa a especificação do formato de saída, que pode ser uma barra (/) que indica a exibição da saída começando em uma nova linha. Além da barra, a especificação do formato inclui um número de coluna e um comprimento de coluna. Por exemplo, a instrução WRITE/04 (6) mostra que uma nova linha começa com a coluna 4 e o comprimento da coluna é 6, enquanto a instrução WRITE 20 mostra a linha atual com a coluna 20. O parâmetro <f> representa uma variável de dados ou texto numerado.

A tabela a seguir descreve várias cláusulas usadas para formatação -

S. Não.	Cláusula e Descrição
1	JUSTIFICADO À ESQUERDA Especifica que a saída é justificada à esquerda.
2	CENTRADO Denota que a saída está centralizada.
3	JUSTIFICADO CORRETAMENTE Especifica que a saída é justificada à direita.
4	SOB <g> A saída começa diretamente no campo <g>.
5	NENHUMA LACUNA Especifica que o espaço em branco após o campo <f> será rejeitado.
6	USANDO MÁSCARA DE EDIÇÃO <m> Denota a especificação do modelo de formato <m>. Usando No EDIT Mask: Especifica que o modelo de formato especificado no Dicionário ABAP está desativado.
7	SEM ZERO Se um campo contiver apenas zeros, eles serão substituídos por espaços em branco.

A seguir estão as opções de formatação para campos de tipo numérico -

S. Não.	Cláusula e Descrição
1	SEM SINAL Especifica que nenhum sinal inicial é exibido na tela.
2	EXPONENTE <e> Especifica que no tipo F (campos de ponto flutuante), o expoente é definido em <e>.
3	RODADA <r> Os campos do tipo P (tipos de dados numéricos compactados) são primeiro multiplicados por $10^{**}(-r)$ e depois arredondados para um valor inteiro.
4	MOEDA <c> Indica que a formatação é feita de acordo com o valor da moeda <c> que está armazenado na tabela do banco de dados TCURX.
5	UNIDADE Especifica que o número de casas decimais é fixado de acordo com a unidade <u> conforme especificado na tabela do banco de dados T006 para o tipo P.
6	DECIMAIS <d> Especifica que o número de dígitos <d> deve ser exibido após a vírgula decimal.

Por exemplo, a tabela a seguir mostra diferentes opções de formatação para os campos de data

-

Opção de formatação	Exemplo
DD/MM/AA	13/01/15
MM/DD/AA	13/01/15
DD/MM/AAAA	13/01/2015
MM/DD/AAAA	13/01/2015
DDMMAA	130115
MMDDAA	011315
AAMMDD	150113

Aqui, DD representa a data com dois algarismos, MM representa o mês com dois algarismos, AA representa o ano com dois algarismos e AAAA representa o ano com quatro algarismos.

Vamos dar uma olhada em um exemplo de código ABAP que implementa algumas das opções de formatação acima -

```
REPORT ZTest123_01.
```

```
DATA: n(9) TYPE C VALUE 'Tutorials',
m(5) TYPE C VALUE 'Point'.
```

```
WRITE: n, m.
WRITE: / n,
/ m UNDER n.
```

```
WRITE: / n NO-GAP, m.
DATA time TYPE T VALUE '112538'.
```

```
WRITE: / time,
/(8) time Using EDIT MASK '__:__:_'.
```

O código acima produz a seguinte saída -

```
Tutorials Point
Tutorials
Point
TutorialsPoint
112538
11:25:38
```

SAP ABAP - Tratamento de Exceções

Uma **exceção** é um problema que surge durante a execução de um programa. Quando ocorre uma exceção, o fluxo normal do programa é interrompido e a aplicação do programa termina de forma anormal, o que não é recomendado, portanto, essas exceções devem ser tratadas.

As exceções fornecem uma maneira de transferir o controle de uma parte de um programa para outra. O tratamento de exceções ABAP é baseado em três palavras-chave - RAISE, TRY, CATCH e CLEANUP. Supondo que um bloco irá gerar uma exceção, um método captura uma exceção usando uma combinação das palavras-chave TRY e CATCH. Um bloco TRY - CATCH é colocado ao redor do código que pode gerar uma exceção. A seguir está a sintaxe para usar TRY - CATCH -

TRY.

Try Block <Code that raises an exception>

CATCH

Catch Block <exception handler M>

...

...

...

CATCH

Catch Block <exception handler R>

CLEANUP.

Cleanup block <to restore consistent state>

ENDTRY.

RAISE - Exceções são levantadas para indicar que ocorreu alguma situação excepcional. Normalmente, um manipulador de exceções tenta reparar o erro ou encontrar uma solução alternativa.

TRY - O bloco TRY contém a codificação da aplicação cujas exceções devem ser tratadas. Este bloco de instruções é processado sequencialmente. Pode conter outras estruturas de controle e chamadas de procedimentos ou outros programas ABAP. É seguido por um ou mais blocos catch.

CATCH - Um programa captura uma exceção com um manipulador de exceção no local do programa onde você deseja tratar o problema. A palavra-chave CATCH indica a captura de uma exceção.

CLEANUP - As instruções do bloco CLEANUP são executadas sempre que ocorre uma exceção em um bloco TRY que não é capturado pelo manipulador da mesma construção TRY - ENDTRY. Dentro da cláusula CLEANUP, o sistema pode restaurar um objeto para um estado consiste

ou liberar recursos externos. Ou seja, o trabalho de limpeza pode ser executado para o contexto do bloco TRY.

Levantando exceções

Exceções podem ser levantadas em qualquer ponto de um método, módulo de função, subrotina e assim por diante. Existem duas maneiras de gerar uma exceção -

- Exceções levantadas pelo sistema de tempo de execução ABAP.

Por exemplo $Y = 1/0$. Isso resultará em um erro de tempo de execução do tipo CX_SY_ZERODIVIDE.

- Exceções levantadas pelo programador.

Levante e crie um objeto de exceção simultaneamente. Gere uma exceção com um objeto de exceção que já existe no primeiro cenário. A sintaxe é: RAISE EXCEPTION exep.

Capturando exceções

Manipuladores são usados para capturar exceções.

Vamos dar uma olhada em um trecho de código -

```
DATA: result TYPE P LENGTH 8 DECIMALS 2,
      exref TYPE REF TO CX_ROOT,
      msgtxt TYPE STRING.

PARAMETERS: Num1 TYPE I, Num2 TYPE I.

TRY.

  result = Num1 / Num2.

  CATCH CX_SY_ZERODIVIDE INTO exref.
    msgtxt = exref->GET_TEXT( ).

  CATCH CX_SY_CONVERSION_NO_NUMBER INTO exref.
    msgtxt = exref->GET_TEXT( ).
```

No trecho de código acima, estamos tentando dividir Num1 por Num2 para obter o resultado em uma variável do tipo float.

Dois tipos de exceções podem ser gerados.

- Erro de conversão de número.
- Divida por zero exceção. Os manipuladores capturaram a exceção CX_SY_CONVERSION_NO_NUMBER e também a exceção CX_SY_ZERODIVIDE. Aqui o método GET_TEXT() da classe de exceção é usado para obter a descrição da exceção.

Atributos de exceções

Aqui estão os cinco atributos e métodos de exceções -

S. Não.	Atributo e descrição
1	Texto Utilizado para definir diferentes textos para exceções e também afeta o resultado do método get_text.
2	Anterior Este atributo pode armazenar a exceção original que permite construir uma cadeia de exceções.
3	obter_texto Isso retorna a representação textual como uma string de acordo com o idioma do sistema da exceção.
4	get_longtext Isso retorna a variante longa da representação textual da exceção como uma string.
5	get_source_position Fornece o nome do programa e o número da linha alcançada onde a exceção foi gerada.

Exemplo

```

REPORT ZExceptionsDemo.

PARAMETERS Num_1 TYPE I.

DATA res_1 TYPE P DECIMALS 2.
DATA orf_1 TYPE REF TO CX_ROOT.
DATA txt_1 TYPE STRING.

start-of-selection.
Write: / 'Square Root and Division with:', Num_1.
write: /.

```

```

TRY.
IF ABS( Num_1 ) > 150.
RAISE EXCEPTION TYPE CX_DEMO_ABS_TOO_LARGE.
ENDIF.

TRY.
res_1 = SQRT( Num_1 ).
Write: / 'Result of square root:', res_1.
res_1 = 1 / Num_1.

Write: / 'Result of division:', res_1.
CATCH CX_SY_ZERODIVIDE INTO orf_1.
txt_1 = orf_1→GET_TEXT( ).
CLEANUP.
CLEAR res_1.
ENDTRY.

CATCH CX_SY_ARITHMETIC_ERROR INTO orf_1.
txt_1 = orf_1→GET_TEXT( ).

CATCH CX_ROOT INTO orf_1.
txt_1 = orf_1→GET_TEXT( ).
ENDTRY.

IF NOT txt_1 IS INITIAL.
Write / txt_1.
ENDIF.
Write: / 'Final Result is:', res_1.

```

Neste exemplo, se o número for maior que 150, a exceção CX_DEMO_ABS_TOO_LARGE é levantada. O código acima produz a seguinte saída para o número 160.

```

Square Root and Division with: 160
The absolute value of number is too high
Final Result is: 0.00

```

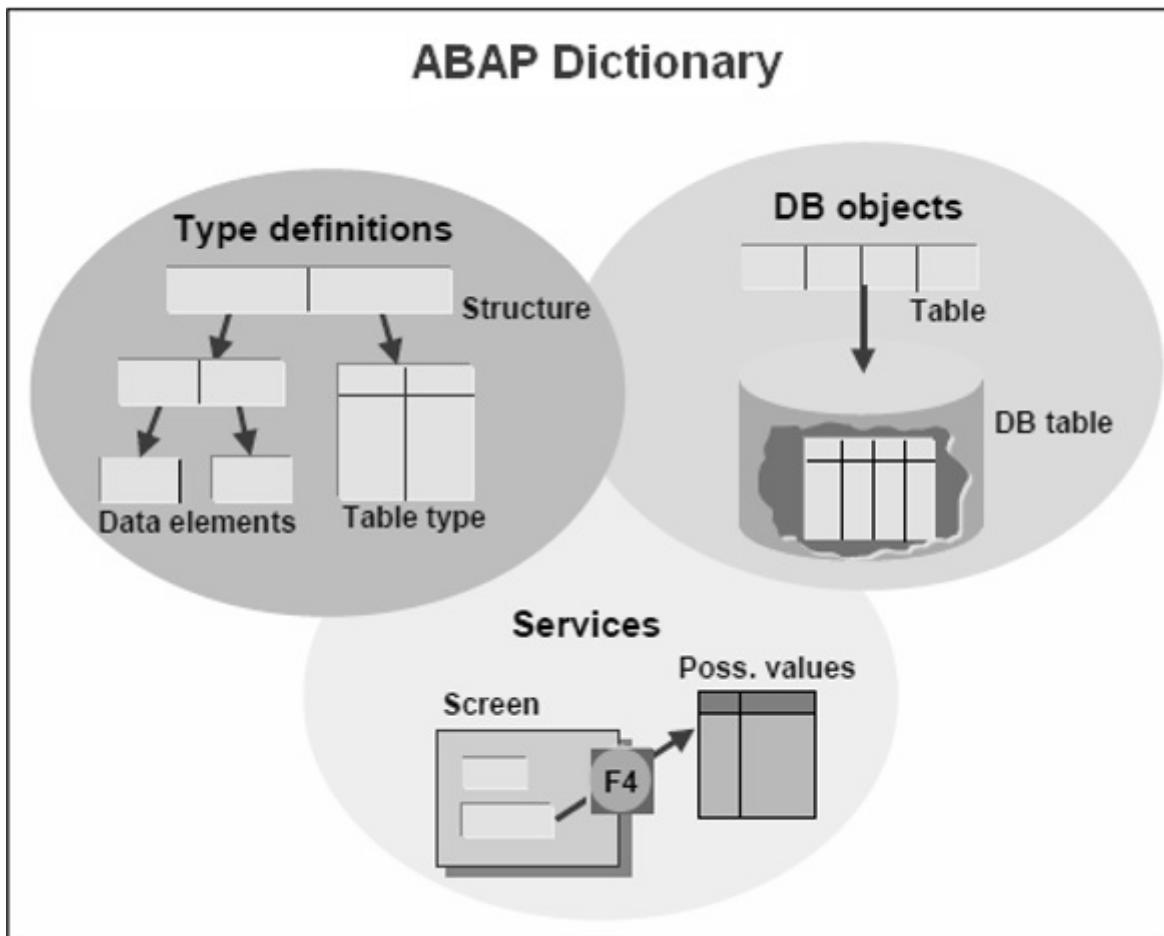
SAP ABAP - Dicionário

Como você sabe, o SQL pode ser dividido em duas partes -

- DML (linguagem de manipulação de dados)
- DDL (linguagem de definição de dados)

A parte DML consiste em comandos de consulta e atualização, como SELECT, INSERT, UPDATE, DELETE, etc. e os programas ABAP lidam com a parte DML do SQL. A parte DDL consiste

comandos como CREATE TABLE, CREATE INDEX, DROP TABLE, ALTER TABLE, etc. e o ABAP Dictionary lida com a parte DDL do SQL.



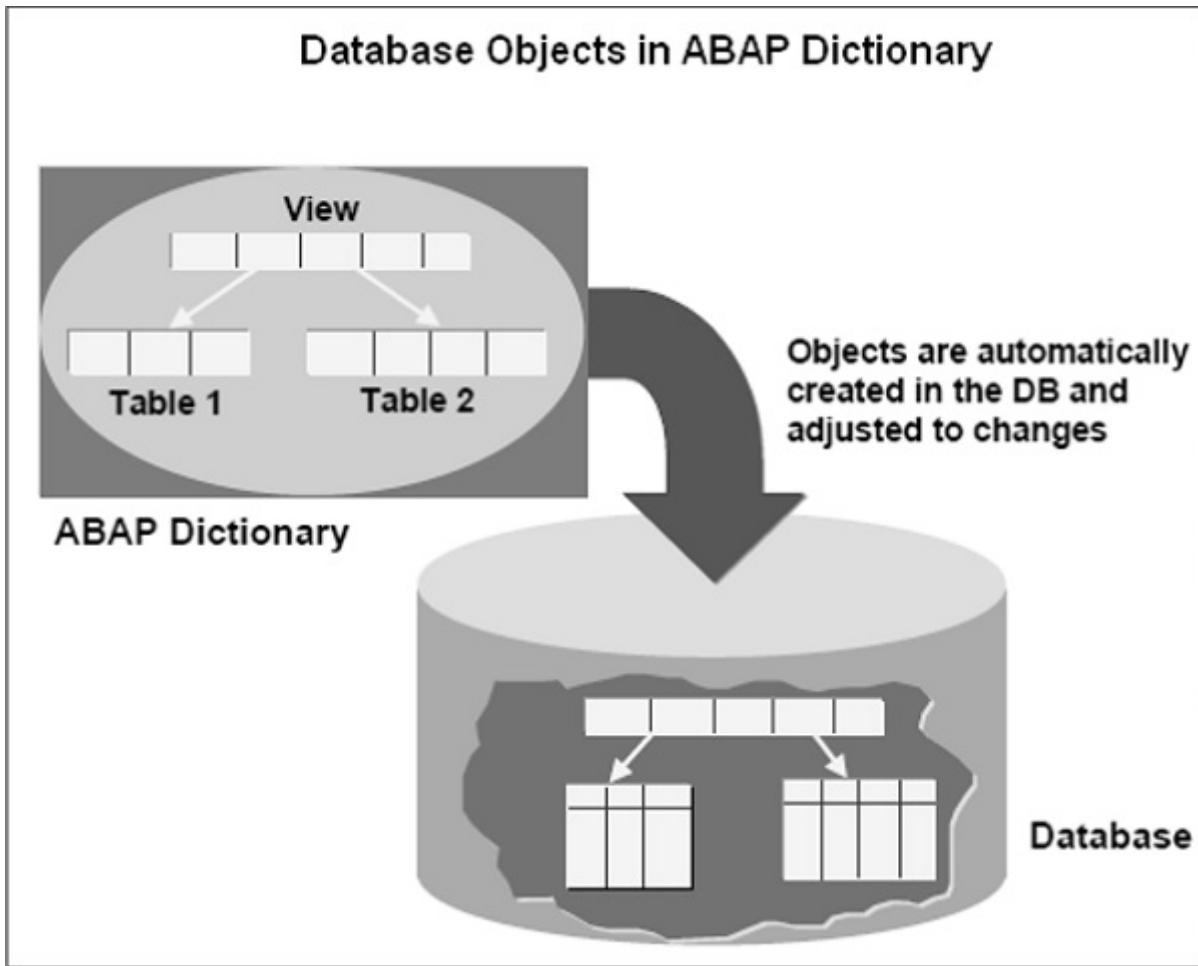
O Dicionário ABAP pode ser visto como metadados (ou seja, dados sobre dados) que residem no banco de dados SAP junto com os metadados mantidos pelo banco de dados. O Dicionário é usado para criar e gerenciar definições de dados e para criar Tabelas, Elementos de Dados, Domínios, Visualizações e Tipos.

Tipos básicos no dicionário ABAP

Os tipos básicos no Dicionário ABAP são os seguintes -

- **Os elementos de dados** descrevem um tipo elementar definindo o tipo de dados, comprimento e possivelmente casas decimais.
- **Estruturas** com componentes que podem ser de qualquer tipo.
- **Os tipos de tabela** descrevem a estrutura de uma tabela interna.

Vários objetos no ambiente Dicionário podem ser referenciados em programas ABAP. O Dicionário é conhecido como área global. Os objetos no Dicionário são globais para todos os programas ABAP e os dados nos programas ABAP podem ser declarados por referência a esses objetos globais do Dicionário.



O Dicionário suporta a definição de tipos definidos pelo usuário e esses tipos são usados em programas ABAP. Eles também definem a estrutura dos objetos do banco de dados, como tabelas, visualizações e índices. Esses objetos são criados automaticamente no banco de dados subjacente em suas definições de Dicionário quando os objetos são ativados. O Dicionário também fornece ferramentas de edição como Search Help e ferramentas de bloqueio como Lock Objects.

Tarefas de Dicionário

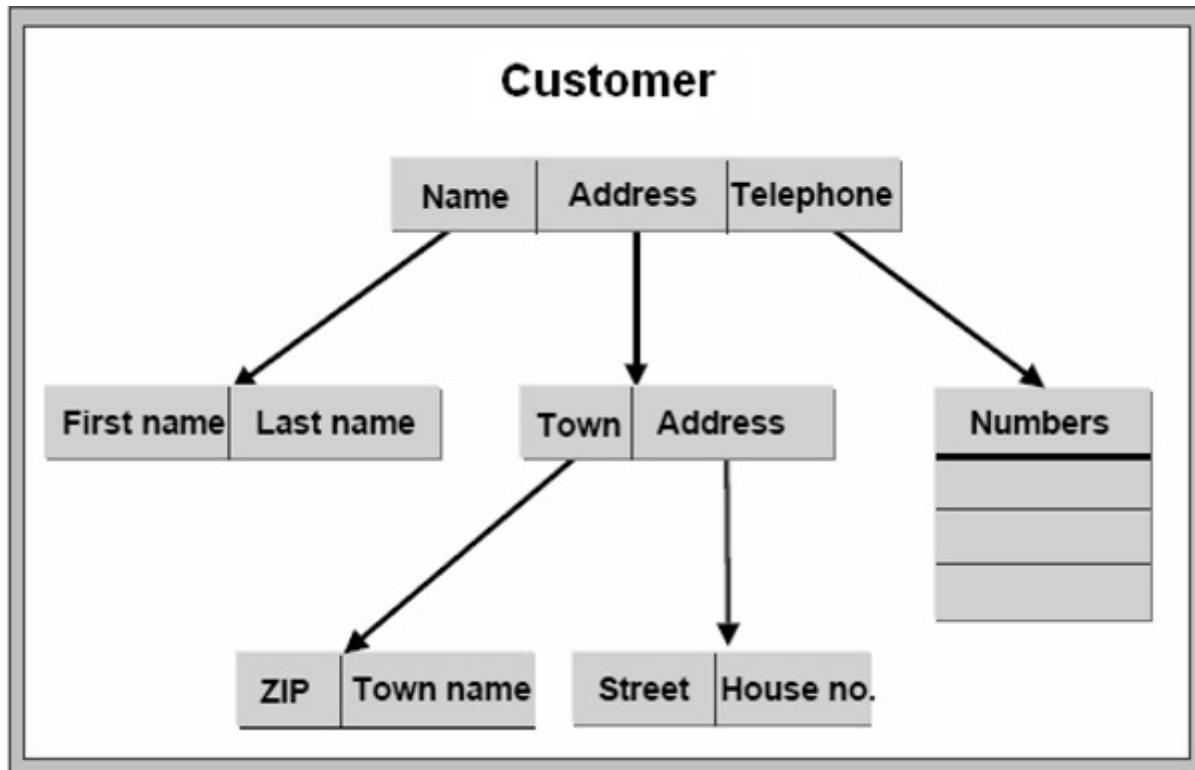
O Dicionário ABAP alcança o seguinte -

- Aplica a integridade dos dados.
 - Gerencia definições de dados sem redundância.
- Integra-se perfeitamente com o restante do ambiente de trabalho de desenvolvimento ABAP.

Exemplo

Qualquer tipo complexo definido pelo usuário pode ser construído a partir dos 3 tipos básicos do Dicionário. Os dados do cliente são armazenados numa estrutura 'Cliente' com os componentes Nome, Morada e Telefone conforme imagem seguinte. O nome também é uma

estrutura com componentes, Nome e Sobrenome. Ambos os componentes são elementares porque seu tipo é definido por um elemento de dados.



O tipo do componente Endereço é definido por uma estrutura cujos componentes também são estruturas, e o componente Telefone é definido por um tipo tabela porque um cliente pode ter mais de um número de telefone. Os tipos são usados em programas ABAP e também para definir os tipos de parâmetros de interface dos módulos de função.

SAP ABAP - Domínios

Os três objetos básicos para definição de dados no Dicionário ABAP são Domínios, Elementos de Dados e Tabelas. O domínio é usado para a definição técnica de um campo da tabela, como tipo e comprimento do campo, e o elemento de dados é usado para a definição semântica (breve descrição). Um elemento de dados descreve o significado de um domínio em um determinado contexto comercial. Ele contém principalmente a ajuda dos campos e os rótulos dos campos na tela.

O domínio é atribuído ao elemento de dados, que por sua vez é atribuído aos campos da tabela ou aos campos da estrutura. Por exemplo, o domínio MATNR (número de material CHAR) é atribuído a elementos de dados como MATNR_N, MATNN e MATNR_D, e estes são atribuídos a muitos campos de tabela e campos de estrutura.

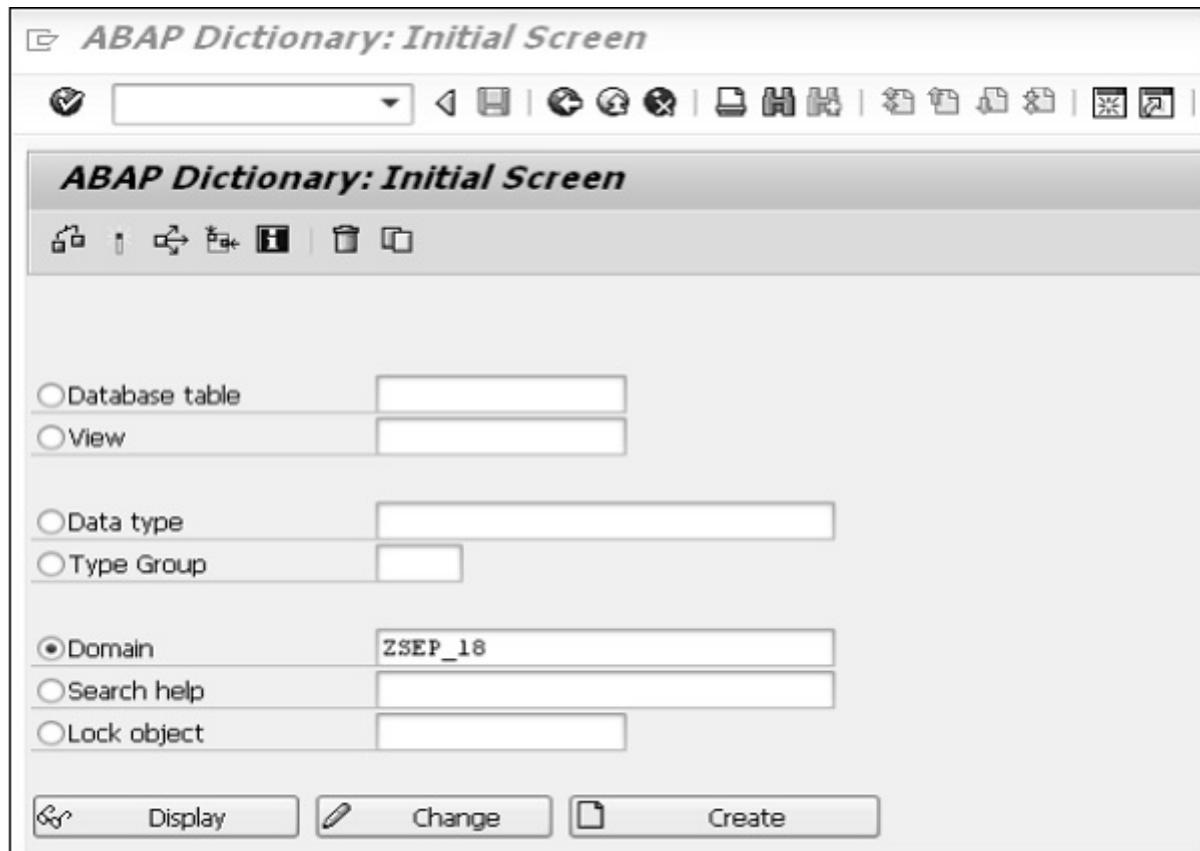
Criando Domínios

Antes de criar um novo domínio, verifique se algum domínio existente possui as mesmas especificações técnicas exigidas no campo da sua tabela. Nesse caso, devemos usar e

domínio existente. Vamos discutir o procedimento para criar o domínio.

Step 1 - Vá para a transação SE11.

Step 2 - Selecione o botão de opção Domínio na tela inicial do Dicionário ABAP e insira o nome do domínio conforme mostrado na imagem a seguir. Clique no botão CRIAR. Você pode criar domínios nos namespaces do cliente e o nome do objeto sempre começa com 'Z' ou 'Y'.



Step 3 – Enter the description in the short text field of the maintenance screen of the domain. In this case, it is “Customer Domain”. **Note** – You cannot enter any other attribute until you have entered this attribute.

Step 4 – Enter the Data Type, No. of Characters, and Decimal Places in the Format block of the Definition tab. Press the key on Output Length and it proposes and displays the output length. If you overwrite the proposed output length, you may see a warning while activating the domain. You may fill in the Convers. Routine, Sign and Lower Case fields if required. But these are always optional attributes.

Step 5 – Select the Value Range tab. If the domain is restricted to having only fixed values then enter the fixed values or intervals. Define the value table if the system has to propose this table as a check table while defining a foreign key for the fields referring to this domain. But all these are optional attributes.

Dictionary: Change Domain

Domain	ZSEP_18	New(Revised)
Short Description	Customer Domain	

Properties Definition Value Range

Format

Data Type	NUMC	Character string with only digits
No. Characters	8	
Decimal Places	0	

Output Characteristics

Output Length	8
Convers. Routine	
<input type="checkbox"/> Sign	
<input type="checkbox"/> Lower Case	

Step 6 – Save your changes. The Create Object Directory Entry pop-up appears and asks for a package. You may enter the package name in which you are working. If you do not have any package then you may create it in the Object Navigator or you can save your domain using the Local Object button.

Step 7 - Ative seu domínio. Clique no ícone Ativar (ícone de palito de fósforo) ou pressione CTRL + F3 para ativar o domínio. Uma janela pop-up aparece listando os 2 objetos atualmente inativos, conforme mostrado no instantâneo a seguir -

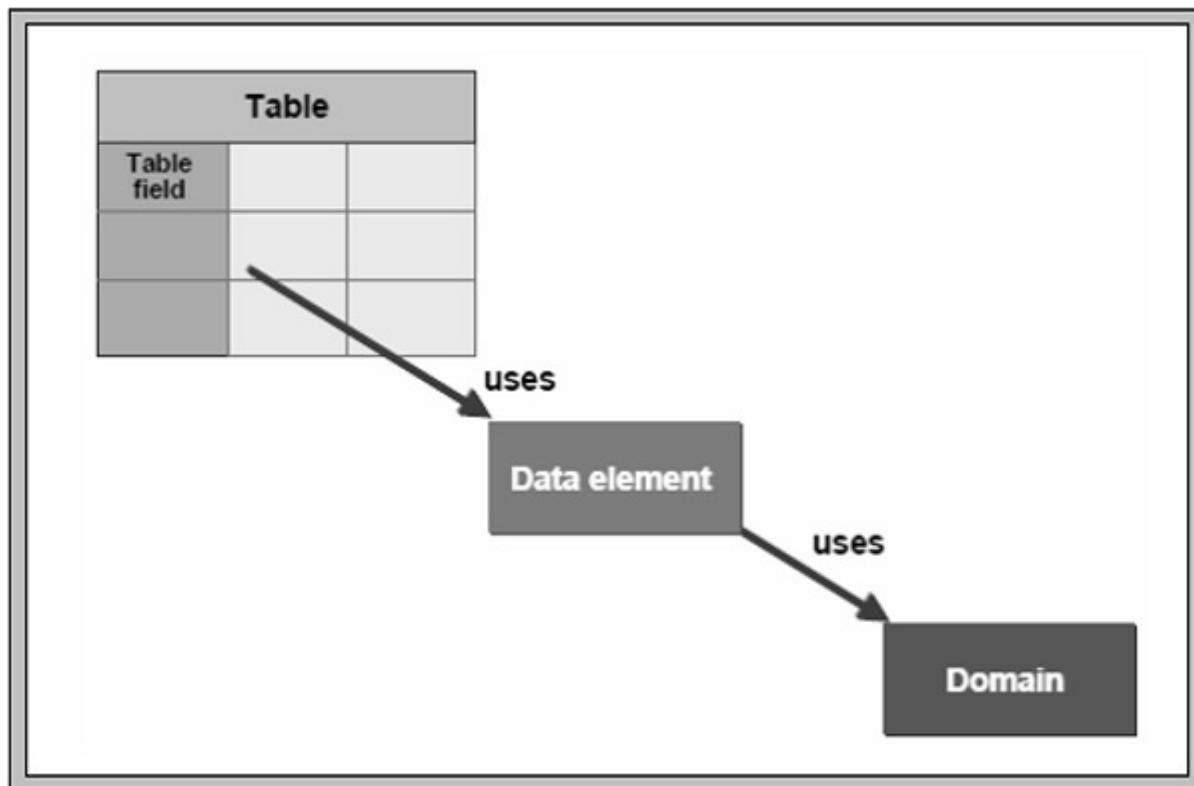
Object name		
D..	Object	Obj. name
	DOMA	ZSEP_18
	DOMA	Z_SEP_21

Step 8 - Neste ponto, a entrada superior denominada 'DOMA' com o nome ZSEP_18 deve ser ativada. Como isso está destacado, clique no botão verde. Esta janela desaparece e a barra de status exibirá a mensagem 'Objeto ativado'.

Se mensagens de erro ou avisos ocorrerem quando você ativou o domínio, o log de ativação será exibido automaticamente. O log de ativação exibe informações sobre o fluxo de ativação. Você também pode acessar o log de ativação com Utilities(M) → Activation log.

SAP ABAP - Elementos de dados

Os elementos de dados descrevem os campos individuais no ABAP Data Dictionary. São as menores unidades indivisíveis dos tipos complexos e são utilizadas para definir o tipo de campo da tabela, componente de estrutura ou tipo de linha de uma tabela. Informações sobre o significado de um campo de tabela e também informações sobre a edição do campo de tela correspondente podem ser atribuídas a um elemento de dados. Esta informação fica disponível automaticamente para todos os campos da tela referentes ao elemento de dado. Os elementos de dados descrevem tipos elementares ou tipos de referência.



Criando Elementos de Dados

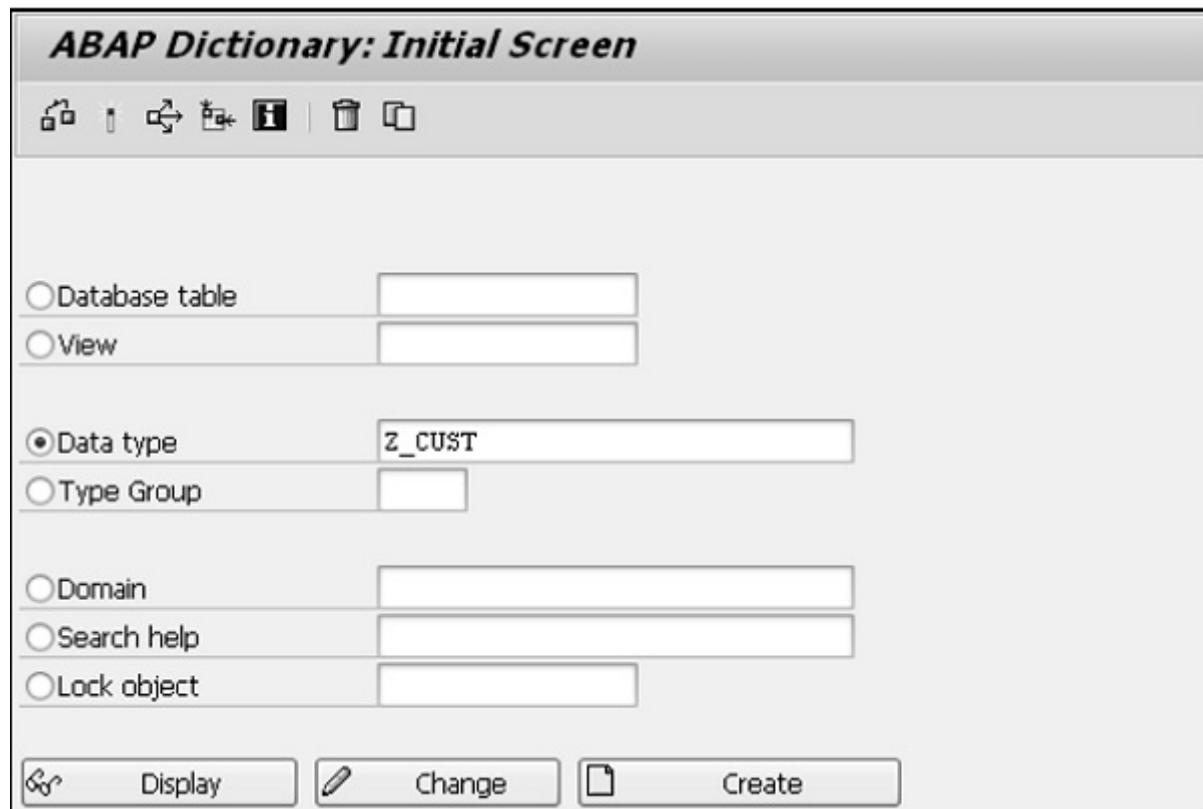
Antes de criar um novo elemento de dados, você precisa verificar se algum elemento de dados existente tem as mesmas especificações semânticas exigidas no campo da tabela. Nesse caso, você poderá usar esse elemento de dados existente. Você pode atribuir ao elemento de dados um tipo, domínio ou tipo de referência predefinido.

A seguir está o procedimento para criar o elemento de dados -

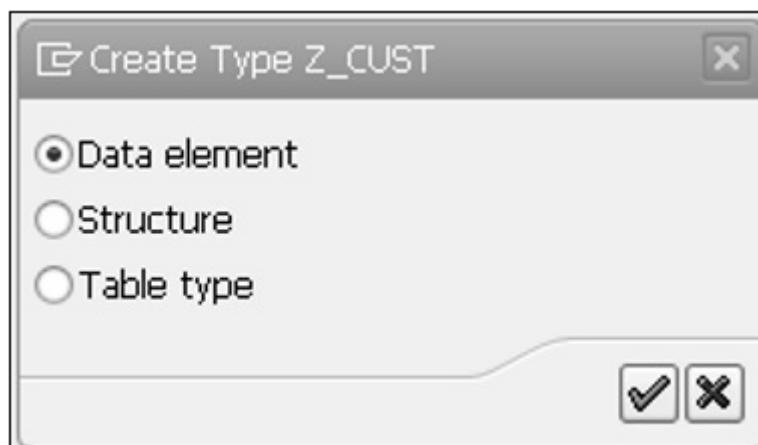
Step 1 - Vá para a transação SE11.

Step 2 - Selecione o botão de opção Tipo de dados na tela inicial do Dicionário ABAP e insira o nome do elemento de dados conforme mostrado abaixo.

Step 3 - Clique no botão CRIAR. Você pode criar elementos de dados nos namespaces do cliente, e o nome do objeto sempre começa com 'Z' ou 'Y'.



Step 4 - Verifique o botão de opção Elemento de dados no pop-up CREATE TYPE que aparece com três botões de opção.



Step 5 - Clique no ícone da marca de seleção verde. Você é direcionado para a tela de manutenção do elemento de dados.

Step 6 - Insira a descrição no campo de texto curto da tela de manutenção do elemento de dados. Neste caso, é “Elemento de Dados do Cliente”. **Note** - Você não pode inserir nenhum outro atributo até inserir este atributo.

Dictionary: Change Data Element

Data element	Z_CUST	New(Revised)
Short Description	Customer Data Element	
<input checked="" type="radio"/> Elementary Type <input type="radio"/> Domain ZSEP_18 Customer Domain Data Type NUMC Character string with only digits Length 8		

Step 7 - Atribua o elemento de dados ao tipo. Você pode criar um elemento de dados elementar verificando o tipo elementar ou um elemento de dados de referência verificando o tipo de referência. Você pode atribuir um elemento de dados a um Domínio ou Tipo Predefinido dentro do Tipo Elementar e com Nome do Tipo de Referência ou Referência ao Tipo Predefinido dentro do Tipo de Referência.

Step 8 - Insira os campos para texto curto, texto médio, texto longo e cabeçalho na guia Etiqueta do campo. Você pode pressionar Enter e o comprimento será gerado automaticamente para essas etiquetas.

Dictionary: Change Data Element

Data element	Z_CUST	New(Revised)
Short Description	Customer Data Element	
<input checked="" type="radio"/> Attributes <input type="radio"/> Data Type <input type="radio"/> Further Characteristics <input type="radio"/> Field Label		
Length	Field Label	
Short	10	Customer M
Medium	15	Customer Number
Long	20	Customer Number
Heading	15	Customer Number

Step 9 - Salve suas alterações. O pop-up Criar entrada de diretório de objetos aparece e solicita um pacote. Você pode inserir o nome do pacote no qual está trabalhando. Se você não possui nenhum pacote, você pode criá-lo no Object Navigator ou pode salvar seu elemento de dados usando o botão Local Object.

Step 10 - Ative seu elemento de dados. Clique no ícone Ativar (ícone de palito de fósforo) ou pressione CTRL + F3 para ativar o elemento de dados. Uma janela pop-up aparece listando os 2 objetos atualmente inativos, conforme mostrado na captura de tela a seguir.

Object name		
D..	Object	Obj. name
	DTEL	Z_CUST
	DOMA	Z_SEP_21

Step 11 - Neste ponto, a entrada superior denominada 'DTEL' com o nome Z_CUST deve ser ativada. Como isso está destacado, clique no botão verde. Esta janela desaparece e a barra de status exibirá a mensagem 'Objeto ativado'.

Se mensagens de erro ou avisos ocorrerem quando você ativou o elemento de dados, o log de ativação será exibido automaticamente. O log de ativação exibe informações sobre o fluxo de ativação. Você também pode acessar o log de ativação com Utilities(M) → Activation log.

SAP ABAP - Tabelas

As tabelas podem ser definidas independentemente do banco de dados no Dicionário ABAP. Quando uma tabela é ativada no Dicionário ABAP, uma cópia semelhante de seus campos também é criada no banco de dados. As tabelas definidas no Dicionário ABAP são traduzidas automaticamente para o formato compatível com o banco de dados, pois a definição da tabela depende do banco de dados utilizado pelo sistema SAP.

Uma tabela pode conter um ou mais campos, cada um definido com seu tipo e comprimento de dados. A grande quantidade de dados armazenados em uma tabela é distribuída entre os diversos campos definidos na tabela.

Tipos de campos de tabela

Uma tabela consiste em muitos campos e cada campo contém muitos elementos. A tabela a seguir lista os diferentes elementos dos campos da tabela -

S. Não.	Elementos e Descrição
1	<p>Nome do campo</p> <p>Este é o nome dado a um campo que pode conter no máximo 16 caracteres. O nome do campo pode ser composto por dígitos, letras e sublinhados. Deve começar com uma carta.</p>
2	<p>Bandeira chave</p> <p>Determina se um campo pertence ou não a um campo-chave.</p>
3	<p>Tipo de campo</p> <p>Atribui um tipo de dados a um campo.</p>
4	<p>Comprimento do campo</p> <p>O número de caracteres que podem ser inseridos em um campo.</p>
5	<p>Casas decimais</p> <p>Define o número de dígitos permitidos após a vírgula decimal. Este elemento é usado apenas para tipos de dados numéricos.</p>
6	<p>Pequeno texto</p> <p>Descreve o significado do campo correspondente.</p>

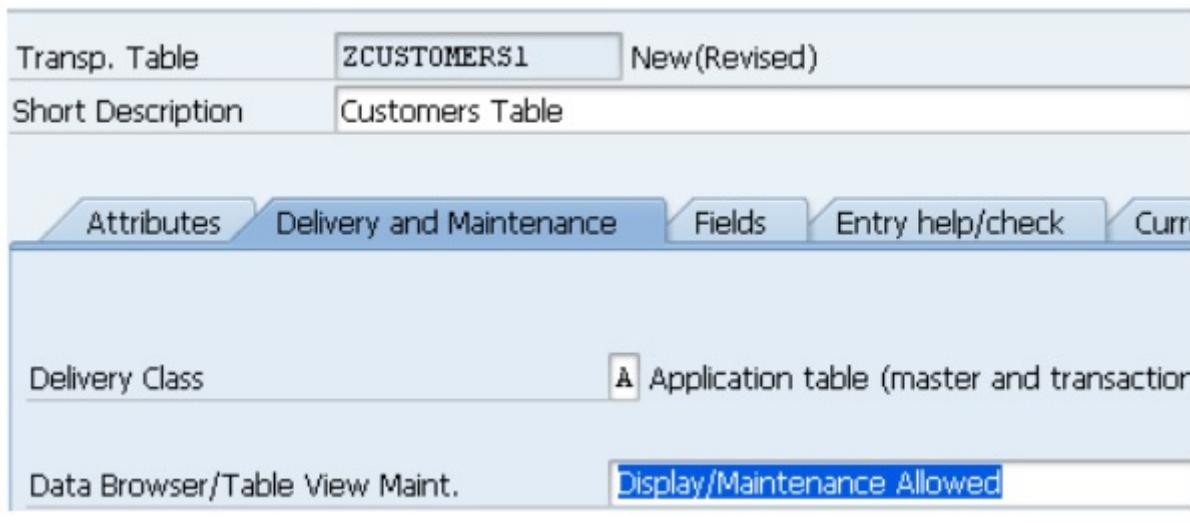
Criando tabelas no dicionário ABAP

Step 1 - Vá para a transação SE11, selecione o botão de opção 'Tabela de banco de dados' e insira um nome para a tabela a ser criada. No nosso caso, inserimos o nome ZCUSTOMERS1. Clique no botão Criar. A tela Dicionário: Manter tabela é exibida. Aqui a guia 'Entrega e Manutenção' é selecionada por padrão.

Step 2 - Insira um breve texto explicativo no campo Breve descrição.

Step 3 - Clique no ícone Pesquisar Ajuda ao lado do campo Classe de Entrega. Selecione a opção 'Uma [tabela de aplicação (dados mestre e de transação)]'.

Step 4 - Selecione a opção 'Exibição/Manutenção permitida' no menu suspenso 'Navegador de dados/Manutenção de visualização de tabela'. A tela Dicionário: Tabela de manutenção é exibida.



Step 5 - Selecione a guia Campos. Aparece a tela contendo as opções relacionadas à aba Campos.

Step 6 - Insira os nomes dos campos da tabela na coluna Campo. Um nome de campo pode conter letras, dígitos e sublinhados, mas deve sempre começar com uma letra e não deve ter mais de 16 caracteres.

Os campos a serem criados também devem possuir elementos de dados porque utilizam os atributos, como tipo de dados, comprimento, casas decimais e texto curto, do elemento de dados definido.

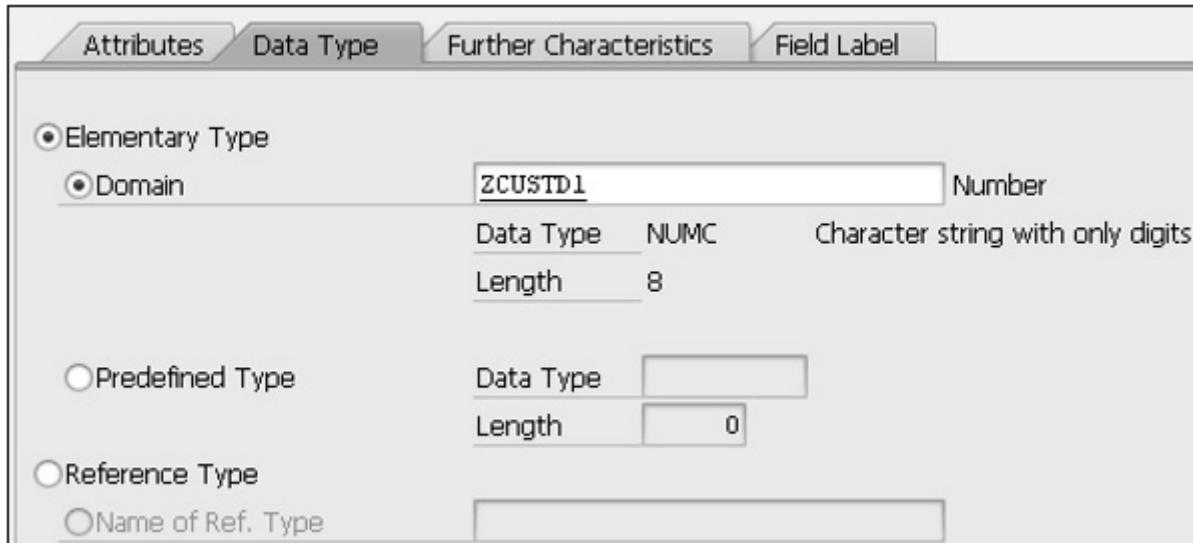
Step 7 - Selecione a coluna Chave se desejar que o campo faça parte da chave da tabela. Vamos criar campos como CLIENTE, CLIENTE, NOME, TÍTULO e DOB.

Step 8 - O primeiro campo é importante e identifica o cliente ao qual os registros estão associados. Insira 'Cliente' como campo e 'MANDT' como elemento de dados. O sistema preenche automaticamente o Tipo de Dados, Comprimento, Decimais e Breve Descrição. O campo 'Cliente' torna-se um campo-chave marcando a caixa 'Chave'.

Step 9 - O próximo campo é 'Cliente'. Marque a caixa para torná-lo um campo-chave e insira o novo elemento de dados 'ZCUSTNUM'. Clique no botão Salvar.

Step 10 - Como o Elemento de Dados 'ZCUSTNUM' ainda não existe, ele deve ser criado. Clique duas vezes no novo elemento de dados e a janela 'Criar elemento de dados' será exibida. Responda 'Sim' e uma janela 'Manter Elemento de Dados' aparecerá.

Step 11 - Insira 'Número do cliente' na área de breve descrição. O tipo de dados Elementar denominado 'Domínio' deve ser definido para o novo elemento Dados. Então digite 'ZCUSTD1', clique duas vezes nele e concorde em salvar as alterações feitas. Escolha 'Sim' para criar o domínio e digite na caixa 'Descrição resumida' uma descrição do domínio.

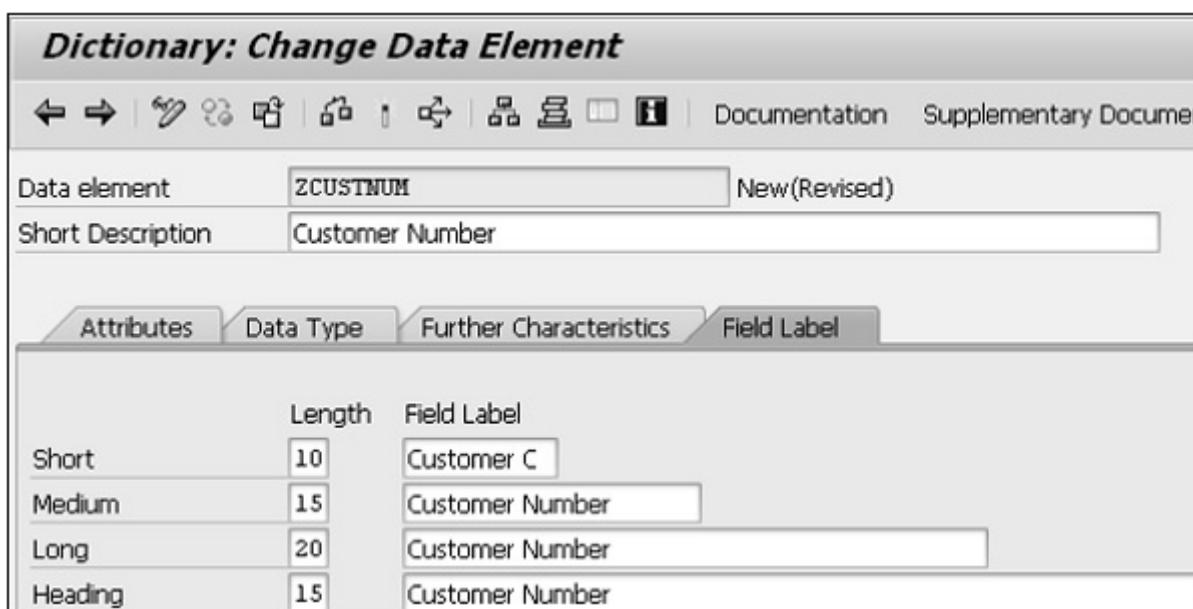


A guia 'Definição' abre automaticamente. O primeiro campo é 'Tipo de dados'.

Step 12 - Clique dentro da caixa e selecione o tipo 'NUMC' no menu suspenso. Digite o número 8 no campo 'Não. de caracteres (máximo de 8 caracteres)' e digite 0 na área 'Casas decimais'. O comprimento de saída de 8 deve ser selecionado e pressione Enter. A descrição do campo 'NUMC' deve reaparecer, confirmando que esta é uma entrada válida.

Step 13 - Clique no botão Salvar e ative o objeto.

Step 14 - Pressione F3 para retornar à tela 'Manter/Alterar elemento de dados'. Crie quatro rótulos de campo conforme mostrado no instantâneo a seguir. Depois disso, salve e ative o elemento.



Step 15 - Pressione o botão Voltar para retornar à tela de manutenção da mesa. A coluna Cliente possui o tipo de dados, comprimento, decimais e descrição resumida corretos. Isso indica a criação bem-sucedida de um elemento de Dados e também do Domínio utilizado.

Dictionary: Change Table							
Technical Settings Indexes... API							
Transp. Table		ZCUSTOMERS1		New			
Short Description		Customers Table					
Attributes	Delivery and Maintenance	Fields	Entry help/check	Currency/Quantity Field			
Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
CLIENT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	MANDT	CLNT	3	0	Client
CUSTOMER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	ZCUSTNUM	NUMC	8	0	Customer Number

Da mesma forma, precisamos criar três campos adicionais, como NOME, TÍTULO e DOB.

Step 16 - Selecione 'Configurações técnicas' na barra de ferramentas. Escolha APPL0 para a 'classe de dados' e a primeira categoria de tamanho 0 para o campo 'categoria de tamanho'. No caso de opções de buffer, 'Buffering não permitido' deve ser selecionado.

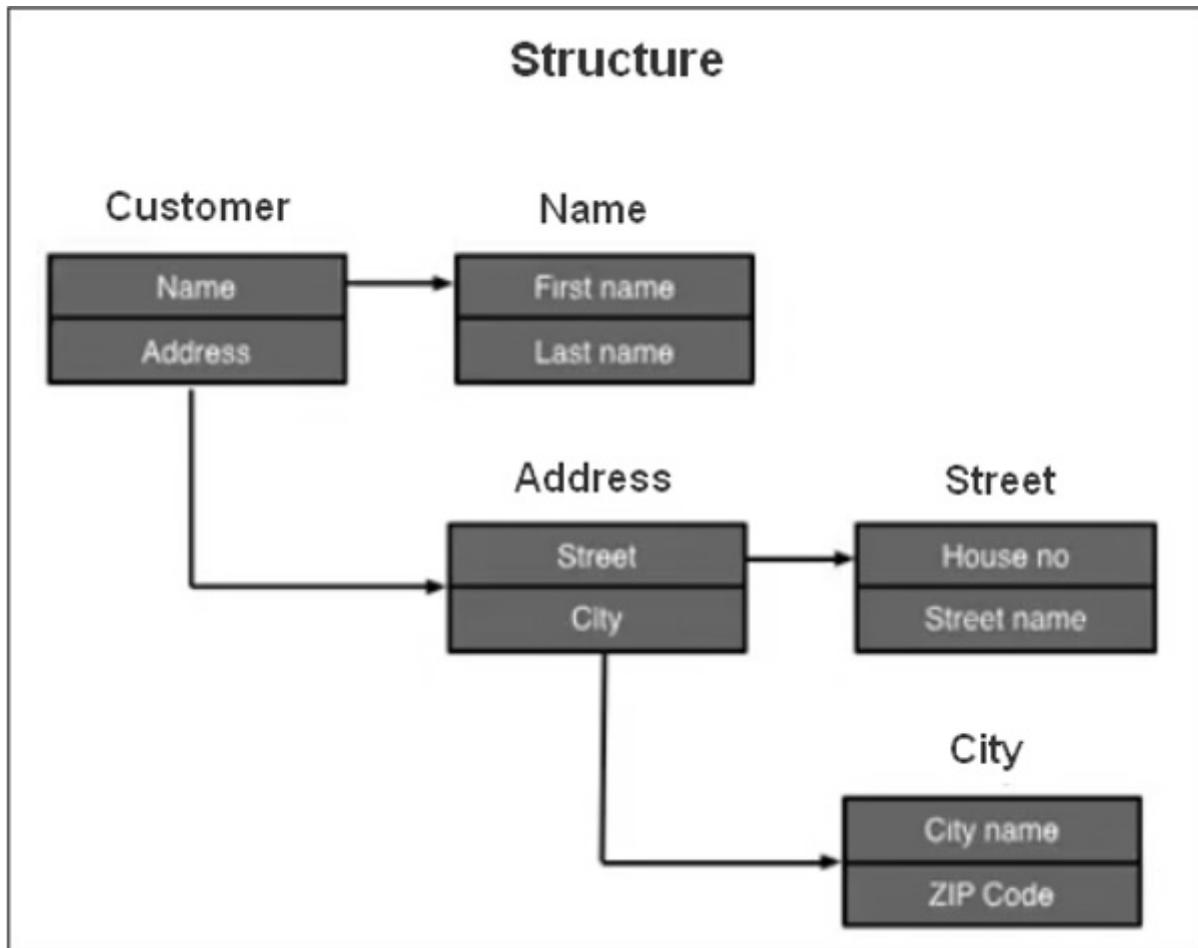
Step 17 - Clique em Salvar. Volte para a mesa e ative-a. A tela a seguir é exibida.

Dictionary: Display Table							
Technical Settings Indexes... API							
Transp. Table		ZCUSTOMERS1		Active			
Short Description		Customers Table					
Attributes	Delivery and Maintenance	Fields	Entry help/check	Currency/Quantity Fields			
Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
CLIENT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	MANDT	CLNT	3	0	Client
CUSTOMER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	ZCUSTNUM	NUMC	8	0	Customer Number
NAME	<input type="checkbox"/>	<input type="checkbox"/>	ZCUSTNAME	CHAR	40	0	Name Data Element
TITLE	<input type="checkbox"/>	<input type="checkbox"/>	ZTITLE1	CHAR	15	0	Title Data Element
DOB	<input type="checkbox"/>	<input type="checkbox"/>	ZDOB1	DATS	8	0	DOB Data Element

A tabela 'ZCUSTOMERS1' está ativada.

SAP ABAP - Estruturas

Estrutura é um objeto de dados composto por componentes de qualquer tipo de dados armazenados um após o outro na memória.



As estruturas são úteis para pintar campos de tela e para manipular dados que possuem um formato consistente definido por um número discreto de campos.

Uma estrutura pode ter apenas um único registro em tempo de execução, mas uma tabela pode ter muitos registros.

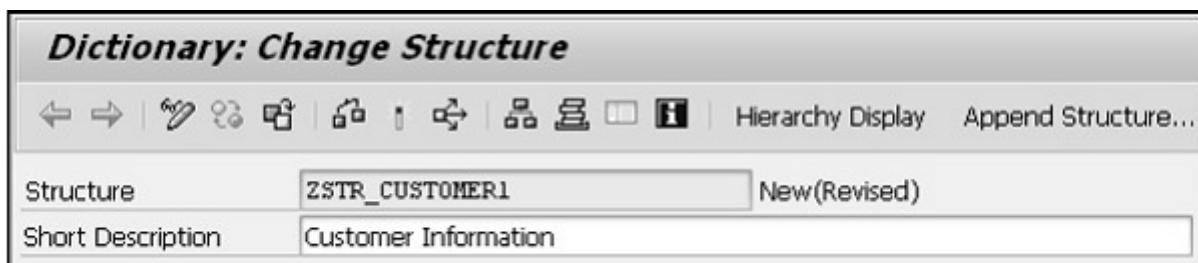
Criando uma Estrutura

Step 1 - Vá para a transação SE11.

Step 2 - Clique na opção 'Tipo de dados' na tela. Digite o nome 'ZSTR_CUSTOMER1' e clique no botão Criar.

Step 3 - Selecione a opção 'Estrutura' na próxima tela e pressione Enter. Você pode ver o assistente 'Manter/Alterar estrutura'.

Step 4 - Insira a breve descrição conforme mostrado no instantâneo a seguir.



Step 5 - Insira o componente (nome do campo) e o tipo de componente (elemento de dados).

Nota : Aqui os nomes dos componentes começam com Z conforme recomendação SAP. Vamos usar elementos de dados que já criamos na tabela do banco de dados.

Step 6 - Você precisa salvar, verificar e ativar depois de fornecer todos os componentes e tipos de componentes.

A seguinte tela aparece -

Object name	
D.. Object	Obj. name
TABL	ZSTR_CUST
TABL	ZSTR_CUSTOMER1
DOMA	Z_SEP_21

Step 7 - Como este 'ZSTR_CUSTOMER1' está destacado, clique no botão verde. Esta janela desaparece e a barra de status exibirá a mensagem 'Ativo'.

A estrutura agora está ativada conforme mostrado no instantâneo a seguir -

Structure	ZSTR_CUSTOMER1	Active		
Short Description	Customer Information			
Attributes Components Entry help/check Currency/quantity fields				
Component	Typing Method	Component Type		
CLIENT	Types	MANDT		
NAME	Types	ZCUSTNAME		
CUSTOMER	Types	ZCUSTNUM		
DOB	Types	ZDOB1		

SAP ABAP - Visualizações

Uma View atua apenas como uma tabela de banco de dados. Mas não ocupará espaço de armazenamento. Uma visualização age de forma semelhante a uma tabela virtual - uma tabela que não possui existência física. Uma visão é criada combinando os dados de uma ou mais tabelas contendo informações sobre um objeto de aplicação. Usando visualizações, você pode representar um subconjunto dos dados contidos em uma tabela ou unir várias tabelas em uma única tabela virtual.

Os dados relacionados a um objeto de aplicativo são distribuídos entre diversas tabelas usando visualizações de banco de dados. Eles usam a condição de junção interna para unir os dados de diferentes tabelas. Uma visualização de manutenção é usada para exibir e modificar os dados armazenados em um objeto de aplicativo. Cada visualização de manutenção possui um status de manutenção associado a ela.

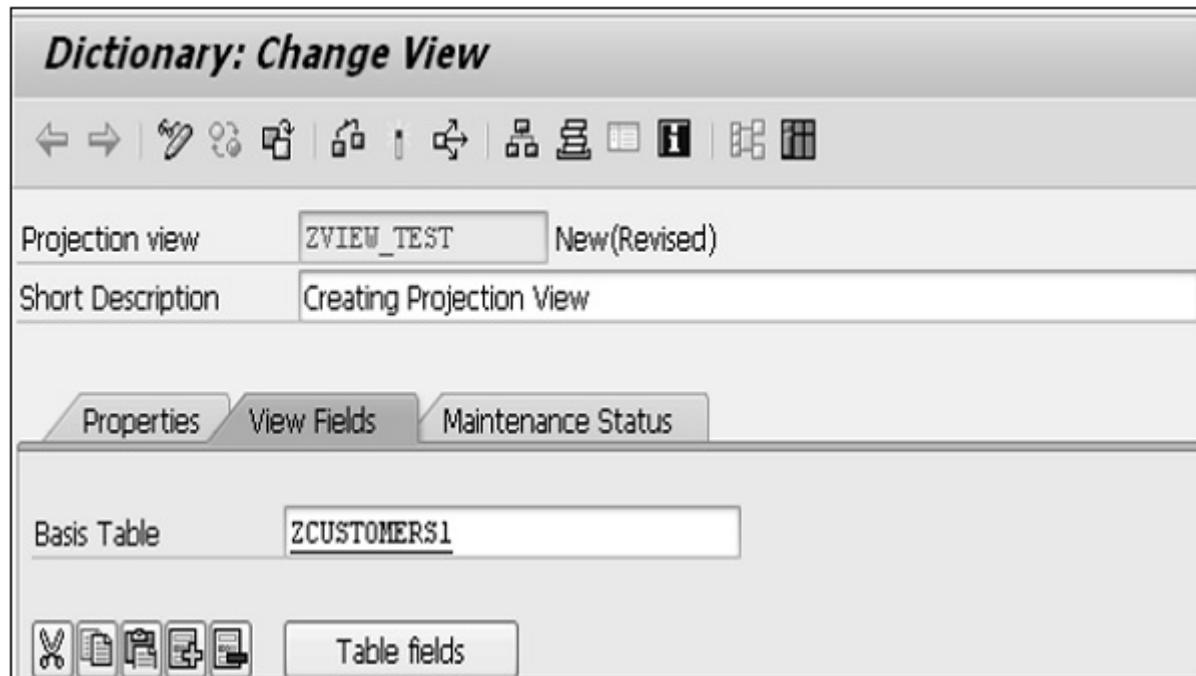
Usamos a visualização de projeção para mascarar campos indesejados e exibir apenas os campos relevantes em uma tabela. As vistas de projeção devem ser definidas em uma única tabela transparente. Uma visualização de projeção contém exatamente uma tabela. Não podemos definir condições de seleção para vistas de projeção.

Criando uma visualização

Step 1 - Selecione o botão de opção Exibir na tela inicial do ABAP Dictionary. Insira o nome da visualização a ser criada e clique no botão Criar. Inserimos o nome da visualização como ZVIEW_TEST.

Step 2 - Selecione o botão de opção de visualização de projeção ao escolher o tipo de visualização e clique no botão Copiar. A tela 'Dicionário: Alterar visualização' é exibida.

Step 3 - Insira uma breve descrição no campo Descrição resumida e o nome da tabela a ser usada no campo Tabela base, conforme mostrado no instantâneo a seguir.



Step 4 - Clique no botão 'Campos da tabela' para incluir os campos da tabela ZCUSTOMERS1 na visualização da projeção.

Step 5 - A tela Seleção de Campo da Tabela ZCUSTOMERS1 é exibida. Selecione os campos que você deseja incluir na visualização da projeção, conforme mostrado no instantâneo a seguir.

Field Selection from Table ZCUSTOMERS1	
Field Name	S Short description
<input checked="" type="checkbox"/> CLIENT	<input checked="" type="checkbox"/> Client
<input checked="" type="checkbox"/> CUSTOMER	<input checked="" type="checkbox"/> Customer Number
<input checked="" type="checkbox"/> NAME	<input type="checkbox"/> Name Data Element
<input type="checkbox"/> TITLE	<input type="checkbox"/> Title Data Element
<input type="checkbox"/> DOB	<input type="checkbox"/> DOB Data Element

Step 6 - Após clicar no botão Copiar, todos os campos selecionados para a visualização da projeção são exibidos na tela 'Dicionário: Alterar visualização'.

Dictionary: Change View						
Projection view	ZVIEW_TEST Active					
Short Description	Creating Projection View					
Properties	View Fields					
Maintenance Status						
Basis Table	ZCUSTOMERS1					
	Table fields					
View fields						
Field Name	Key	Data element	M...	DTyp	Length	Short description
CLIENT	<input checked="" type="checkbox"/>	<u>MANDT</u>	<input type="checkbox"/>	CLNT	3	Client
CUSTOMER	<input checked="" type="checkbox"/>	<u>ZCUSTNUM</u>	<input type="checkbox"/>	NUMC	8	Customer Number
NAME	<input type="checkbox"/>	<u>ZCUSTNAME</u>	<input type="checkbox"/>	CHAR	40	Name Data Element

Step 7 - Selecione a guia Status de manutenção para definir um método de acesso. Escolha o botão de opção somente leitura e a opção 'Exibição/Manutenção permitida com restrições' no menu suspenso de 'Navegador de dados/Manutenção de exibição de tabela'.

Step 8 - Salve e ative-o. Na tela 'Dicionário: Alterar visualização' selecione Utilities(M) > Contents para exibir a tela de seleção para ZVIEW_TEST.

Step 9 - Clique no ícone Executar. A saída da visualização de projeção aparece conforme mostrado na captura de tela a seguir.

Data Browser: Table ZVIEW_TEST Select Entries 4			
Table: ZVIEW_TEST		Displayed Fields: 3 of 3 Fixed Columns: 2	
	Client	Customer Number	Name
	800	00100001	MARK
	800	00100002	JAMES
	800	00100003	AURIELE
	800	00100004	STEPHEN

A tabela ZCUSTOMERS1 consiste em 5 campos. Aqui os campos exibidos são 3 (Cliente, Número do Cliente e Nome) com 4 entradas. Os números de clientes vão de 100001 a 100004 com nomes apropriados.

SAP ABAP - Pesquisar Ajuda

Search Help, outro objeto de repositório do Dicionário ABAP, é usado para exibir todos os valores possíveis para um campo na forma de uma lista. Esta lista também é conhecida como **lista de ocorrências**. Você pode selecionar os valores que serão inseridos nos campos desta lista de ocorrências em vez de inserir o valor manualmente, o que é tedioso e sujeito a erros.

Criando ajuda de pesquisa

Step 1 - Vá para a transação SE11. Selecione o botão de opção para Pesquisar ajuda. Insira o nome da ajuda de pesquisa a ser criada. Vamos inserir o nome ZSRCH1. Clique no botão Criar.

Step 2 - O sistema solicitará que o tipo de ajuda de pesquisa seja criado. Selecione a ajuda de pesquisa elementar, que é o padrão. A tela para criar ajuda de pesquisa elementar, conforme mostrado na imagem a seguir, é exibida.

Step 3 - No método de seleção, precisamos indicar se nossa fonte de dados é uma tabela ou uma visualização. No nosso caso, é uma mesa. A tabela é ZCUSTOMERS1. Ele é selecionado em uma lista de seleção.

Step 4 - Após inserir o método de seleção, o próximo campo é o tipo de diálogo. Isso controla a aparência da caixa de diálogo restritiva. Há uma lista suspensa com três opções. Vamos selecionar a opção 'Exibir valores imediatamente'.

Dictionary: Change Search Help

Elementary srch hlp	ZSRCH1	Inactive
Short description	Search Help Demo	
Attributes Definition		
Data collection Selection method: ZCUSTOMERS1		Dialog behavior Dialog type: Display values immediately Hot key: <input type="text"/>

Step 5 - A seguir está a área de parâmetros. Para cada parâmetro ou campo de ajuda da Pesquisa, esses campos de coluna devem ser inseridos de acordo com os requisitos.

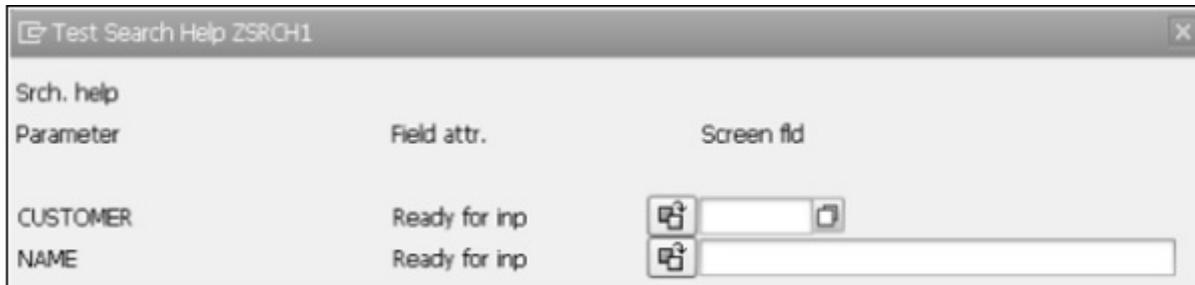
- **Parâmetro de ajuda de pesquisa** - Este é um campo da fonte de dados. Os campos da tabela são listados na lista de seleção. Os campos participantes da ajuda de pesquisa seriam inseridos, um campo em cada linha. Vamos incluir os dois campos CLIENTE e NOME. A forma como esses dois campos participam é indicada no restante das colunas.

Search help parameter	IMP	EXP	LPos	SPos	SDis	Data element
CUSTOMER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	1	<input type="checkbox"/>	ZCUSTNUM
NAME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2	2	<input type="checkbox"/>	ZCUSTNAME

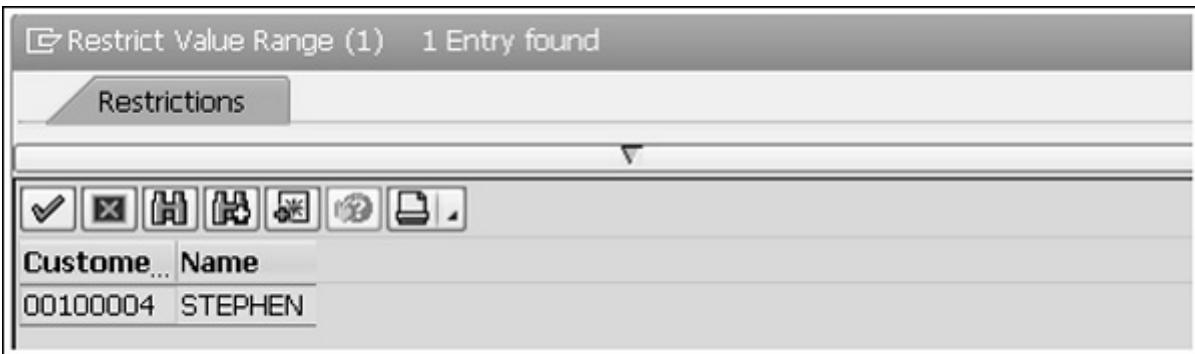
- **Import** - Este campo é uma caixa de seleção para indicar se um parâmetro de ajuda de pesquisa é um parâmetro de importação. A exportação ou importação é feita com referência à ajuda da pesquisa.
- **Export** - Este campo é uma caixa de seleção para indicar se um parâmetro de ajuda de pesquisa é um parâmetro de exportação. A exportação será a transferência dos valores dos campos da lista de seleção para os campos da tela.
- **LPos** - Seu valor controla a posição física do parâmetro ou campo de ajuda da Pesquisa na lista de seleção. Se você inserir o valor 1, o campo aparecerá na primeira posição da lista de seleção e assim por diante.
- **SPos** - Controla a posição física do parâmetro ou campo Search Help na caixa de diálogo restritiva. Se você inserir o valor 1, o campo aparecerá na primeira posição da caixa de diálogo restritiva e assim por diante.
- **Elemento de dados** - Cada parâmetro ou campo da Ajuda de pesquisa, por padrão, recebe um elemento de dados que foi atribuído a ele na fonte de dados (Tabela ou Visualizaç~`

Este nome de elemento de dados aparece no modo de exibição.

Step 6 - Execute uma verificação de consistência e ative a ajuda da pesquisa. Pressione F8 para executar. A tela 'Test Search Help ZSRCH1' aparece conforme mostrado na captura de tela a seguir.



Step 7 - Vamos inserir o número 100004 no campo da tela 'Pronto para entrada' do CLIENTE. Pressione Enter.



O número do cliente, 100004, e o nome 'STEPHEN' são exibidos.

SAP ABAP - Bloquear objetos

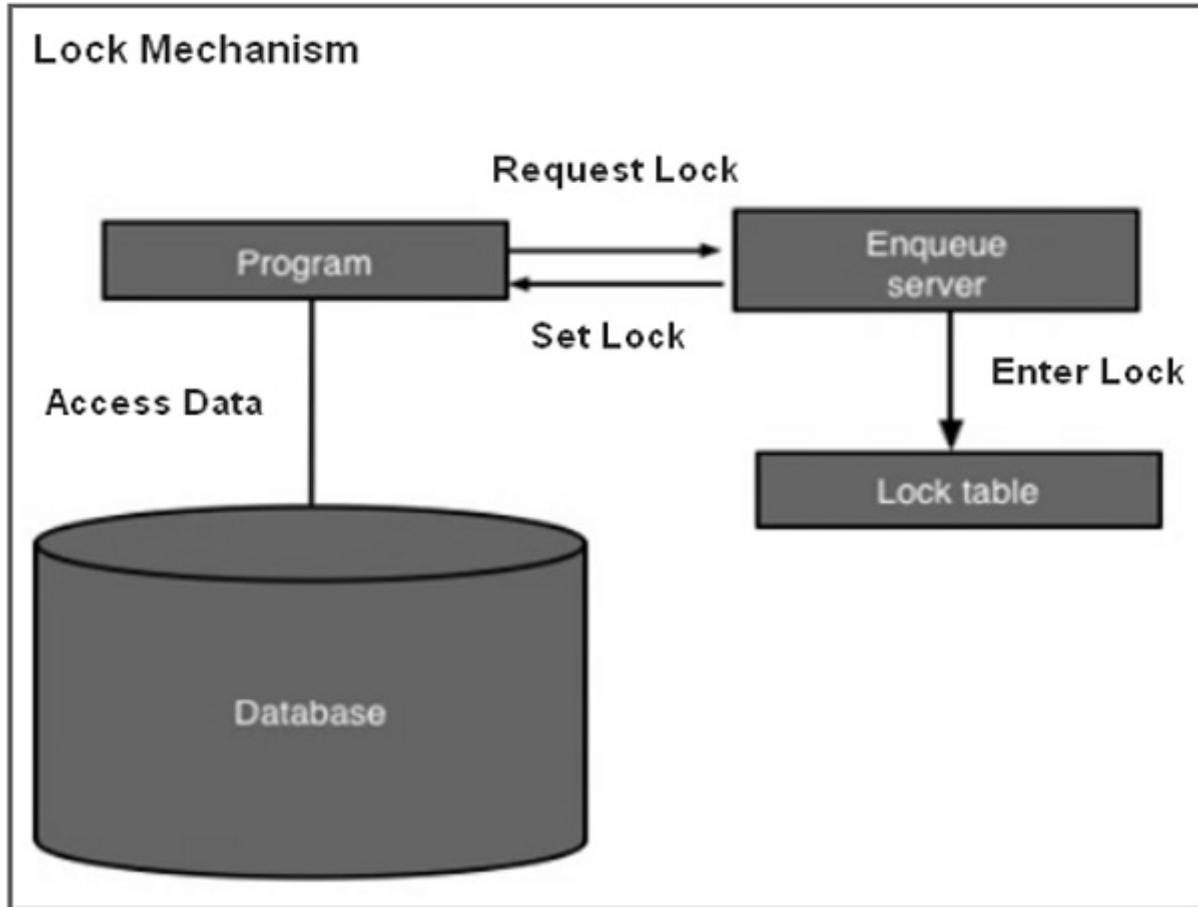
Lock Object é um recurso oferecido pelo ABAP Dictionary que permite sincronizar o acesso aos mesmos dados por mais de um programa. Os registros de dados são acessados com a ajuda de programas específicos. Objetos de bloqueio são usados no SAP para evitar inconsistência quando os dados são inseridos ou alterados no banco de dados. As tabelas cujos registros de dados serão bloqueados devem ser definidas em um Lock Object, juntamente com seus campos-chave.

Mecanismo de bloqueio

A seguir estão as duas funções principais realizadas com o mecanismo de bloqueio -

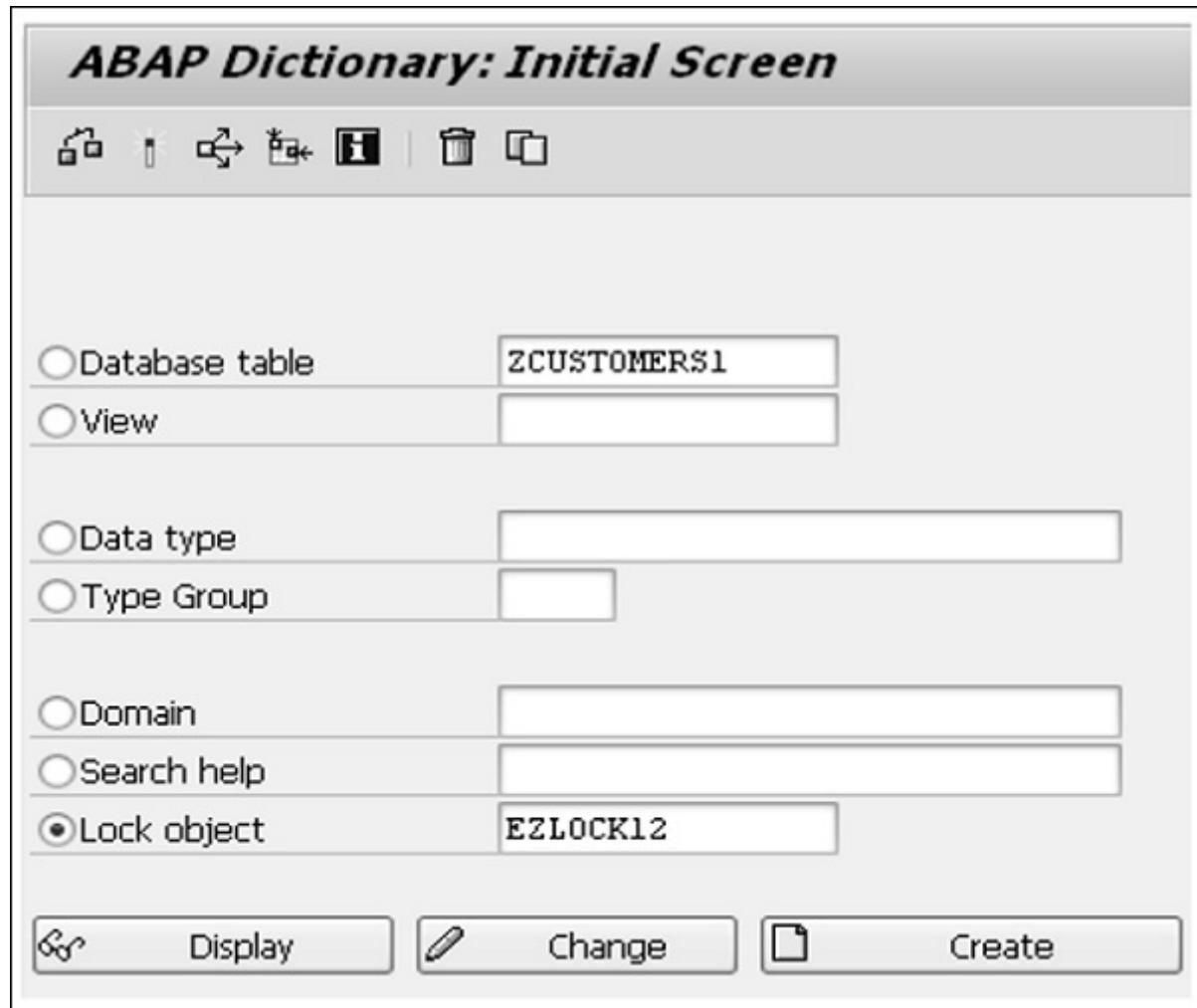
- Um programa pode se comunicar com outros programas sobre registros de dados que está apenas lendo ou alterando.
- Um programa pode impedir a leitura de dados que acabaram de ser alterados por outro programa.

Uma **solicitação de bloqueio** é gerada primeiro pelo programa. Então essa solicitação vai para o servidor Enqueue e o bloqueio é criado na tabela de bloqueios. O servidor Enqueue define o bloqueio e o programa está finalmente pronto para acessar os dados.



Criando objetos de bloqueio

Step 1 - Vá para a transação SE11. A tela a seguir é aberta.

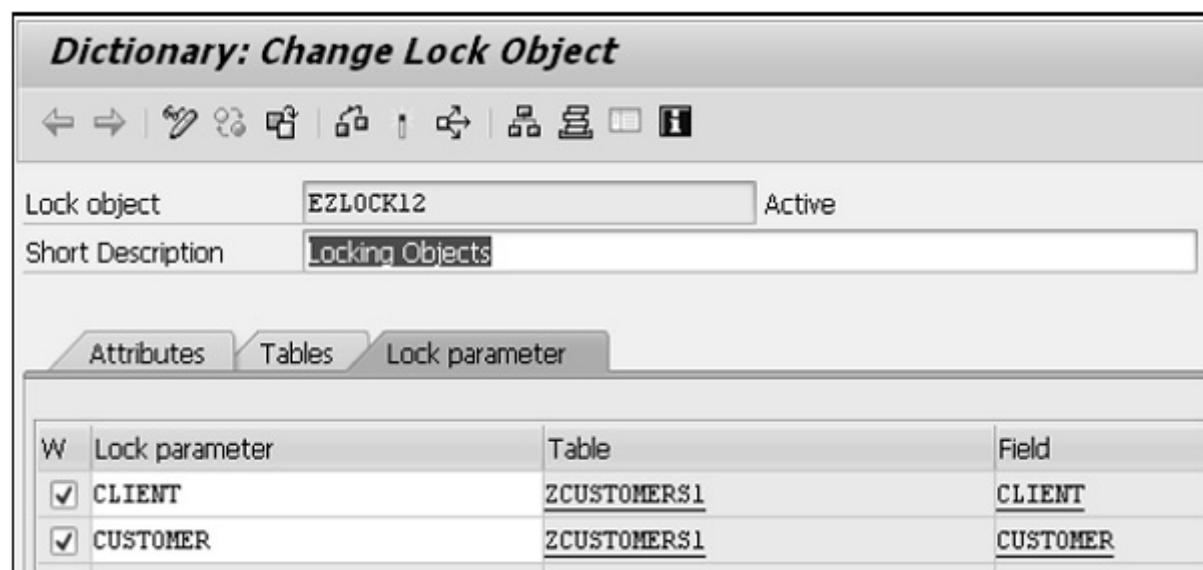


Step 2 - Clique no botão de opção 'Bloquear objeto'. Insira o nome do objeto de bloqueio começando com E e clique no botão Criar. Aqui usamos EZLOCK12.

Step 3 - Insira o campo de breve descrição e clique na guia Tabelas.

Step 4 - Insira o nome da tabela no campo Nome e selecione o modo de bloqueio como Write Lock.

Step 5 - Clique na guia Bloquear parâmetro, a seguinte tela aparecerá.



Step 6 - Salve e ative. Automaticamente serão gerados 2 módulos de função. Para verificar os módulos de função, podemos usar Ir para → Bloquear Módulos.

Step 7 - Clique em Bloquear Módulos e a tela a seguir será aberta.

Repository Info System: Function Modules Find (2 Hits)	
Function group	Function group short text
Function Module Name	Short text for function module
/1BCDWBEN/TEN0000	xRPM 4.0 Demand Planning
DEQUEUE_EZLOCK12	Release lock on object EZLOCK12
ENQUEUE_EZLOCK12	Request lock for object EZLOCK12

O objeto de bloqueio foi criado com sucesso.

Os campos-chave de uma tabela incluída em um objeto Lock são chamados de argumentos de bloqueio e são usados como parâmetros de entrada em módulos de função. Esses argumentos são usados para definir e remover os bloqueios gerados pela definição do Lock Object.

SAP ABAP - Modularização

É uma boa prática manter seus programas tão independentes e fáceis de ler quanto possível. Apenas tente dividir tarefas grandes e complicadas em tarefas menores e mais simples, colocando cada tarefa em seu módulo individual, no qual o desenvolvedor possa se concentrar sem outras distrações.

No ambiente SAP ABAP, a modularização envolve a organização de programas em unidades modulares, também conhecidas como **blocos lógicos**. Ele reduz a redundância e aumenta a legibilidade do programa mesmo durante sua criação e, posteriormente, durante o ciclo de manutenção. A modularização também permite a reutilização do mesmo código novamente. O ABAP tornou necessário que os desenvolvedores modularizassem, ou seja, organizassem os programas relativamente mais, do que nas linguagens baseadas em OOPS, que possuem relativamente mais recursos modulares integrados. Depois que uma pequena seção modularizada de código é concluída, depurada e assim por diante, ela não precisa ser devolvida posteriormente e os desenvolvedores podem então seguir em frente e se concentrar em outras questões.

Os programas ABAP são compostos de blocos de processamento conhecidos como blocos de processamento modularizantes. Eles são -

- Os blocos de processamento chamados de fora do programa e do ambiente de tempo de execução ABAP (ou seja, blocos de eventos e módulos de diálogo).
- Processando blocos chamados de programas ABAP.

Além da modularização com blocos de processamento, módulos de código-fonte são utilizados para modularizar seu código-fonte por meio de macros e incluir programas.

Modularização no nível do código-fonte -

- Macros locais
- Programas de inclusão global

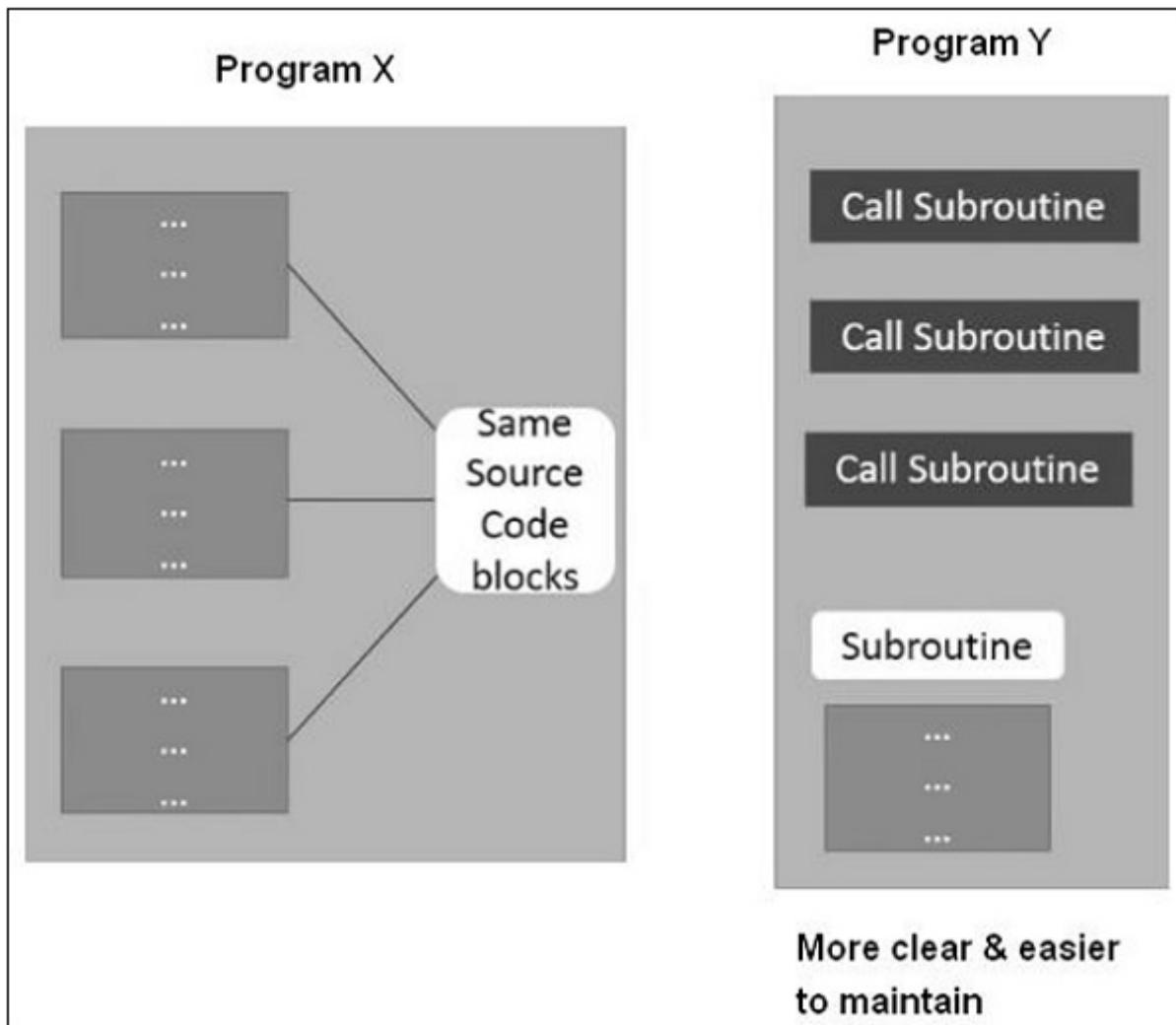
Modularização através do processamento de blocos chamados de programas ABAP -

- Sub-rotinas
- Módulos de função

Modularizar um código-fonte significa colocar uma sequência de instruções ABAP em um módulo. O código-fonte modularizado pode ser chamado em um programa conforme a necessidade do usuário. Os módulos de código-fonte melhoram a legibilidade e a compreensão dos programas ABAP. A criação de módulos de código-fonte individuais também evita que seja necessário escrever repetidamente as mesmas instruções, o que, por sua vez, torna o código mais fácil de entender para qualquer pessoa que o leia pela primeira vez.

SAP ABAP - Subrotinas

Uma sub-rotina é uma seção reutilizável de código. É uma unidade de modularização dentro do programa onde uma função é encapsulada na forma de código-fonte. Você pagina uma parte de um programa para uma sub-rotina para obter uma melhor visão geral do programa principal e para usar a sequência correspondente de instruções muitas vezes, conforme ilustrado no diagrama a seguir.



Temos o programa X com 3 **blocos de código-fonte** diferentes . Cada bloco possui as mesmas instruções ABAP. Basicamente, são os mesmos blocos de código. Para tornar esse código mais fácil de manter, podemos encapsular o código em uma sub-rotina. Podemos chamar esta sub-rotina em nossos programas quantas vezes quisermos. Uma sub-rotina pode ser definida usando instruções Form e EndForm.

A seguir está a sintaxe geral de uma definição de sub-rotina.

```
FORM <subroutine_name>.
```

```
<statements>
```

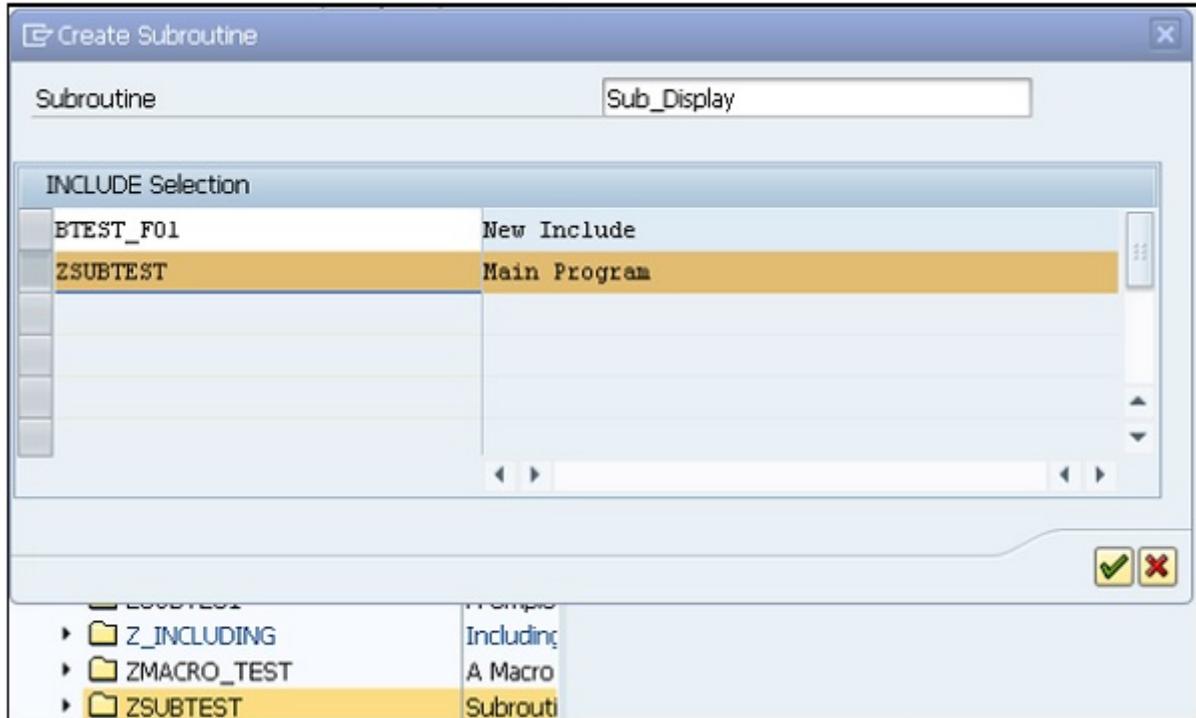
```
ENDFORM.
```

Podemos chamar uma sub-rotina usando a instrução PERFORM. O controle salta para a primeira instrução executável na sub-rotina <subroutine_name>. Quando ENDFORM é encontrado, o controle volta para a instrução seguinte à instrução PERFORM.

Exemplo

Step 1 - Vá para a transação SE80. Abra o programa existente e clique com o botão direito no programa. Neste caso, é 'ZSUBTEST'.

Step 2 – Select Create and then select Subroutine. Write the subroutine name in the field and then click the continue button. The subroutine name is 'Sub_Display' as shown in the following screenshot.



Step 3 – Write the code in FORM and ENDFORM statement block. The subroutine has been created successfully.

We need to include PERFORM statement to call the subroutine. Let's take a look at the code –

```

REPORT ZSUBTEST.
PERFORM Sub_Display.

* Form Sub_Display
* --> p1 text
* <-- p2 text

FORM Sub_Display.
  Write: 'This is Subroutine'.
  Write: / 'Subroutine created successfully'.
ENDFORM.          " Sub_Display

```

Step 4 – Save, activate and execute the program. The above code produces the following output –

Subroutine Test:

This is Subroutine

Subroutine created successfully

Hence, using subroutines makes your program more function-oriented. It splits the program's task into sub-functions, so that each subroutine is responsible for one subfunction. Your program becomes easier to maintain as changes to functions often only have to be implemented in the subroutine.

SAP ABAP - Macros

If we want to reuse the same set of statements more than once in a program, we need to include them in a macro. For example, a macro can be useful for long calculations or for writing complex WRITE statements. We can only use a macro within a program in which it is defined. Macro definition should occur before the macro is used in the program.

Macros are designed based on placeholders. Placeholder works like pointers in C language. You can define a macro within the DEFINE...END-OF-DEFINITION statement.

Following is the basic syntax of a macro definition –

```
DEFINE <macro_name>. <statements>
END-OF-DEFINITION.

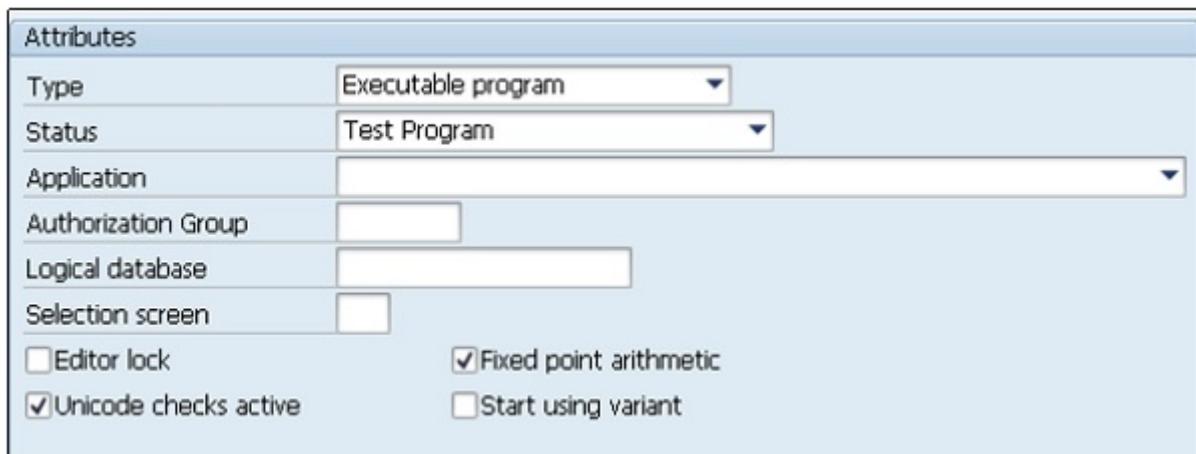
.....
<macro_name> [<param1> <param2>....].
```

É necessário definir uma macro antes de invocá-la. O <param1>.... substitui os espaços reservados &1...nas instruções ABAP contidas na definição da macro.

O número máximo de espaços reservados em uma definição de macro é nove. Ou seja, quando um programa é executado, o sistema SAP substitui a macro pelas instruções apropriadas e os marcadores &1, &2,...&9 são substituídos pelos parâmetros param1, param2,...param9. Podemos invocar uma macro dentro de outra macro, mas não na mesma macro.

Exemplo

Vá para a transação SE38. Crie um novo programa ZMACRO_TEST junto com a descrição no campo de texto curto, e também com atributos apropriados como Tipo e Status conforme mostrado na imagem a seguir -



A seguir está o código -

```

REPORT ZMACRO_TEST.
DEFINE mac_test.
WRITE: 'This is Macro &1'.
END-OF-DEFINITION.

PARAMETERS: s1 type C as checkbox.
PARAMETERS: s2 type C as checkbox.
PARAMETERS: s3 type C as checkbox default 'X'.

START-OF-SELECTION.
IF s1 = 'X'.
  mac_test 1. ENDIF.
IF s2 = 'X'.
  mac_test 2.
ENDIF.

IF s3 = 'X'.
  mac_test 3.
ENDIF.

```

Temos 3 caixas de seleção. Ao executar o programa, vamos marcar a caixa de seleção S2.



O código acima produz a seguinte saída -

A Macro Program

This is Macro 2

Se todas as caixas de seleção estiverem marcadas, o código produz a seguinte saída -

A Macro Program

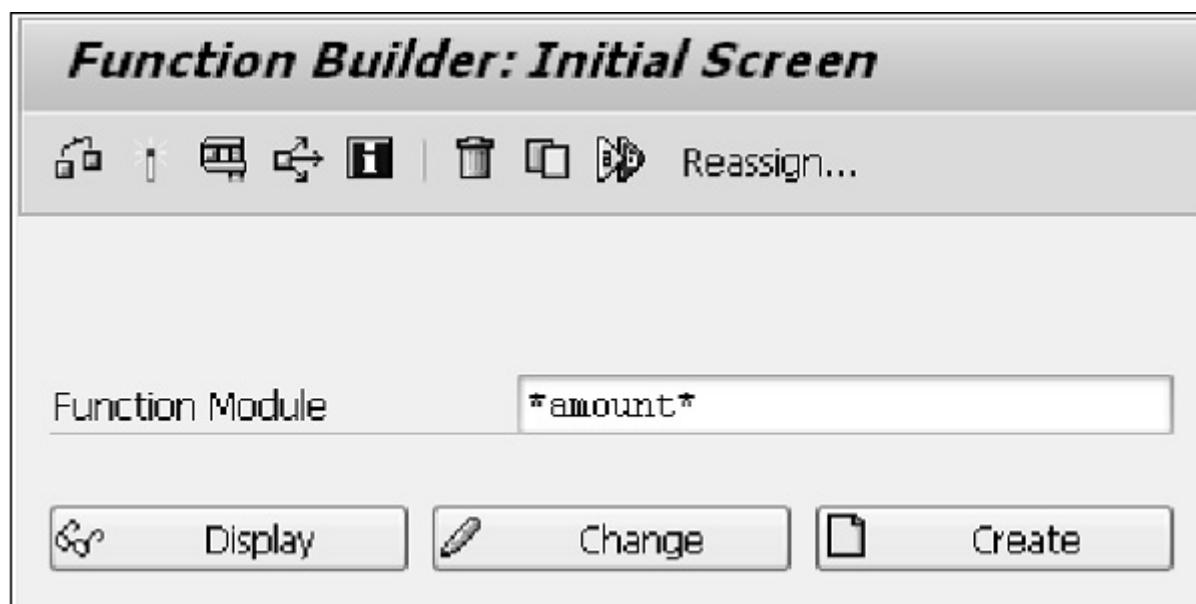
This is Macro 1 This is Macro 2 This is Macro 3

SAP ABAP - Módulos Funcionais

Os módulos de função constituem a maior parte de um sistema SAP, pois há anos a SAP modulariza o código por meio de módulos de função, permitindo a reutilização do código, por si, por seus desenvolvedores e também por seus clientes.

Módulos de função são subprogramas que contêm um conjunto de instruções reutilizáveis com parâmetros de importação e exportação. Ao contrário dos programas Include, os módulos de função podem ser executados de forma independente. O sistema SAP contém vários módulos de função predefinidos que podem ser chamados a partir de qualquer programa ABAP. O grupo de funções atua como uma espécie de contêiner para vários módulos de função que pertenceriam logicamente um ao outro. Por exemplo, os módulos de função para um sistema de folha de pagamento de RH seriam reunidos em um grupo de funções.

To look at how to create function modules, the function builder must be explored. You can find the function builder with transaction code SE37. Just type a part of a function module name with a wild card character to demonstrate the way function modules can be searched for. Type *amount* and then press the F4 key.



The results of the search will be displayed in a new window. The function modules are displayed in the lines with blue background and their function groups in pink lines. You may look further at the function group ISOC by using the Object Navigator screen (Transaction

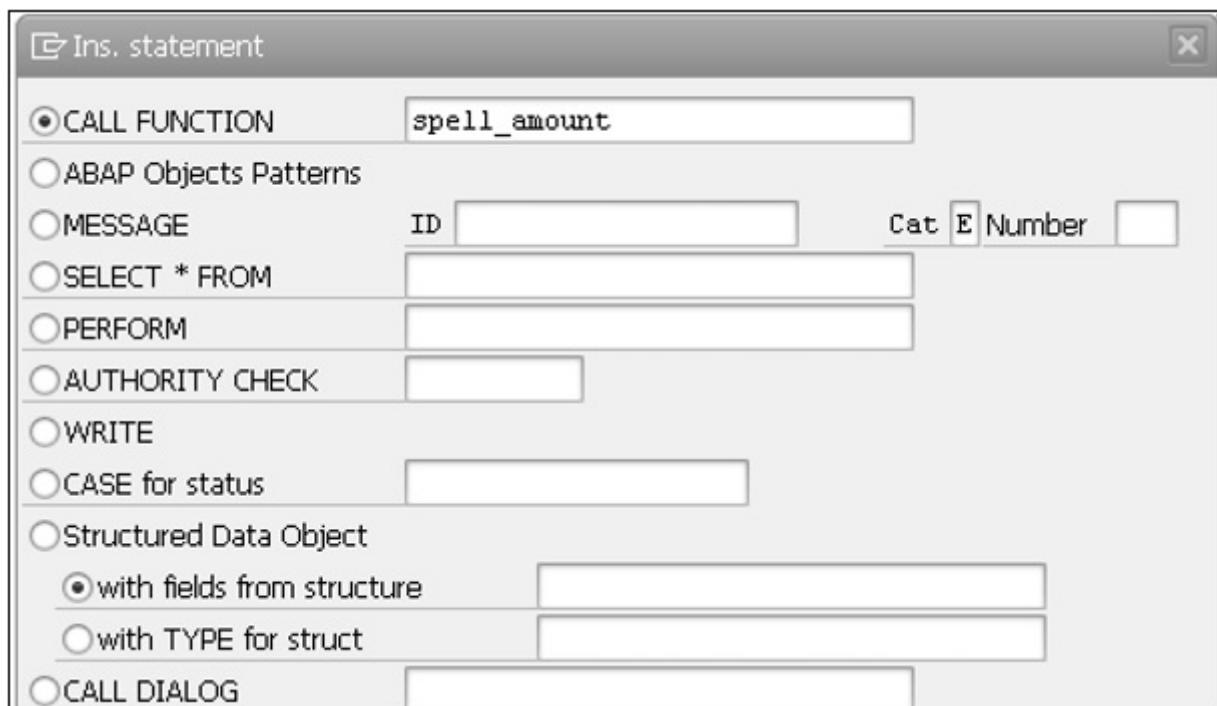
SE80). You can see a list of function modules and also other objects held in the function group. Let's consider the function module SPELL_AMOUNT. This function module converts numeric figures into words.

Creating a New Program

Step 1 – Go to transaction SE38 and create a new program called Z_SPELLAMOUNT.

Step 2 – Enter some code so that a parameter can be set up where a value could be entered and passed on to the function module. The text element text-001 here reads 'Enter a Value'.

Step 3 – To write the code for this, use CTRL+F6. After this, a window appears where 'CALL FUNCTION' is the first option in a list. Enter 'spell_amount' in the text box and click the continue button.



Step 4 – Some code is generated automatically. But we need to enhance the IF statement to include a code to WRITE a message to the screen to say "The function module returned a value of: sy-subrc" and add the ELSE statement so as to write the correct result out when the function module is successful. Here, a new variable must be set up to hold the value returned from the function module. Let's call this as 'result'.

Following is the code –

```

REPORT Z_SPELLAMOUNT.
data result like SPELL.

selection-screen begin of line.
selection-screen comment 1(15) text-001.

```

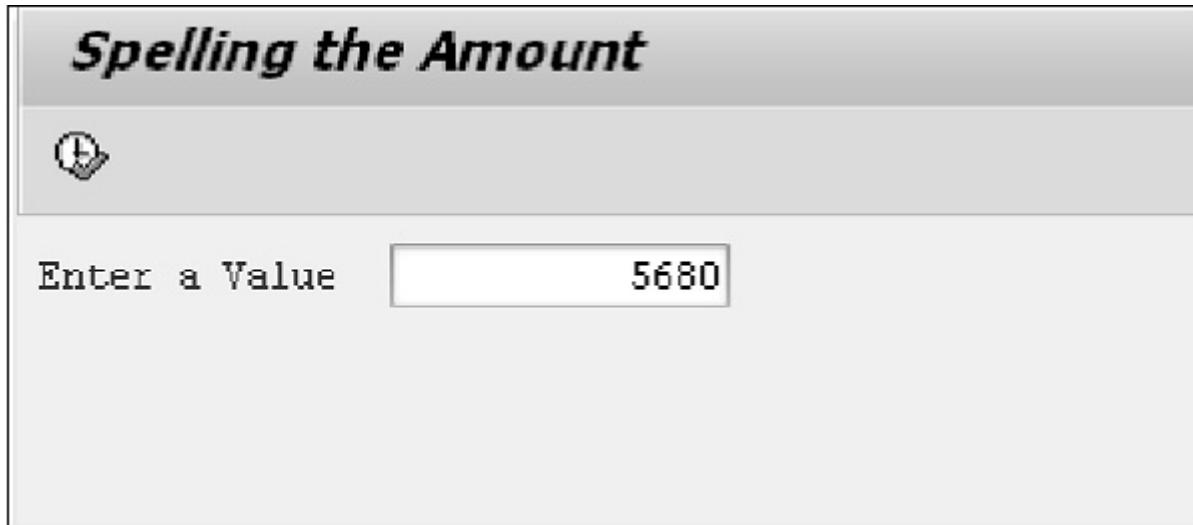
```

parameter num_1 Type I.
selection-screen end of line.
CALL FUNCTION 'SPELL_AMOUNT'
EXPORTING
AMOUNT = num_1
IMPORTING
IN_WORDS = result.

IF SY-SUBRC <> 0.
  Write: 'Value returned is:', SY-SUBRC.
else.
  Write: 'Amount in words is:', result-word.
ENDIF.
```

Step 5 – The variable which the function module returns is called IN_WORDS. Set up the corresponding variable in the program called ‘result’. Define IN_WORDS by using the LIKE statement to refer to a structure called SPELL.

Step 6 – Save, activate and execute the program. Enter a value as shown in the following screenshot and press F8.



The above code produces the following output –

```

Spelling the Amount
Amount in words is:
FIVE THOUSAND SIX HUNDRED EIGHTY
```

SAP ABAP - Include Programs

Include programs are global repository objects used to modularize the source code. They allow you to use the same source code in different programs. Include programs also allow you to

manage complex programs in an orderly way. In order to use an include program in another program, we use the following syntax –

```
INCLUDE <program_name>.
```

INCLUDE statement has the same effect as copying the source code of the include program <program_name> into another program. As include program can't run independently, it has to be built into other programs. You may also nest include programs.

Following are a couple of restrictions while writing the code for Include programs –

- Include programs can't call themselves.
- Include programs must contain complete statements.

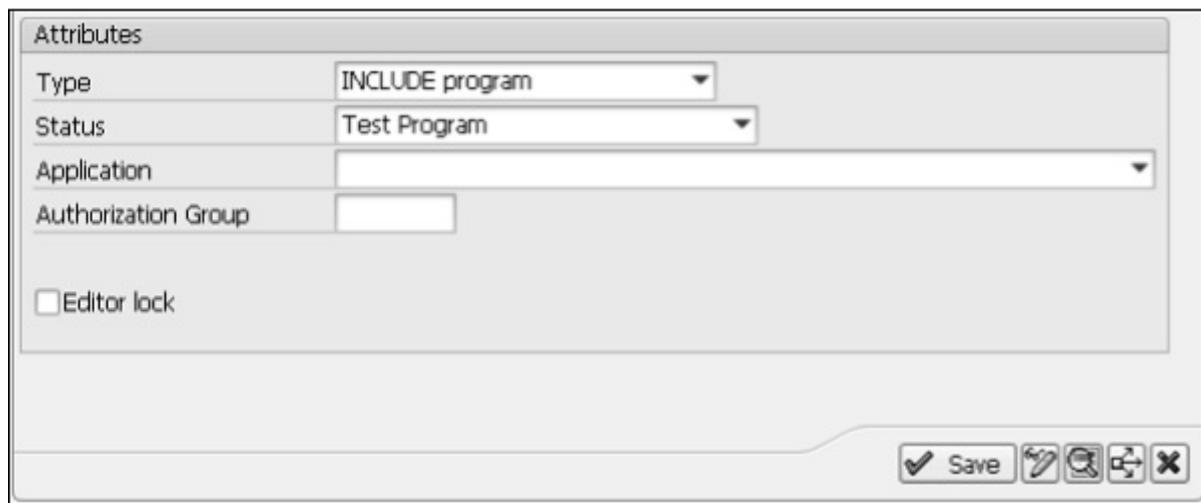
Following are the steps to create and use an Include program –

Step 1 – Create the program (Z_TOBEINCLUDED) to be included in ABAP Editor. Code to be included in ABAP Editor is –

```
PROGRAM Z_TOBEINCLUDED.
```

```
Write: / 'This program is started by:', SY-UNAME,
      / 'The Date is:', SY-DATUM,
      / 'Time is', SY-UZEIT.
```

Step 2 – Set the Type of the program to INCLUDE program, as shown in the following screenshot.



Step 3 – Click the 'Save' button and save the program in a package named ZINCL_PCKG.

Step 4 – Create another program where the program Z_TOBEINCLUDED has to be used. Here we have created another program named Z_INCLUDINGTEST and assigned the type for the program as Executable program.

Step 5 – The coding for Z_INCLUDINGTEST program includes the Z_TOBEINCLUDED program with the help of the INCLUDE statement as shown in the following code.

```
REPORT Z_INCLUDINGTEST.  
INCLUDE Z_TOBEINCLUDED.
```

Step 6 – Save, activate and execute the program.

The above code produces the following output –

```
This program is started by: SAPUSER  
The Date is: 06.10.2015  
Time is 13:25:11
```

SAP ABAP - Open SQL Overview

Open SQL indicates the subset of ABAP statements that enable direct access to the data in the central database of the current AS ABAP. Open SQL statements map the Data Manipulation Language functionality of SQL in ABAP that is supported by all database systems.

The statements of Open SQL are converted to database specific SQL in the Open SQL interface of the database interface. They are then transferred to the database system and executed. Open SQL statements can be used to access database tables that are declared in the ABAP Dictionary. The central database of AS ABAP is accessed by default and also access to other databases is possible via secondary database connections.

Whenever any of these statements are used in an ABAP program, it is important to check whether the action executed has been successful. If one tries to insert a record into a database table and it is not inserted correctly, it is very essential to know so that the appropriate action can be taken in the program. This can be done using a system field that has already been used, that is SY-SUBRC. When a statement is executed successfully, the SY-SUBRC field will contain a value of 0, so this can be checked for and one can continue with the program if it appears.

The DATA statement is used to declare a work area. Let's give this the name 'wa_customers1'. Rather than declaring one data type for this, several fields that make up the table can be declared. The easiest way to do this is using the LIKE statement.

INSERT Statement

The wa_customers1 work area is declared here LIKE the ZCUSTOMERS1 table, taking on the same structure without becoming a table itself. This work area can only store one record. Once it has been declared, the INSERT statement can be used to insert the work area and the record it holds into the table. The code here will read as 'INSERT ZCUSTOMERS1 FROM wa_customers1'.

The work area has to be filled with some data. Use the field names from the ZCUSTOMERS1 table. This can be done by forward navigation, double clicking the table name in the code or by opening a new session and using the transaction SE11. The fields of the table can then be copied and pasted into the ABAP editor.

Following is the code snippet –

```
DATA wa_customers1 LIKE ZCUSTOMERS1.
wa_customers1-customer = '100006'.
wa_customers1-name = 'DAVE'.
wa_customers1-title = 'MR'.
wa_customers1-dob = '19931017'.
INSERT ZCUSTOMERS1 FROM wa_customers1.
```

CHECK statement can then be used as follows. It means that if the record is inserted correctly, the system will state this. If not, then the SY-SUBRC code which will not equal zero will be displayed. Following is the code snippet –

```
IF SY-SUBRC = 0.
  WRITE 'Record Inserted Successfully'.
ELSE.
  WRITE: 'The return code is ', SY-SUBRC.
ENDIF.
```

Check the program, save, activate the code, and then test it. The output window should display as 'Record Inserted Successfully'.

CLEAR Statement

CLEAR statement allows a field or variable to be cleared out for the insertion of new data in its place, allowing it to be reused. CLEAR statement is generally used in programs and it allows existing fields to be used many times.

In the previous code snippet, the work area structure has been filled with data to create a new record to be inserted into the ZCUSTOMERS1 table and then a validation check is performed. If we want to insert a new record, CLEAR statement must be used so that it can then be filled again with the new data.

UPDATE Statement

If you want to update one or more existing records in a table at the same time then use UPDATE statement. Similar to INSERT statement, a work area is declared, filled with the r

data that is then put into the record as the program is executed. The record previously created with the INSERT statement will be updated here. Just edit the text stored in the NAME and TITLE fields. Then on a new line, the same structure as for the INSERT statement is used, and this time by using the UPDATE statement as shown in the following code snippet –

```
DATA wa_customers1 LIKE ZCUSTOMERS1.
wa_customers1-customer = '100006'.
wa_customers1-name = 'RICHARD'.
wa_customers1-title = 'MR'.
wa_customers1-dob = '19931017'.
UPDATE ZCUSTOMERS1 FROM wa_customers1.
```

As UPDATE statement gets executed, you can view the Data Browser in the ABAP Dictionary to see that the record has been updated successfully.

MODIFY Statement

MODIFY statement can be considered as a combination of the INSERT and UPDATE statements. It can be used to either insert a new record or modify an existing record. It follows a similar syntax to the previous two statements in modifying the record from the data entered into a work area.

When this statement is executed, the key fields involved will be checked against those in the table. If a record with these key field values already exist, it will be updated. If not, then a new record will be created.

Following is the code snippet for creating a new record –

```
CLEAR wa_customers1.

DATA wa_customers1 LIKE ZCUSTOMERS1.
wa_customers1-customer = '100007'.
wa_customers1-name = 'RALPH'.
wa_customers1-title = 'MR'.
wa_customers1-dob = '19910921'.
MODIFY ZCUSTOMERS1 FROM wa_customers1.
```

In this example, CLEAR statement is used so that a new entry can be put into the work area, and then customer (number) 100007 is added. Since this is a new, unique key field value, a new record will be inserted, and another validation check is executed.

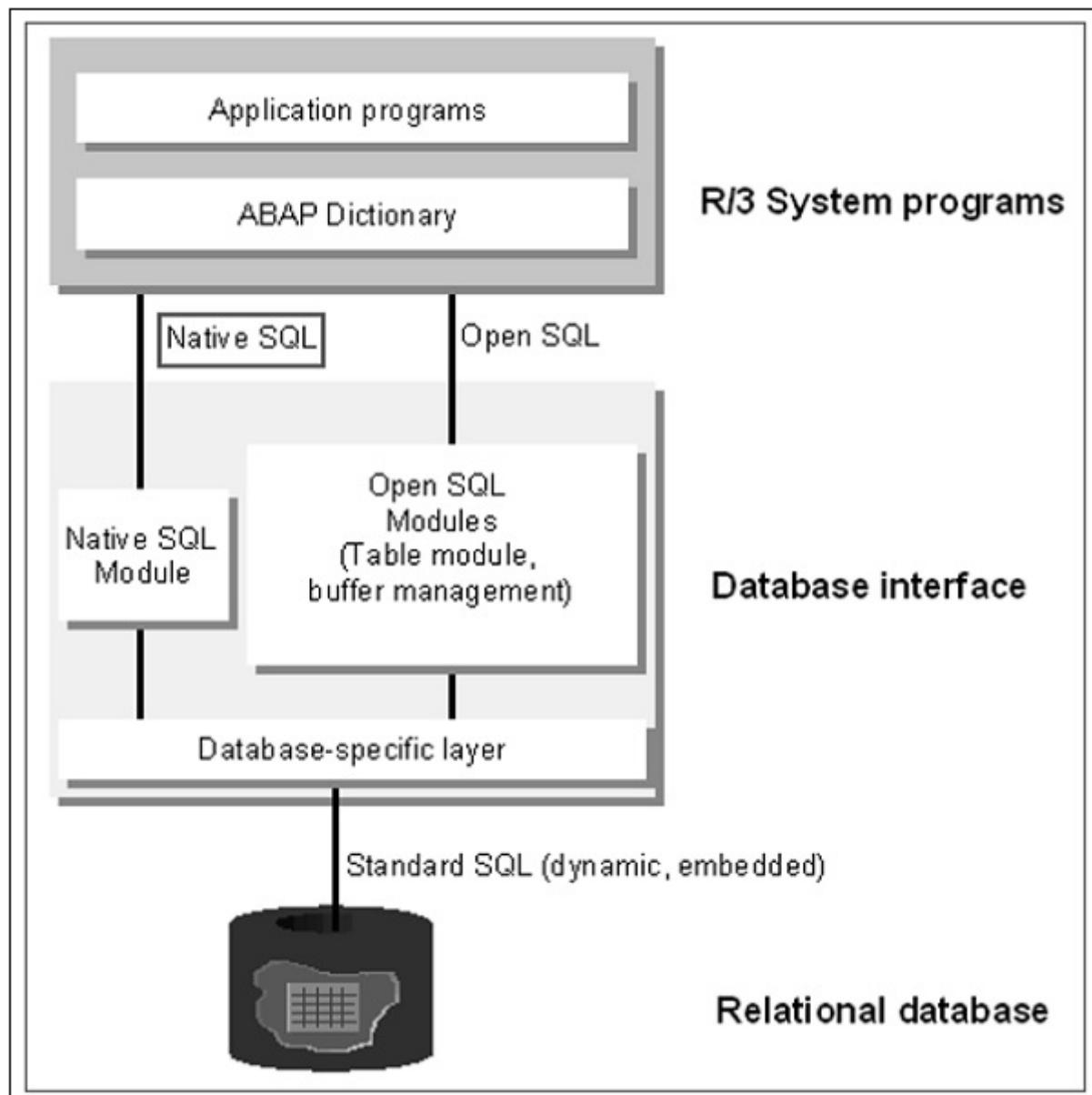
When this is executed and the data is viewed in the Data Browser, a new record will have been created for the customer number 100007 (RALPH).

The above code produces the following output (table contents) –

Client	Customer Number	Name	Title	DOB
800	00100001	MARK	MR	21.05.1981
800	00100002	JAMES	MR	14.08.1977
800	00100003	AURIELE	MS	19.06.1990
800	00100004	STEPHEN	MR	22.07.1985
800	00100005	MARGARET	MS	02.11.1994
800	00100006	RICHARD	MR	17.10.1993
800	00100007	RALPH	MR	21.09.1991

SAP ABAP - Native SQL Overview

The term ‘Native SQL’ refers to all statements that can be statically transferred to the Native SQL interface of the database interface. Native SQL statements do not fall within the language scope of ABAP and do not follow the ABAP syntax. ABAP merely contains statements for isolating program sections in which Native SQL statements can be listed.



In native SQL, mainly database-specific SQL statements can be used. These are transferred unchanged from the native SQL interface to a database system and executed. The full SQL language scope of the relevant database can be used and the addressed database tables do not have to be declared in the ABAP Dictionary. There is also a small set of SAP specific Native SQL statements that are handled in a specific way by the native SQL interface.

To use a Native SQL statement, you have to precede it with the EXEC SQL statement and end with ENDEXEC statement.

Following is the syntax –

```
EXEC SQL PERFORMING <form>.
  <Native SQL statement>
ENDEXEC.
```

These statements define an area in an ABAP program where one or more Native SQL statements can be listed. The statements entered are passed to the Native SQL interface and then processed as follows –

- All SQL statements that are valid for the program interface of the addressed database system can be listed between EXEC and ENDEXEC, in particular the DDL (data definition language) statements.
- These SQL statements are passed from the Native SQL interface to the database system largely unchanged. The syntax rules are specified by the database system, especially the case sensitivity rules for database objects.
- If the syntax allows a separator between individual statements, you may include many Native SQL statements between EXEC and ENDEXEC.
- SAP specific Native SQL language elements can be specified between EXEC and ENDEXEC. These statements are not passed directly from the Native SQL interface to the database, but they are transformed appropriately.

Example

SPFLI is a standard SAP Table that is used to store Flight schedule information. This is available within R/3 SAP systems depending on the version and release level. You can view this information when you enter the Table name SPFLI into the relevant SAP transaction such as SE11 or SE80. You can also view the data contained in this database table by using these two transactions.

```
REPORT ZDEMONATIVE_SQL.
DATA: BEGIN OF wa,
      connid  TYPE SPFLI-connid,
```

```

    cityfrom TYPE SPFLI-cityfrom,
    cityto  TYPE SPFLI-cityto,
END OF wa.

DATA c1 TYPE SPFLI-carrid VALUE 'LH'.
EXEC SQL PERFORMING loop_output.
  SELECT connid, cityfrom, cityto
  INTO :wa
  FROM SPFLI
  WHERE carrid = :c1
ENDEXEC.

FORM loop_output.
  WRITE: / wa-connid, wa-cityfrom, wa-cityto.
ENDFORM.

```

The above code produces the following output –

```

0400 FRANKFURT NEW YORK
2402 FRANKFURT BERLIN
0402 FRANKFURT NEW YORK

```

SAP ABAP - Internal Tables

Internal table is actually a temporary table, which contains the records of an ABAP program that it is being executed. An internal table exists only during the run-time of a SAP program. They are used to process large volumes of data by using ABAP language. We need to declare an internal table in an ABAP program when you need to retrieve data from database tables.

Data in an internal table is stored in rows and columns. Each row is called a **line** and each column is called a **field**. In an internal table, all the records have the same structure and key. As internal table exists till the associated program is being executed, the records of the internal table are discarded when the execution of the program is terminated. So internal tables can be used as temporary storage areas or temporary buffers where data can be modified as required. These tables occupy memory only at run-time and not at the time of their declaration.

Internal tables only exist when a program is running, so when the code is written, the internal table must be structured in such a way that the program can make use of it. You will find that internal tables operate in the same way as structures. The main difference being that structures only have one line, while an internal table can have as many lines as required.

An internal table can be made up of a number of fields, corresponding to the columns of a table. just as in the ABAP dictionary a table was created using a number of fields. Key fields can be

be used with internal tables, and while creating these internal tables they offer slightly more flexibility. With internal tables, one can specify a non-unique key, allowing any number of non-unique records to be stored, and allowing duplicate records to be stored if required.

The size of an internal table or the number of lines it contains is not fixed. The size of an internal table changes according to the requirement of the program associated with the internal table. But it is recommended to keep internal tables as small as possible. This is to avoid the system running slowly as it struggles to process enormous amounts of data.

Internal tables are used for many purposes –

- They can be used to hold results of calculations that could be used later in the program.
- An internal table can also hold records and data so that this can be accessed quickly rather than having to access this data from database tables.
- They are hugely versatile. They can be defined using any number of other defined structures.

Example

Assume that a user wants to create a list of contact numbers of various customers from one or several large tables. The user first creates an internal table, selects the relevant data from customer tables and then places the data in the internal table. Other users can access and use this internal table directly to retrieve the desired information, instead of writing database queries to perform each operation during the run-time of the program.

SAP ABAP - Creating Internal Tables

DATA statement is used to declare an internal table. The program must be told where the table begins and ends. So use the BEGIN OF statement and then declare the table name. After this, the OCCURS addition is used, followed by a number, here 0. OCCURS tells SAP that an internal table is being created, and the 0 states that it will not contain any records initially. It will then expand as it is filled with data.

Following is the syntax –

```
DATA: BEGIN OF <internal_tab> Occurs 0,
```

Let's create the fields on a new line. For instance, create 'name' which is declared as LIKE ZCUSTOMERS1-name. Create another field called 'dob', LIKE ZCUSTOMERS1-dob. It is useful initially to give the field names in internal tables the same names as other fields that have been created elsewhere. Finally, declare the end of the internal table with "END OF <internal_tab>" as shown in the following code –

```
DATA: BEGIN OF itab01 Occurs 0,
      name LIKE ZCUSTOMERS1-name,
      dob LIKE ZCUSTOMERS1-dob,
    END OF itab01.
```

Aqui, 'itab01' é uma abreviação comumente usada ao criar tabelas temporárias no SAP. A cláusula OCCURS é usada para definir o corpo de uma tabela interna declarando os campos da tabela. Quando a cláusula OCCURS é usada, você pode especificar uma constante numérica 'n' para determinar a memória padrão adicional, se necessário. O tamanho padrão de memória usado pela cláusula OCCURS é 8 KB. A estrutura da tabela interna agora está criada e o código pode ser escrito para preenchê-la com registros.

Uma tabela interna pode ser criada com ou sem linha de cabeçalho. Para criar uma tabela interna com uma linha de cabeçalho, use a cláusula BEGIN OF antes da cláusula OCCURS ou a cláusula WITH HEADER LINE após a cláusula OCCURS na definição da tabela interna. Para criar uma tabela interna sem linha de cabeçalho, use a cláusula OCCURS sem a cláusula BEGIN OF.

Você também pode criar uma tabela interna como um tipo de dados local (um tipo de dados usado somente no contexto do programa atual) usando a instrução TYPES. Esta instrução usa a cláusula TYPE ou LIKE para se referir a uma tabela existente.

A sintaxe para criar uma tabela interna como tipo de dados local é -

```
TYPES <internal_tab> TYPE|LIKE <internal_tab_type> OF
      <line_type_itab> WITH <key> INITIAL SIZE <size_number>.
```

Aqui <internal_tab_type> especifica um tipo de tabela para uma tabela interna <internal_tab> e <line_type_itab> especifica o tipo de uma linha de uma tabela interna. Na instrução TYPES, você pode usar a cláusula TYPE para especificar o tipo de linha de uma tabela interna como um tipo de dados e a cláusula LIKE para especificar o tipo de linha como um objeto de dados. A especificação de uma chave para uma tabela interna é opcional e se o usuário não especificar uma chave, o sistema SAP define um tipo de tabela com uma chave arbitrária.

INITIAL SIZE <size_number> cria um objeto de tabela interno alocando uma quantidade inicial de memória para ele. Na sintaxe anterior, a cláusula INITIAL SIZE reserva um espaço de memória para linhas da tabela size_number. Sempre que um objeto de tabela interno é declarado, o tamanho da tabela não pertence ao tipo de dados da tabela.

Note - Muito menos memória é consumida quando uma tabela interna é preenchida pela primeira vez.

Exemplo

Step 1 - Abra o Editor ABAP executando o código de transação SE38. A tela inicial do ABAP Editor é exibida.

Step 2 - Na tela inicial, insira um nome para o programa, selecione o botão de opção Código fonte e clique no botão Criar para criar um novo programa.

Step 3 – In the 'ABAP: Program Attributes' dialog box, enter a short description for the program in the Title field, select the 'Executable program' option from the Type drop-down menu in the Attributes group box. Click the Save button.

Step 4 – Write the following code in ABAP editor.

```
REPORT ZINTERNAL_DEMO.
TYPES: BEGIN OF CustomerLine,
        Cust_ID TYPE C,
        Cust_Name(20) TYPE C,
      END OF CustomerLine.

TYPES mytable TYPE SORTED TABLE OF CustomerLine
WITH UNIQUE KEY Cust_ID.
WRITE:/ 'The mytable is an Internal Table'.
```

Step 5 – Save, activate and execute the program as usual.

In this example, mytable is an internal table and a unique key is defined on the Cust_ID field.

The above code produces the following output –

The mytable is an Internal Table.

SAP ABAP - Populating Internal Tables

In internal tables, populating includes features such as selection, insertion and append. This chapter focuses on INSERT and APPEND statements.

INSERT Statement

INSERT statement is used to insert a single line or a group of lines into an internal table.

Following is the syntax to add a single line to an internal table –

```
INSERT <work_area_itab> INTO <internal_tab> INDEX <index_num>.
```

In this syntax, the INSERT statement inserts a new line in the internal_tab internal table. A new line can be inserted by using the work_area_itab INTO expression before the internal_tab parameter. When the work_area_itab INTO expression is used, the new line is taken from the work_area_itab work area and inserted into the internal_tab table. However, when the work_area_itab INTO expression is not used to insert a line, the line is taken from the header line of the internal_tab table.

When a new line is inserted in an internal table by using the INDEX clause, the index number of the lines after the inserted line is incremented by 1. If an internal table contains <index_num> - 1 lines, the new line is added at the end of the table. When the SAP system successfully adds a line to an internal table, the SY-SUBRC variable is set to 0.

Example

Following is a sample program that uses the insert statement.

```

REPORT ZCUSLIST1.

DATA: BEGIN OF itable1 OCCURS 4,
      F1 LIKE SY-INDEX,
      END OF itable1.

DO 4 TIMES.
  itable1-F1 = sy-index.
  APPEND itable1.
ENDDO.

itable1-F1 = -96.
INSERT itable1 INDEX 2.

LOOP AT itable1.
  Write / itable1-F1.
ENDLOOP.

LOOP AT itable1 WHERE F1 ≥ 3.
  itable1-F1 = -78.
  INSERT itable1.
ENDLOOP.

Skip.
LOOP AT itable1.
  Write / itable1-F1.
ENDLOOP.

```

The above code produces the following output –

```

1
96-
2
3
4
1
96-
2
78-
3
78-
4

```

In the above example, the DO loop appends 4 rows containing the numbers 1 through 4 to it. The header line component itable1-F1 has been assigned a value of -96. Insert statement inserts the header line as new row into the body before row 3. The existing row 3 becomes row 4 after the insert. The LOOP AT statement retrieves those rows from the internal table that have an F1 value greater than or equal to 3. Before each row, Insert statement inserts a new row from the header line of it. Prior to the insert, the F1 component has been changed to contain -78.

After each insert statement is executed, the system re-indexes all rows below the one inserted. This introduces overhead when you insert rows near the top of a large internal table. If you need to insert a block of rows into a large internal table, prepare another table with the rows to be inserted and use insert lines instead.

When inserting a new row inside itable1 inside of a loop at itable1, it doesn't affect the internal table instantly. It actually becomes effective on the next loop pass. While inserting a row after the current row, the table is re-indexed at the ENDLOOP. The sy-tabix is incremented and the next loop processes the row pointed to by sy-tabix. For instance, if you are in the second loop pass and you insert a record before row 3. When endloop is executed, the new row becomes row 3 and the old row 3 becomes row 4 and so on. Sy-tabix is incremented by 1, and the next loop pass processes the newly inserted record.

APPEND Statement

The APPEND statement is used to add a single row or line to an existing internal table. This statement copies a single line from a work area and inserts it after the last existing line in an internal table. The work area can be either a header line or any other field string with the same structure as a line of an internal table. Following is the syntax of the APPEND statement that is used to append a single line in an internal table –

```
APPEND <record_for_itab> TO <internal_tab>.
```

In this syntax, the <record_for_itab> expression can be represented by the <work_area_itab> work area, which is convertible to a line type or by the INITIAL LINE clause. If the user uses a <work_area_itab> work area, the SAP system adds a new line to the <internal_tab> internal table and populates it with the content of the work area. The INITIAL LINE clause appends a blank line that contains the initial value for each field of the table structure. After each APPEND statement, the SY-TABIX variable contains the index number of the appended line.

Appending lines to standard and sorted tables with a non-unique key works regardless of whether the lines with the same key already exist in the table. In other words, duplicate entries may occur. However, a run-time error occurs if the user attempts to add a duplicate entry to a sorted table with a unique key or if the user violates the sort order of a sorted table by appending the lines to it.

Example

```
REPORT ZCUSLIST1.

DATA: BEGIN OF linv Occurs 0,
      Name(20) TYPE C,
      ID_Number TYPE I,
    END OF linv.

DATA table1 LIKE TABLE OF linv.
linv-Name = 'Melissa'.
linv-ID_Number = 105467.
APPEND linv TO table1.
LOOP AT table1 INTO linv.

Write: / linv-name, linv-ID_Number.
ENDLOOP.
```

The above code produces the following output –

Melissa	105467
---------	--------

SAP ABAP - Copying Internal Tables

When we read a record from an internal table with a header line, that record is moved from the table itself into the header line. It is then the header line that our program works with. The same applies while creating a new record. It is the header line with which you work with and from which the new record is sent to the table body itself.

To copy the records, we can use a SELECT statement to select all of the records from the table and then use MOVE statement that will move the records from the original table into the new internal table into the fields where the names correspond.

Following is the syntax for MOVE statement –

```
MOVE <table_field> TO <internal_tab_field>.
```

Example

```
REPORT ZCUSLIST1.
TABLES: ZCUSTOMERS1.
DATA: BEGIN OF itab01 Occurs 0,
      name LIKE ZCUSTOMERS1-name,
      dob LIKE ZCUSTOMERS1-dob,
END OF itab01.
```

```
Select * FROM ZCUSTOMERS1.
MOVE ZCUSTOMERS1-name TO itab01-name.
MOVE ZCUSTOMERS1-dob TO itab01-dob.
ENDSELECT.
```

```
Write: / itab01-name, itab01-dob.
```

The above code produces the following output –

MARGARET	02.11.1994
----------	------------

The select loop fills each field one at a time, using the MOVE statement to move the data from one table's field to the other. In the above example, MOVE statements were used to move the contents of the ZCUSTOMERS1 table to the corresponding fields in the internal table. You can accomplish this action with just one line of code. You can use the MOVECORRESPONDING statement.

Following is the syntax for MOVE-CORRESPONDING statement –

```
MOVE-CORRESPONDING <table_name> TO <internal_tab>.
```

It tells the system to move the data from the fields of ZCUSTOMERS1 to their corresponding fields in itab01.

Example

```

REPORT ZCUSTOMERLIST.
TABLES: ZCUSTOMERS1.
DATA: Begin of itab01 occurs 0,
      customer LIKE ZCUSTOMERS1-customer,
      name LIKE ZCUSTOMERS1-name,
      title LIKE ZCUSTOMERS1-title,
      dob LIKE ZCUSTOMERS1-dob,
END OF itab01.

SELECT * from ZCUSTOMERS1.
MOVE-Corresponding ZCUSTOMERS1 TO itab01.
APPEND itab01.
ENDSELECT.
LOOP AT itab01.
Write: / itab01-name, itab01-dob.
ENDLOOP.

```

The above code produces the following output –

MARK	21.05.1981
JAMES	14.08.1977
AURIELE	19.06.1990
STEPHEN	22.07.1985
MARGARET	02.11.1994

This is made possible by the fact that both have matching field names. When making use of this statement, you need to make sure that both fields have matching data types and lengths. It has been done here with the LIKE statement previously.

SAP ABAP - Reading Internal Tables

We can read the lines of a table by using the following syntax of the READ TABLE statement –

```
READ TABLE <internal_table> FROM <work_area_itab>.
```

In this syntax, the <work_area_itab> expression represents a work area that is compatible with the line type of the <internal_table> table. We can specify a search key, but not a table key, within the READ statement by using the WITH KEY clause, as shown in the following syntax –

```
READ TABLE <internal_table> WITH KEY = <internal_tab_field>.
```

Here the entire line of the internal table is used as a **search key**. The content of the entire line of the table is compared with the content of the <internal_tab_field> field. If the values of the <internal_tab_field> field are not compatible with the line type of the table, these values are converted according to the line type of the table. The search key allows you to find entries in internal tables that do not have a structured line type, that is, where the line is a single field or an internal table type.

The following syntax of the READ statement is used to specify a work area or field symbol by using the COMPARING clause –

```
READ TABLE <internal_table> <key> INTO <work_area_itab>
[COMPARING <F1> <F2>...<Fn>].
```

When the COMPARING clause is used, the specified table fields <F1>, <F2>....<Fn> of the structured line type are compared with the corresponding fields of the work area before being transported. If the ALL FIELDS clause is specified, the SAP system compares all the components. When the SAP system finds an entry on the basis of a key, the value of the SY-SUBRC variable is set to 0. In addition, the value of the SY-SUBRC variable is set to 2 or 4 if the content of the compared fields is not the same or if the SAP system cannot find an entry. However, the SAP system copies the entry into the target work area whenever it finds an entry, regardless of the result of the comparison.

Example

```
REPORT ZREAD_DEMO.

*/Creating an internal table
DATA: BEGIN OF Record1,
      ColP TYPE I,
      ColQ TYPE I,
      END OF Record1.

DATA mytable LIKE HASHED TABLE OF Record1 WITH UNIQUE KEY ColP.
DO 6 Times.
  Record1-ColP = SY-INDEX.
  Record1-ColQ = SY-INDEX + 5.
  INSERT Record1 INTO TABLE mytable.
ENDDO.

Record1-ColP = 4.
Record1-ColQ = 12.

READ TABLE mytable FROM Record1 INTO Record1 COMPARING ColQ.

WRITE: 'SY-SUBRC =', SY-SUBRC.
```

```
SKIP.
```

```
WRITE: / Record1-ColP, Record1-ColQ.
```

The above code produces the following output –

```
SY-SUBRC = 2
```

```
4 9
```

In the above example, mytable is an internal table of the hashed table type, with Record1 as the work area and ColP as the unique key. Initially, mytable is populated with six lines, where the ColP field contains the values of the SY-INDEX variable and the ColQ field contains (SY-INDEX + 5) values.

The Record1 work area is populated with 4 and 12 as values for the ColP and ColQ fields respectively. The READ statement reads the line of the table after comparing the value of the ColP key field with the value in the Record1 work area by using the COMPARING clause, and then copies the content of the read line in the work area. The value of the SY-SUBRC variable is displayed as 2 because when the value in the ColP field is 4, the value in the ColQ is not 12, but 9.

SAP ABAP - Deleting Internal Tables

The DELETE statement is used to delete one or more records from an internal table. The records of an internal table are deleted either by specifying a table key or condition or by finding duplicate entries. If an internal table has a non-unique key and contains duplicate entries, the first entry from the table is deleted.

Following is the syntax to use the DELETE statement to delete a record or line from an internal table –

```
DELETE TABLE <internal_table> FROM <work_area_itab>.
```

In the above syntax, the <work_area_itab> expression is a work area and it should be compatible with the type of the <internal_table> internal table. The delete operation is performed on the basis of a default key that could be taken from the work area components.

You may also specify a table key explicitly in the DELETE TABLE statement by using the following syntax –

```
DELETE TABLE <internal_table> WITH TABLE KEY <K1> = <F1>..... <Kn> = <Fn>.
```

In this syntax, <F1>, <F2>....<Fn> are the fields of an internal table and <K1>, <K2>....<Kn> are the key fields of the table. The DELETE statement is used to delete the records or lines of the <internal_table> table based on the expressions <K1> = <F1>, <K2> = <F2>...<Kn> = <Fn>.

Note – If the data types of the <F1>, <F2>....<Fn> fields are not compatible with the <K1>, <K2>...<Kn> key fields then the SAP system automatically converts them into the compatible format.

Example

```

REPORT ZDELETE_DEMO.

DATA: BEGIN OF Line1,
      ColP TYPE I,
      ColQ TYPE I,
    END OF Line1.

DATA mytable LIKE HASHED TABLE OF Line1
WITH UNIQUE KEY ColP.

DO 8 TIMES.

  Line1-ColP = SY-INDEX.
  Line1-ColQ = SY-INDEX + 4.
  INSERT Line1 INTO TABLE mytable.
ENDDO.

Line1-ColP = 1.
DELETE TABLE mytable: FROM Line1,
WITH TABLE KEY ColP = 3.
LOOP AT mytable INTO Line1.

WRITE: / Line1-ColP, Line1-ColQ.
ENDLOOP.

```

The above code produces the following output –

2	6
4	8
5	9
6	10
7	11
8	12

In this example, mytable has two fields, ColP and ColQ. Initially, mytable is populated with eight lines, where the ColP contains the values 1, 2, 3, 4, 5, 6, 7 and 8. The ColQ contains the values 5, 6, 7, 8, 9, 10, 11 and 12 because the ColP values are incremented by 4 every time.

The DELETE statement is used to delete the lines from mytable where the value of the ColP key field is either 1 or 3. After deletion, the ColP field of mytable contains the values 2, 4, 5, 6, 7 and 8, as shown in the output. The ColQ field contains the values 6, 8, 9, 10, 11 and 12.

SAP ABAP - Object Orientation

Object orientation simplifies software design to make it easier to understand, maintain, and reuse. **Object Oriented Programming (OOP)** represents a different way of thinking in writing software. The beauty of OOP lies in its simplicity. The expressiveness of OOP makes it easier to deliver quality software components on time.

As solutions are designed in terms of real-world objects, it becomes much easier for programmers and business analysts to exchange ideas and information about a design that uses a common domain language. These improvements in communication help to reveal hidden requirements, identify risks, and improve the quality of software being developed. The object-oriented approach focuses on objects that represent abstract or concrete things of the real world. These objects are defined by their character and properties that are represented by their internal structure and their attributes (data). The behavior of these objects is described by methods (i.e. functionality).

Let's compare the procedural and object oriented programming –

Features	Procedure Oriented approach	Object Oriented approach
Emphasis	Emphasis is on tasks.	Emphasis is on things that does those tasks.
Modularization	Programs can be divided into smaller programs known as functions.	Programs are organized into classes and objects and the functionalities are embedded into methods of a class.
Data security	Most of the functions share global data.	Data can be hidden and can't be accessed by external sources.
Extensibility	This is more time consuming to modify and extend the existing functionality.	New data and functions can be added effortlessly as and when required.

ABAP was initially developed as a procedural language (just similar to earlier procedural programming language like COBOL). But ABAP has now adapted the principles of object oriented paradigms with the introduction of ABAP Objects. The object-oriented concepts in ABAP such as class, object, inheritance, and polymorphism, are essentially the same as those of other modern object-oriented languages such as Java or C++.

As object orientation begins to take shape, each class assumes specific role assignments. This division of labor helps to simplify the overall programming model, allowing each class

specialize in solving a particular piece of the problem at hand. Such classes have high cohesion and the operations of each class are closely related in some intuitive way.

The key features of object orientation are –

- Effective programming structure.
- Real-world entities can be modeled very well.
- Stress on data security and access.
- Minimizes code redundancy.
- Data abstraction and encapsulation.

SAP ABAP - Objects

An object is a special kind of variable that has distinct characteristics and behaviors. The characteristics or attributes of an object are used to describe the state of an object, and behaviors or methods represent the actions performed by an object.

An object is a pattern or instance of a class. It represents a real-world entity such as a person or a programming entity like variables and constants. For example, accounts and students are examples of real-world entities. But hardware and software components of a computer are examples of programming entities.

An object has the following three main characteristics –

- Has a state.
- Has a unique identity.
- May or may not display the behavior.

The state of an object can be described as a set of attributes and their values. For example, a bank account has a set of attributes such as Account Number, Name, Account Type, Balance, and values of all these attributes. The behavior of an object refers to the changes that occur in its attributes over a period of time.

Each object has a unique identity that can be used to distinguish it from other objects. Two objects may exhibit the same behavior and they may or may not have the same state, but they never have the same identity. Two persons may have the same name, age, and gender but they are not identical. Similarly, the identity of an object will never change throughout its lifetime.

Objects can interact with one another by sending messages. Objects contain data and code to manipulate the data. An object can also be used as a user-defined data type with the help of a class. Objects are also called variables of the type class. After defining a class, you can create any number of objects belonging to that class. Each object is associated with the data of the type class with which it has been created.

Creating an Object

The object creation usually includes the following steps –

- Creating a reference variable with reference to the class. The syntax for which is –

```
DATA: <object_name> TYPE REF TO <class_name>.
```

- Creating an object from the reference variable. The syntax for which is –

```
CREATE Object: <object_name>.
```

Example

```
REPORT ZDEMO_OBJECT.

CLASS Class1 Definition.
  Public Section.
    DATA: text1(45) VALUE 'ABAP Objects.'.
    METHODS: Display1.
  ENDCLASS.

  CLASS Class1 Implementation.
    METHOD Display1.
      Write:/ 'This is the Display method.'.
    ENDMETHOD.
  ENDCLASS.

  START-OF-SELECTION.
    DATA: Class1 TYPE REF TO Class1.
    CREATE Object: Class1.
    Write:/ Class1->text1.
    CALL METHOD: Class1->Display1.
```

The above code produces the following output –

```
ABAP Objects.
This is the Display method.
```

SAP ABAP - Classes

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called **members of the class**.

Class Definition and Implementation

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, what an object of the class will consist of, and what operations can be performed on such an object. That is, it defines the abstract characteristics of an object, such as attributes, fields, and properties.

The following syntax shows how to define a class –

```
CLASS <class_name> DEFINITION.  
.....  
.....  
ENDCLASS.
```

A class definition starts with the keyword CLASS followed by the class name, DEFINITION and the class body. The definition of a class can contain various components of the class such as attributes, methods, and events. When we declare a method in the class declaration, the method implementation must be included in the class implementation. The following syntax shows how to implement a class –

```
CLASS <class_name> IMPLEMENTATION.  
.....  
.....  
ENDCLASS.
```

Note – Implementation of a class contains the implementation of all its methods. In ABAP Objects, the structure of a class contains components such as attributes, methods, events, types, and constants.

Attributes

Attributes are data fields of a class that can have any data type such as C, I, F, and N. They are declared in the class declaration. These attributes can be divided into 2 categories: instance and static attributes. An **instance attribute** defines the instance specific state of an object. The states are different for different objects. An instance attribute is declared by using the DATA statement.

Static attributes define a common state of a class that is shared by all the instances of the class. That is, if you change a static attribute in one object of a class, the change is visible to all other objects of the class as well. A static attribute is declared by using the CLASS-DATA statement.

Methods

A method is a function or procedure that represents the behavior of an object in the class. The methods of the class can access any attribute of the class. The definition of a method can also contain parameters, so that you can supply the values to these parameters when methods are called. The definition of a method is declared in the class declaration and implemented in the implementation part of a class. The METHOD and ENDMETHOD statements are used to define the implementation part of a method. The following syntax shows how to implement a method

```
—  
METHOD <m_name>.  
.....  
.....  
ENDMETHOD.
```

In this syntax, <m_name> represents the name of a method. **Note** – You can call a method by using the CALL METHOD statement.

Accessing Attributes and Methods

Class components can be defined in public, private, or protected visibility sections that control how these components could be accessed. The private visibility section is used to deny access to components from outside of the class. Such components can only be accessed from inside the class such as a method.

Components defined in the public visibility section can be accessed from any context. By default all the members of a class would be private. Practically, we define data in private section and related methods in public section so that they can be called from outside of the class as shown in the following program.

- The attributes and methods declared in Public section in a class can be accessed by that class and any other class, sub-class of the program.
- When the attributes and methods are declared in Protected section in a class, those can be accessed by that class and sub classes (derived classes) only.
- When the attributes and methods are declared in Private section in a class, those can be accessed by only that class and not by any other class.

Example

```

Report ZAccess1.

CLASS class1 Definition.
  PUBLIC Section.
    Data: text1 Type char25 Value 'Public Data'.
    Methods meth1.

  PROTECTED Section.
    Data: text2 Type char25 Value 'Protected Data'.

  PRIVATE Section.
    Data: text3 Type char25 Value 'Private Data'.
ENDCLASS.

CLASS class1 Implementation.
  Method meth1.
    Write: / 'Public Method:',  

           / text1,  

           / text2,  

           / text3.

    Skip.
  EndMethod.
ENDCLASS.

Start-Of-Selection.
  Data: Objectx Type Ref To class1.
  Create Object: Objectx.
  CALL Method: Objectx->meth1.
  Write: / Objectx->text1.

```

The above code produces the following output –

```

Public Method:  

Public Data  

Protected Data  

Private Data

Public Data

```

Static Attributes

A Static attribute is declared with the statement CLASS-DATA. All the objects or instances can use the static attribute of the class. Static attributes are accessed directly with the help of cl

```
name like class_name⇒name_1 = 'Some Text'.
```

Example

Following is a program where we want to print a text with line number 4 to 8 times. We define a class class1 and in the public section we declare CLASS-DATA (static attribute) and a method. After implementing the class and method, we directly access the static attribute in Start-Of-Selection event. Then we just create the instance of the class and call the method.

```
Report ZStatic1.

CLASS class1 Definition.
  PUBLIC Section.
    CLASS-DATA: name1 Type char45,
                 data1 Type I.
    Methods: meth1.
  ENDCLASS.

  CLASS class1 Implementation.
    Method meth1.
      Do 4 Times.
        data1 = 1 + data1.
        Write: / data1, name1.
      EndDo.
      Skip.
    EndMethod.
  ENDCLASS.

  Start-Of-Selection.
    class1⇒name1 = 'ABAP Object Oriented Programming'.
    class1⇒data1 = 0.
    Data: Object1 Type Ref To class1,
          Object2 Type Ref To class1.

    Create Object: Object1, Object2.
    CALL Method: Object1→meth1,
                Object2→meth1.
```

The above code produces the following output –

<i>Static Attributes</i>	
Static Attributes	
1	ABAP Object Oriented Programming
2	ABAP Object Oriented Programming
3	ABAP Object Oriented Programming
4	ABAP Object Oriented Programming
5	ABAP Object Oriented Programming
6	ABAP Object Oriented Programming
7	ABAP Object Oriented Programming
8	ABAP Object Oriented Programming

Constructors

Constructors are special methods that are called automatically, either while creating an object or accessing the components of a class. Constructor gets triggered whenever an object is created, but we need to call a method to trigger the general method. In the following example, we have declared two public methods method1 and constructor. Both these methods have different operations. While creating an object of the class, the constructor method triggers its operation.

Example

```

Report ZConstructor1.
CLASS class1 Definition.
  PUBLIC Section.
    Methods: method1, constructor.
  ENDCLASS.

  CLASS class1 Implementation.
    Method method1.
      Write: / 'This is Method1'.
    EndMethod.

    Method constructor.
      Write: / 'Constructor Triggered'.
    EndMethod.
  ENDCLASS.

  Start-Of-Selection.
    Data Object1 Type Ref To class1.
    Create Object Object1.
  
```

The above code produces the following output –

Constructor Triggered

ME Operator in Methods

When you declare a variable of any type in public section of a class, you can use it in any other implementation. A variable can be declared with an initial value in public section. We may declare the variable again inside a method with a different value. When we write the variable inside the method, the system will print the changed value. To reflect the previous value of the variable, we have to use 'ME' operator.

In this program, we have declared a public variable text1 and initiated with a value. We have declared the same variable again, but instantiated with different value. Inside the method, we are writing that variable with 'ME' operator to get the previously initiated value. We get the changed value by declaring directly.

Example

```
Report ZMEOperator1.
CLASS class1 Definition.
  PUBLIC Section.

  Data text1 Type char25 Value 'This is CLASS Attribute'.
  Methods method1.
ENDCLASS.

CLASS class1 Implementation.
  Method method1.

    Data text1 Type char25 Value 'This is METHOD Attribute'.
    Write: / ME→text1,
           / text1.
  ENDMETHOD.

ENDCLASS.

Start-Of-Selection.
  Data objectx Type Ref To class1.
  Create Object objectx.
  CALL Method objectx→method1.
```

The above code produces the following output –

```
This is CLASS Attribute  
This is METHOD Attribute
```

SAP ABAP - Inheritance

One of the most important concepts in object oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and methods, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base class or super class**, and the new class is referred to as the **derived class or sub class**.

- An object of one class can acquire the properties of another class.
- Derived class inherits the data and methods of a super class. However, they can overwrite methods and also add new methods.
- The main advantage of inheritance is reusability.

The inheritance relationship is specified using the 'INHERITING FROM' addition to the class definition statement.

Following is the syntax –

```
CLASS <subclass> DEFINITION INHERITING FROM <superclass>.
```

Example

```
Report ZINHERITAN_1.  
CLASS Parent Definition.  
PUBLIC Section.  
Data: w_public(25) Value 'This is public data'.  
Methods: ParentM.  
ENDCLASS.
```

```
CLASS Child Definition Inheriting From Parent.  
PUBLIC Section.  
Methods: ChildM.  
ENDCLASS.
```

```
CLASS Parent Implementation.
```

```
Method ParentM.
```

```
Write /: w_public.
```

```
EndMethod. ENDCLASS.
```

```
CLASS Child Implementation.
```

```
Method ChildM.
```

```
Skip.
```

```
Write /: 'Method in child class', w_public.
```

```
EndMethod.
```

```
ENDCLASS.
```

Start-of-selection.

Data: Parent Type Ref To Parent,

Child Type Ref To Child.

Create Object: Parent, Child.

Call Method: Parent->ParentM,

child->ChildM.

The above code produces the following output –

This is public data

Method in child class

This is public data

Access Control and Inheritance

A derived class can access all the non-private members of its base class. Thus super class members that should not be accessible to the member functions of sub classes should be declared private in the super class. We can summarize the different access types according to who can access them in the following way –

Access	Public	Protected	Private
Same class	Yes	Yes	Yes
Derived class	Yes	Yes	No
Outside class	Yes	No	No

When deriving a class from a super class, it can be inherited through public, protected or private inheritance. The type of inheritance is specified by the access specifier as explained above. We hardly use protected or private inheritance, but public inheritance is commonly used. The following rules are applied while using different types of inheritance.

- **Public Inheritance** – When deriving a class from a public super class, public members of the super class become public members of the sub class and protected members of the super class become protected members of the sub class. Super class's private members are never accessible directly from a sub class, but can be accessed through calls to the public and protected members of the super class.
- **Protected Inheritance** – When deriving from a protected super class, public and protected members of the super class become protected members of the sub class.
- **Private Inheritance** – When deriving from a private super class, public and protected members of the super class become private members of the sub class.

Redefining Methods in Sub Class

The methods of the super class can be re-implemented in the sub class. Few rules of redefining methods –

- The redefinition statement for the inherited method must be in the same section as the definition of the original method.
- If you redefine a method, you do not need to enter its interface again in the subclass, but only the name of the method.
- Within the redefined method, you can access components of the direct super class using the super reference.
- The pseudo reference super can only be used in redefined methods.

Example

```
Report Zinheri_Redefine.
CLASS super_class Definition.
  Public Section.
    Methods: Addition1 importing g_a TYPE I
               g_b TYPE I
               exporting g_c TYPE I.
  ENDCLASS.

  CLASS super_class Implementation.
    Method Addition1.
      g_c = g_a + g_b.
    EndMethod.
  ENDCLASS.

  CLASS sub_class Definition Inheriting From super_class.
    Public Section.
```

METHODS: **Addition1 Redefinition.**

ENDCLASS.

CLASS sub_class **Implementation.**

Method Addition1.

g_c = g_a + g_b + 10.

EndMethod.

ENDCLASS.

Start-Of-Selection.

Parameters: P_a Type I, P_b TYPE I.

Data: H_Addition1 TYPE I.

Data: H_Sub TYPE I.

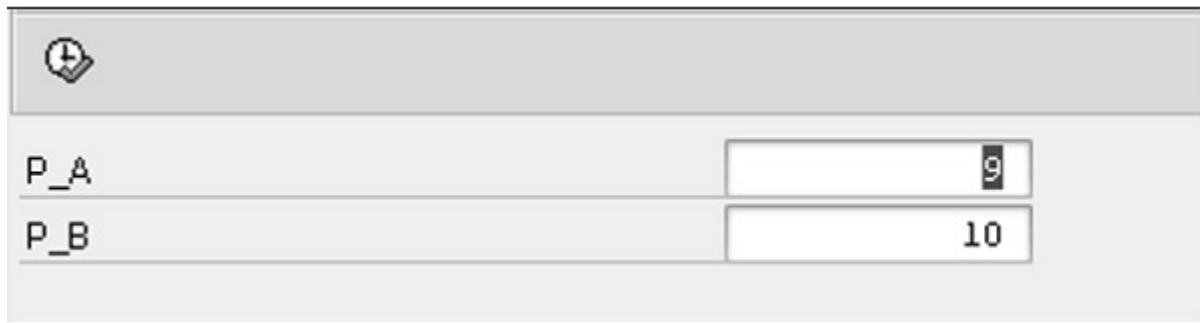
Data: Ref1 TYPE Ref TO sub_class.

Create Object Ref1.

Call Method Ref1->Addition1 exporting g_a = P_a
g_b = P_b
Importing g_c = H_Addition1.

Write:/ H_Addition1.

After executing F8, if we enter the values 9 and 10, the above code produces the following output –



Redefinition Demo

29

SAP ABAP - Polymorphism

The term polymorphism literally means ‘many forms’. From an object-oriented perspective, polymorphism works in conjunction with inheritance to make it possible for various types within an inheritance tree to be used interchangeably. That is, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. ABAP polymorphism means that a call to a method will cause a different method to be executed depending on the type of object that invokes the method.

The following program contains an abstract class 'class_prgm', 2 sub classes (class_procedural and class_OO), and a test driver class 'class_type_approach'. In this implementation, the class method 'start' allow us to display the type of programming and its approach. If you look closely at the signature of method 'start', you will observe that it receives an importing parameter of type class_prgm. However, in the Start-Of-Selection event, this method has been called at run-time with objects of type class_procedural and class_OO.

Example

```

Report ZPolymorphism1.
CLASS class_prgm Definition Abstract.
PUBLIC Section.
Methods: prgm_type Abstract,
approach1 Abstract.
ENDCLASS.

CLASS class_procedural Definition
Inheriting From class_prgm.
PUBLIC Section.
Methods: prgm_type Redefinition,
approach1 Redefinition.
ENDCLASS.

CLASS class_procedural Implementation.
Method prgm_type.
Write: 'Procedural programming'.

EndMethod. Method approach1.
Write: 'top-down approach'.

EndMethod. ENDCLASS.

CLASS class_OO Definition
Inheriting From class_prgm.
PUBLIC Section.
Methods: prgm_type Redefinition,
approach1 Redefinition.
ENDCLASS.

CLASS class_OO Implementation.
Method prgm_type.
Write: 'Object oriented programming'.

EndMethod.

Method approach1.

```

```
Write: 'bottom-up approach'.
```

```
EndMethod.
```

```
ENDCLASS.
```

```
CLASS class_type_approach Definition.
```

```
PUBLIC Section.
```

```
CLASS-METHODS:
```

```
start Importing class1_prgm
```

```
Type Ref To class_prgm.
```

```
ENDCLASS.
```

```
CLASS class_type_approach IMPLEMENTATION.
```

```
Method start.
```

```
CALL Method class1_prgm→prgm_type.
```

```
Write: 'follows'.
```

```
CALL Method class1_prgm→approach1.
```

```
EndMethod.
```

```
ENDCLASS.
```

Start-Of-Selection.

Data: class_1 **Type Ref To** class_procedural,

class_2 **Type Ref To** class_00.

Create Object class_1.

Create Object class_2.

CALL **Method** class_type_approach→start

Exporting

class1_prgm = class_1.

New-Line.

CALL **Method** class_type_approach→start

Exporting

class1_prgm = class_2.

The above code produces the following output –

Procedural programming follows top-down approach

Object oriented programming follows bottom-up approach

ABAP run-time environment performs an implicit narrowing cast during the assignment of the importing parameter class1_prgm. This feature helps the 'start' method to be implemented generically. The dynamic type information associated with an object reference variable allows the ABAP run-time environment to dynamically bind a method call with the implementation defined in the object pointed to by the object reference variable. For instance, the import

parameter 'class1_prgm' for method 'start' in the 'class_type_approach' class refers to an abstract type that could never be instantiated on its own.

Whenever the method is called with a concrete sub class implementation such as class_procedural or class_OO, the dynamic type of the class1_prgm reference parameter is bound to one of these concrete types. Therefore, the calls to methods 'prgm_type' and 'approach1' refer to the implementations provided in the class_procedural or class_OO sub classes rather than the undefined abstract implementations provided in class 'class_prgm'.

SAP ABAP - Encapsulation

Encapsulation is an Object Oriented Programming (OOP) concept that binds together data and functions that manipulate the data, and keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding. Encapsulation is a mechanism of bundling the data and the functions that use them, and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

ABAP supports the properties of encapsulation and data hiding through the creation of user-defined types called classes. As discussed earlier, a class can contain private, protected and public members. By default, all items defined in a class are private.

Encapsulation by Interface

Encapsulation actually means one attribute and method could be modified in different classes. Hence data and method can have different form and logic that can be hidden to separate class.

Let's consider encapsulation by interface. Interface is used when we need to create one method with different functionality in different classes. Here the name of the method need not be changed. The same method will have to be implemented in different class implementations.

Example

The following program contains an Interface inter_1. We have declared attribute and a method method1. We have also defined two classes like Class1 and Class2. So we have to implement the method 'method1' in both of the class implementations. We have implemented the method 'method1' differently in different classes. In the start-ofselection, we create two objects Object1 and Object2 for two classes. Then, we call the method by different objects to get the function declared in separate classes.

```
Report ZEncap1.  
Interface inter_1.  
  Data text1 Type char35.  
  Methods method1.  
EndInterface.
```

```

CLASS Class1 Definition.
  PUBLIC Section.
    Interfaces inter_1.
  ENDCLASS.

CLASS Class2 Definition.
  PUBLIC Section.
    Interfaces inter_1.
  ENDCLASS.

CLASS Class1 Implementation.
  Method inter_1~method1.
    inter_1~text1 = 'Class 1 Interface method'.
    Write / inter_1~text1.
  EndMethod.
ENDCLASS.

CLASS Class2 Implementation.
  Method inter_1~method1.
    inter_1~text1 = 'Class 2 Interface method'.
    Write / inter_1~text1.
  EndMethod.
ENDCLASS.

Start-Of-Selection.
  Data: Object1 Type Ref To Class1,
        Object2 Type Ref To Class2.

  Create Object: Object1, Object2.
  CALL Method: Object1->inter_1~method1,
            Object2->inter_1~method1.

```

The above code produces the following output –

```

Class 1 Interface method
Class 2 Interface method

```

Encapsulated classes do not have a lot of dependencies on the outside world. Moreover, the interactions that they do have with external clients are controlled through a stabilized public interface. That is, an encapsulated class and its clients are loosely coupled. For the most part, classes with well-defined interfaces can be plugged into another context. When designed correctly, encapsulated classes become reusable software assets.

Designing Strategy

Most of us have learned through bitter experience to make class members private by default unless we really need to expose them. That is just good encapsulation. This wisdom is applied most frequently to data members and it also applies equally to all members.

SAP ABAP - Interfaces

Similar to classes in ABAP, interfaces act as data types for objects. The components of interfaces are same as the components of classes. Unlike the declaration of classes, the declaration of an interface does not include the visibility sections. This is because the components defined in the declaration of an interface are always integrated in the public visibility section of the classes.

Interfaces are used when two similar classes have a method with the same name, but the functionalities are different from each other. Interfaces might appear similar to classes, but the functions defined in an interface are implemented in a class to extend the scope of that class. Interfaces along with the inheritance feature provide a base for polymorphism. This is because a method defined in an interface can behave differently in different classes.

Following is the general format to create an interface –

```
INTERFACE <intf_name>.  
DATA.....  
CLASS-DATA.....  
METHODS.....  
CLASS-METHODS.....  
ENDINTERFACE.
```

In this syntax, <intf_name> represents the name of an interface. The DATA and CLASSDATA statements can be used to define the instance and static attributes of the interface respectively. The METHODS and CLASS-METHODS statements can be used to define the instance and static methods of the interface respectively. As the definition of an interface does not include the implementation class, it is not necessary to add the DEFINITION clause in the declaration of an interface.

Note – All the methods of an interface are abstract. They are fully declared including their parameter interface, but not implemented in the interface. All the classes that want to use an interface must implement all the methods of the interface. Otherwise, the class becomes an abstract class.

We use the following syntax in the implementation part of the class –

```
INTERFACE <intf_name>.
```

In this syntax, <intf_name> represents the name of an interface. Note that this syntax must be used in the public section of the class.

The following syntax is used to implement the methods of an interface inside the implementation of a class –

```
METHOD <intf_name~method_m>.  
<statements>.  
ENDMETHOD.
```

In this syntax, <intf_name~method_m> represents the fully declared name of a method of the <intf_name> interface.

Example

```
Report ZINTERFACE1.  
INTERFACE my_interface1.  
Methods msg.  
ENDINTERFACE.  
  
CLASS num_counter Definition.  
PUBLIC Section.  
INTERFACES my_interface1.  
Methods add_number.  
PRIVATE Section.  
Data num Type I.  
ENDCLASS.  
  
CLASS num_counter Implementation.  
Method my_interface1~msg.  
Write: / 'The number is', num.  
EndMethod.  
  
Method add_number.  
ADD 7 TO num.  
EndMethod.  
ENDCLASS.  
  
CLASS drive1 Definition.  
PUBLIC Section.  
INTERFACES my_interface1.  
Methods speed1.
```

```

PRIVATE Section.
Data wheel1 Type I.
ENDCLASS.

CLASS drive1 Implementation.
Method my_interface1~msg.
Write: / 'Total number of wheels is', wheel1.
EndMethod.

Method speed1.
Add 4 To wheel1.
EndMethod.
ENDCLASS.

Start-Of-Selection.
Data object1 Type Ref To num_counter.
Create Object object1.

CALL Method object1->add_number.
CALL Method object1->my_interface1~msg.

Data object2 Type Ref To drive1.
Create Object object2.

CALL Method object2->speed1.
CALL Method object2->my_interface1~msg.

```

The above code produces the following output –

```

The number is 7
Total number of wheels is 4

```

In the above example, my_interface1 is the name of an interface that contains the 'msg' method. Next, two classes, num_counter and drive1 are defined and implemented. Both these classes implement the 'msg' method and also specific methods that define the behavior of their respective instances, such as the add_number and speed1 methods.

Note – The add_number and speed1 methods are specific to the respective classes.

SAP ABAP - Object Events

An **event** is a set of outcomes that are defined in a class to trigger the event handlers in other classes. When an event is triggered, we can call any number of event handler methods. The link between a trigger and its handler method is actually decided dynamically at run-time.

In a normal method call, a calling program determines which method of an object or a class needs to be called. As fixed handler method is not registered for every event, in case of event handling, the handler method determines the event that needs to be triggered.

An event of a class can trigger an event handler method of the same class by using the RAISE EVENT statement. For an event, the event handler method can be defined in the same or different class by using the FOR EVENT clause, as shown in the following syntax –

```
FOR EVENT <event_name> OF <class_name>.
```

Similar to the methods of a class, an event can have parameter interface but it has only output parameters. The output parameters are passed to the event handler method by the RAISE EVENT statement that receives them as input parameters. An event is linked to its handler method dynamically in a program by using the SET HANDLER statement.

When an event is triggered, appropriate event handler methods are supposed to be executed in all the handling classes.

Example

```
REPORT ZEVENT1.

CLASS CL_main DEFINITION.
PUBLIC SECTION.
DATA: num1 TYPE I.
METHODS: PRO IMPORTING num2 TYPE I.
EVENTS: CUTOFF.
ENDCLASS.

CLASS CL_eventhandler DEFINITION.
PUBLIC SECTION.
METHODS: handling_CUTOFF FOR EVENT CUTOFF OF CL_main.
ENDCLASS.

START-OF-SELECTION.
DATA: main1 TYPE REF TO CL_main.
DATA: eventhandler1 TYPE REF TO CL_eventhandler.

CREATE OBJECT main1.
CREATE OBJECT eventhandler1.

SET HANDLER eventhandler1->handling_CUTOFF FOR main1.
main1->PRO( 4 ).
CLASS CL_main IMPLEMENTATION.
METHOD PRO.
num1 = num2.
```

```

IF num2 ≥ 2.
RAISE EVENT CUTOFF.
ENDIF.
ENDMETHOD.
ENDCLASS.

CLASS CL_eventhandler IMPLEMENTATION.
METHOD handling_CUTOFF.
WRITE: 'Handling the CutOff'.
WRITE: / 'Event has been processed'.
ENDMETHOD. ENDCLASS.

```

The above code produces the following output –

```

Handling the CutOff
Event has been processed

```

SAP ABAP - Report Programming

A **report** is a presentation of data in an organized structure. Many database management systems include a report writer that enables you to design and generate reports. SAP applications support report creation.

A classical report is created by using the output data in the WRITE statement inside a loop. They do not contain any sub-reports. SAP also provides some standard reports such as RSCLTCOP that is used to copy tables across clients and RSPARAM that is used to display instance parameters.

These reports consist of only one screen as an output. We can use various events such as INITIALIZATON & TOP-OF-PAGE to create a classical report, and each event has its own importance during the creation of a classical report. Each of these events is associated to a specific user action and is triggered only when the user performs that action.

Following is a table describing the events and descriptions –

S.No.	Event & Description
1	INITIALIZATION Triggered before displaying the selection screen.
2	AT SELECTION-SCREEN Triggered after processing of the user input on the selection screen. This event verifies the user input prior to the execution of a program. After processing the user input, the selection screen remains in the active mode.
3	START-OF-SELECTION Triggered only after the processing of the selection screen is over; that is, when the user clicks the Execute icon on the selection screen.
4	END-OF-SELECTION Triggered after the last statement in the START-OF-SELECTON event is executed.
5	TOP-OF-PAGE Triggered by the first WRITE statement to display the data on a new page.
6	END-OF-PAGE Triggered to display the text at the end of a page in a report. Note, that this event is the last event while creating a report, and should be combined with the LINE-COUNT clause of the REPORT statement.

Example

Let's create a classical report. We will display the information stored in the standard database MARA (contains general material data) by using a sequence of statements in ABAP editor.

```

REPORT ZREPORT2
LINE-SIZE 75
LINE-COUNT 30(3)
NO STANDARD PAGE HEADING.
Tables: MARA.
TYPES: Begin of itab,

```

```
MATNR TYPE MARA-MATNR,
MBRSH TYPE MARA-MBRSH,
MEINS TYPE MARA-MEINS,
MTART TYPE MARA-MTART,
```

End of itab.

```
DATA: wa_ma TYPE itab,
      it_ma TYPE STANDARD TABLE OF itab.
```

```
SELECT-OPTIONS: MATS FOR MARA-MATNR OBLIGATORY.
```

```
INITIALIZATION.
```

```
MATS-LOW = '1'.
```

```
MATS-HIGH = '500'.
```

```
APPEND MATS.
```

```
AT SELECTION-SCREEN. .
```

```
IF MATS-LOW = ' '.
```

```
MESSAGE I000(ZKMESSAGE).
```

```
ELSEIF MATS-HIGH = ' '.
```

```
MESSAGE I001(ZKMESSAGE).
```

```
ENDIF.
```

```
TOP-OF-PAGE.
```

```
WRITE:/ 'CLASSICAL REPORT CONTAINING GENERAL MATERIAL DATA
FROM THE TABLE MARA' COLOR 7.
```

```
ULINE.
```

```
WRITE:/ 'MATERIAL' COLOR 1,
```

```
24 'INDUSTRY' COLOR 2,
```

```
38 'UNITS' COLOR 3,
```

```
53 'MATERIAL TYPE' COLOR 4.
```

```
ULINE.
```

```
END-OF-PAGE.
```

```
START-OF-SELECTION.
```

```
SELECT MATNR MBRSH MEINS MTART FROM MARA
INTO TABLE it_ma WHERE MATNR IN MATS.
```

```
LOOP AT it_ma into wa_ma.
```

```
WRITE:/ wa_ma-MATNR,
```

```
25 wa_ma-MBRSH,
```

```
40 wa_ma-MEINS,
```

```
55 wa_ma-MTART.
```

```
ENDLOOP.
```

```
END-OF-SELECTION.
```

```
ULINE.
```

```
WRITE:/ 'CLASSICAL REPORT HAS BEEN CREATED' COLOR 7.
```

```
ULINE.
```

```
SKIP.
```

The above code produces the following output containing the general material data from the standard table MARA –

CLASSICAL REPORT CONTAINING GENERAL MATERIAL DATA FROM THE TABLE MARA			
MATERIAL	INDUSTRY	UNITS	MATERIAL TYPE
23	I	EA	ROH
38	M	PC	HALB
43	I	HR	HAWA
58	M	PC	HIBE
59	M	PC	HIBE
68	M	PC	FHMI
78	M	PC	DIEN
88	M	PC	FERT
89	M	PC	FERT
98	M	PC	HALB
170	M	PC	NLAG
178	M	PC	NLAG
188	M	PC	NLAG
288	M	PC	HALB
358	M	PC	HAWA
359	M	PC	HAWA

CLASSICAL REPORT HAS BEEN CREATED

SAP ABAP - Dialog Programming

Dialog programming deals with the development of multiple objects. All these objects are linked hierarchically to the main program and they are executed in a sequence. Dialog program development makes use of tools in the ABAP workbench. These are the same tools used in standard SAP application development.

Here are the main components of dialog programs –

- Screens
- Module pools
- Subroutines
- Menus
- Transactions

The Toolset

Central Component	Tool	Transaction
All Components	Object Browser	SE80 Tools > ABAP Workbench > Object Browser
Screen	Screen Painter	SE51 Tools > ABAP Workbench > Screen Painter
ABAP/4 Module Pool	ABAP/4 Editor	SE38 Tools > ABAP Workbench > ABAP/4 Editor
Dictionary Objects (tables, fields, etc.)	ABAP/4 Dictionary	SE11 Tools > ABAP Workbench > ABAP/4 Dictionary
Menu	Menu Painter	SE41 Tools > ABAP Workbench > Menu Painter
Transaction	Maintain Transaction	SE93 Tools > ABAP Workbench > Development > Other Tools > Transactions

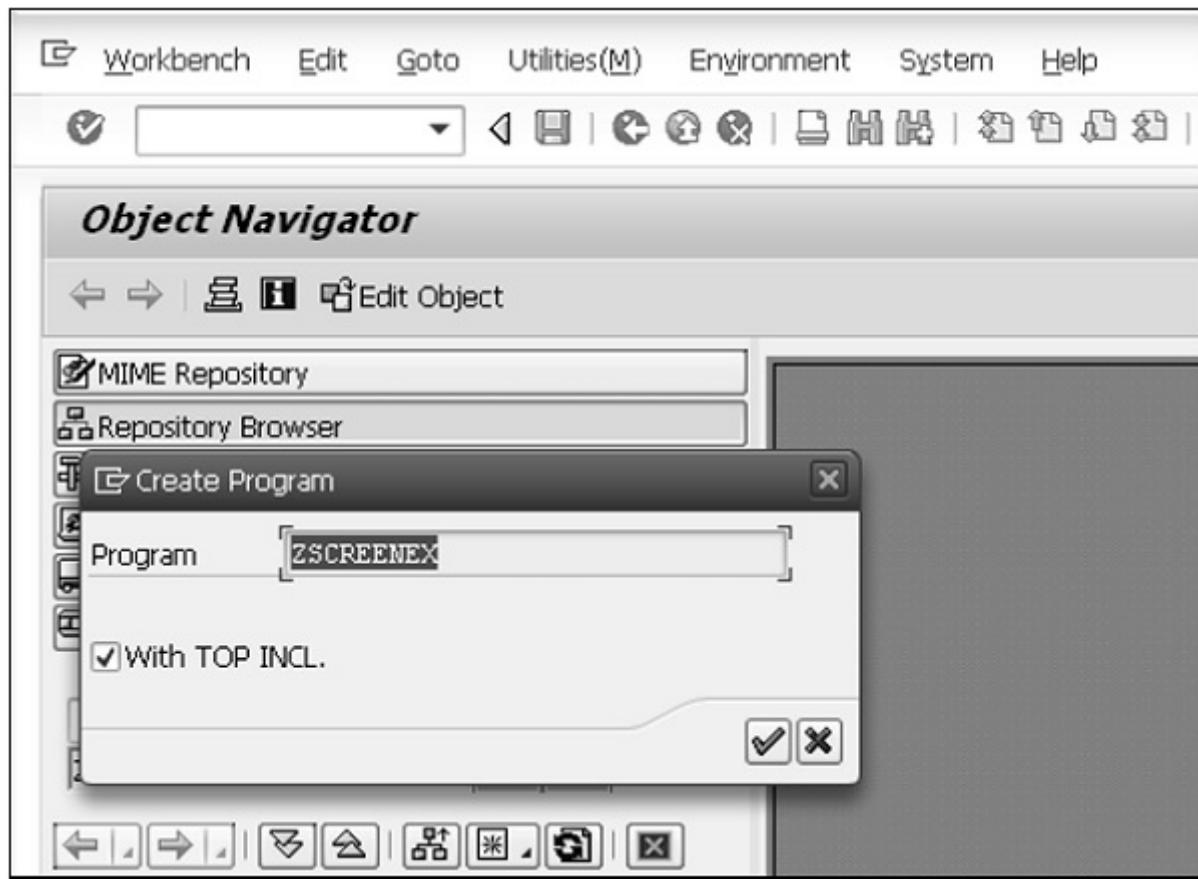
Dialog programs should be developed by the object browser (transaction: SE80) so that all objects become linked to the main program without having to explicitly point each object. Advanced navigation techniques enhance the process of moving from one object to the other.

Screens are made up of screen attributes, screen layout, fields and flow logic. The module pool consists of modularized syntax that is placed inside include programs of the dialog program. These modules can be invoked by the flow logic, which is processed by the dialog processor.

Creating a New Dialog Program

Step 1 – Within the transaction SE80, select ‘Program’ from the dropdown and enter a Z name for your custom SAP program as ‘ZSCREENEX’.

Step 2 – Press Enter, choose ‘With TOP INCL’ and click the ‘Yes’ button.



Step 3 – Enter a name for your top include as 'ZSCRTOP' and click the green tick mark.

Step 4 – Within the attributes screen, simply enter a title and click the save button.

Adding a Screen to the Dialog Program

Step 1 – To add a screen to the program, right-click on the program name and select the options Create → Screen.

Step 2 – Enter a screen number as '0211' and click the green tick mark.

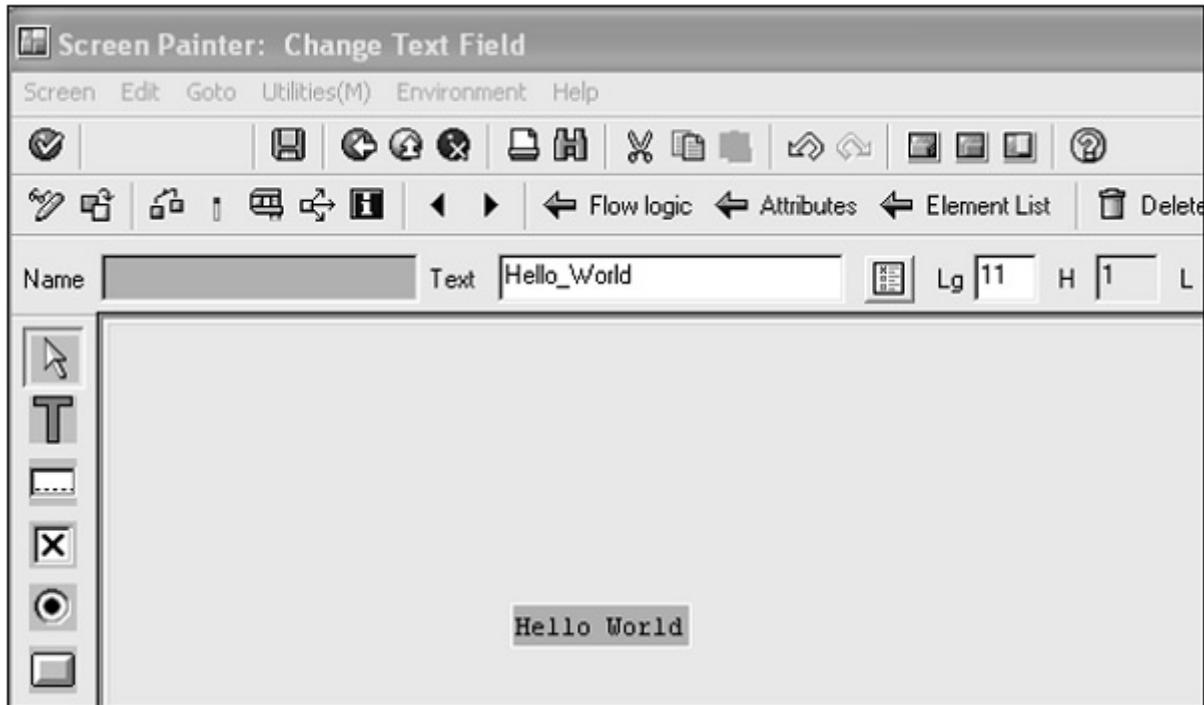
Screen number	211	New(Revised)
Attributes		
Short Description	Adding a screen to dialog program	
Original Language	EN English	Package
Last changed on/at		00:00:00
Last Generation		00:00:00
Screen Type		Settings
<input checked="" type="radio"/> Normal <input type="radio"/> Subscreen <input type="radio"/> Modal dialog box <input type="radio"/> Selection screen		<input type="checkbox"/> Hold Data <input type="checkbox"/> Switch Off Runtime Compress <input type="checkbox"/> Template - non-executable <input type="checkbox"/> Hold Scroll Position <input type="checkbox"/> Without Application Toolbar

Step 3 – In the next screen, enter a short title, set to normal screen type and click the save button on the top application toolbar.

Screen Layout and Adding ‘Hello World’ Text

Step 1 – Click the layout button within the application toolbar and the Screen Painter window appears.

Step 2 – Add a Text Field and enter some text such as "Hello World".



Step 3 – Save and activate the screen.

Creating Transaction

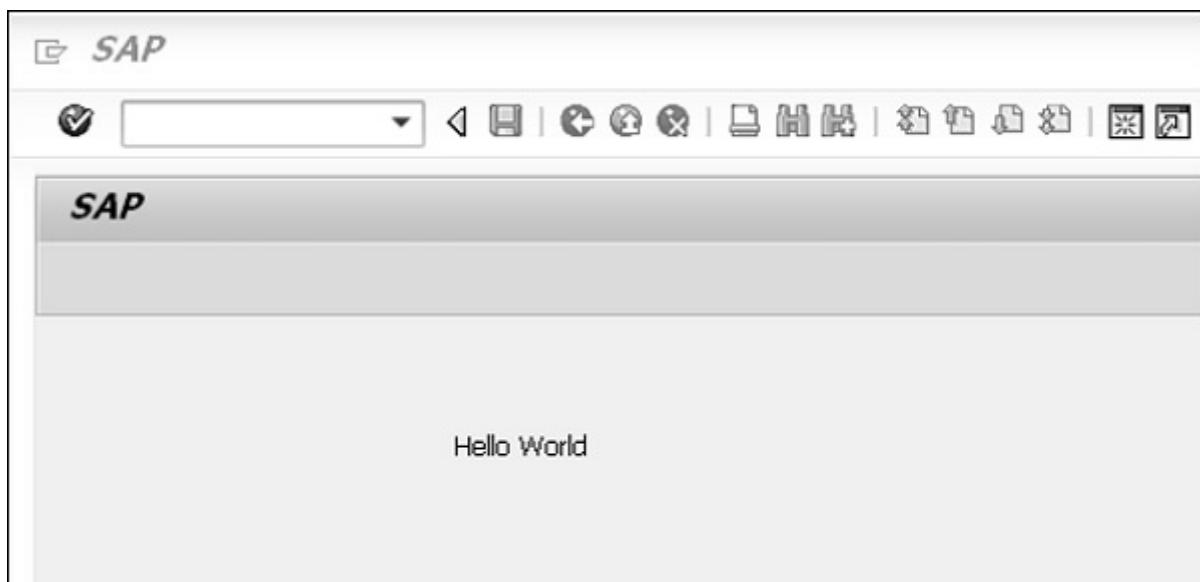
Step 1 – To create a transaction code for your program, simply right click on the program name and choose the option Create → Transaction and enter a transaction code as 'ZTRANEX'.

Transaction code	ZTRANEX
Package	
Transaction text	Creating Transaction
Program	ZSCREENEX
Screen number	0211
Authorization Object	<input type="text"/> Values
<input checked="" type="checkbox"/> Maintenance of standard transaction variant allowed	
Classification	
Transaction classification	
<input checked="" type="radio"/> Professional User Transaction	
<input type="radio"/> Easy Web Transaction	
<input type="checkbox"/> Pervasive enabled	
Service <input type="text"/>	
GUI support	
<input type="checkbox"/> SAPGUI for HTML	
<input type="checkbox"/> SAPGUI for Java	
<input checked="" type="checkbox"/> SAPGUI for Windows	

Step 2 – Enter the transaction text, program and screen you have just created (ZSCREENEX & 0211), and tick the ‘SAPGUI for Windows’ checkbox in the ‘GUI support’ section.

Executing the Program

Save and activate everything. You can execute the program. As the program executes, the text you entered is displayed on the screen as shown in the following screenshot.



SAP ABAP - Smart Forms

SAP Smart Forms tool can be used to print and send documents. This tool is useful in developing forms, PDF files, e-mails and documents for the Internet. The tool provides an interface to build and maintain the layout and logic of a form. SAP also delivers a selection of forms for business processes such as those used in Customer Relationship Management (CRM), Sales and Distribution (SD), Financial Accounting (FI) and Human Resources (HR).

The tool allows you to modify forms by using simple graphical tools instead of using any programming tool. It means that a user with no programming knowledge can configure these forms with data for a business process effortlessly.

In a Smart Form, data is retrieved from static and dynamic tables. The table heading and subtotal are specified by the triggered events and the data is then sorted before the final output. A Smart Form allows you to incorporate graphics that can be displayed either as part of the form or as the background. You can also suppress a background graphic if required while taking a printout of a form.

Some examples of standard Smart Forms available in SAP system are as follows –

- SF_EXAMPLE_01 represents an invoice with a table output for flight booking for a customer.
- SF_EXAMPLE_02 represents an invoice similar to SF_EXAMPLE_01, but with subtotals.
- SF_EXAMPLE_03 specifies an invoice similar to SF_EXAMPLE_02, but one in which several customers can be selected in an application program.

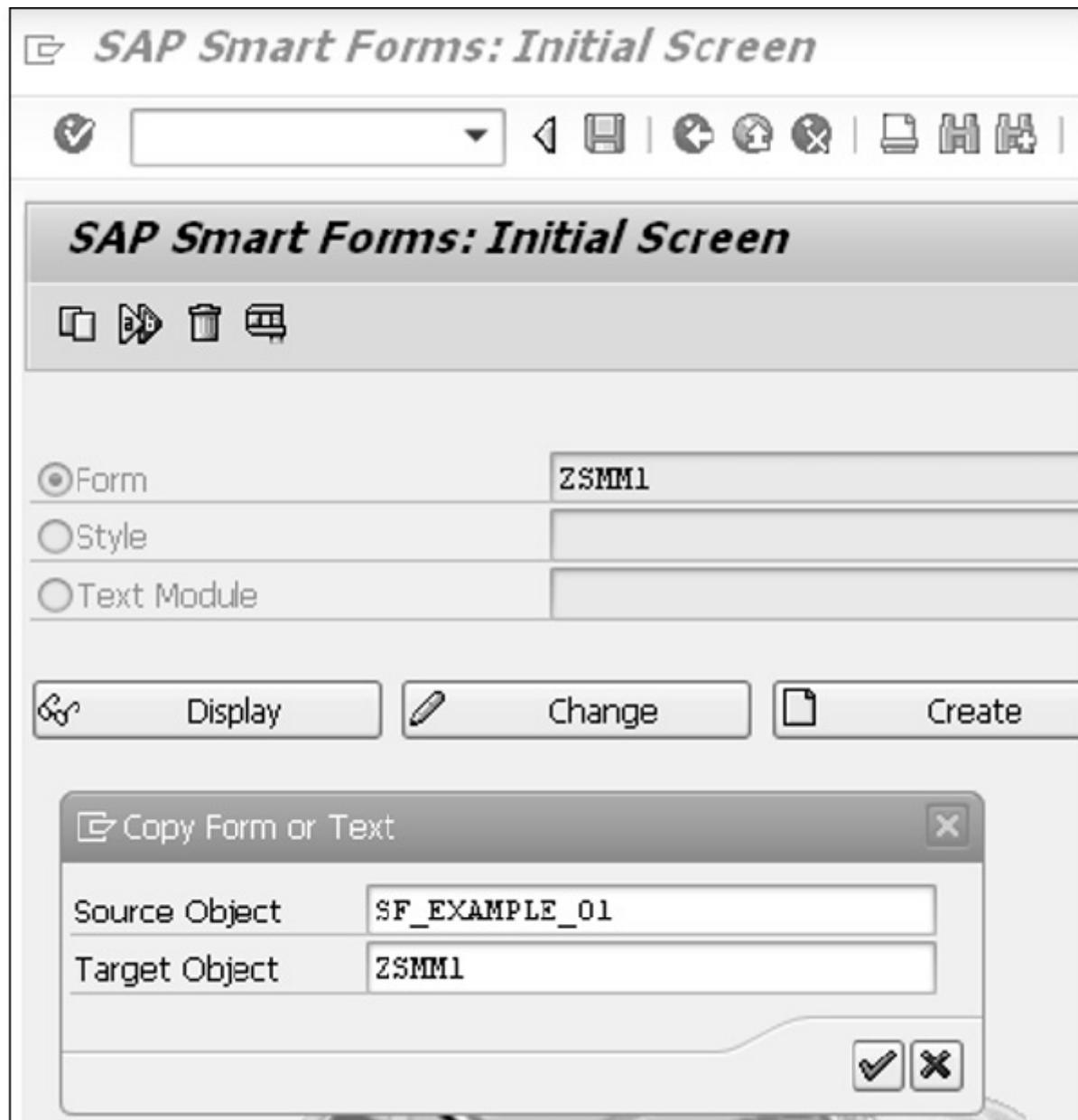
Creating a Form

Let's create a form by using the SAP Smart Forms tool. You will also learn how to add a node in the Smart Form and test the form in this tutorial. Here we begin with creating a copy of the SF_EXAMPLE_01 form. The SF_EXAMPLE_01 form is a standard Smart Form available in the SAP system.

Step 1 – Smart Form Builder is the main interface used to build a Smart Form. It is available on the initial screen of SAP Smart Forms. We need to type the 'SMARTFORMS' transaction code in the Command field to open the initial screen of SAP Smart Forms. In this screen, enter the form name, SF_EXAMPLE_01, in the Form field.

Step 2 – Select Smart Forms → Copy or click the Copy icon to open the Copy Form or Text dialog box.

Step 3 – In the Target Object field, enter a name for the new form. The name must begin with the Y or Z letter. In this case, the name of the form is 'ZSMM1'.

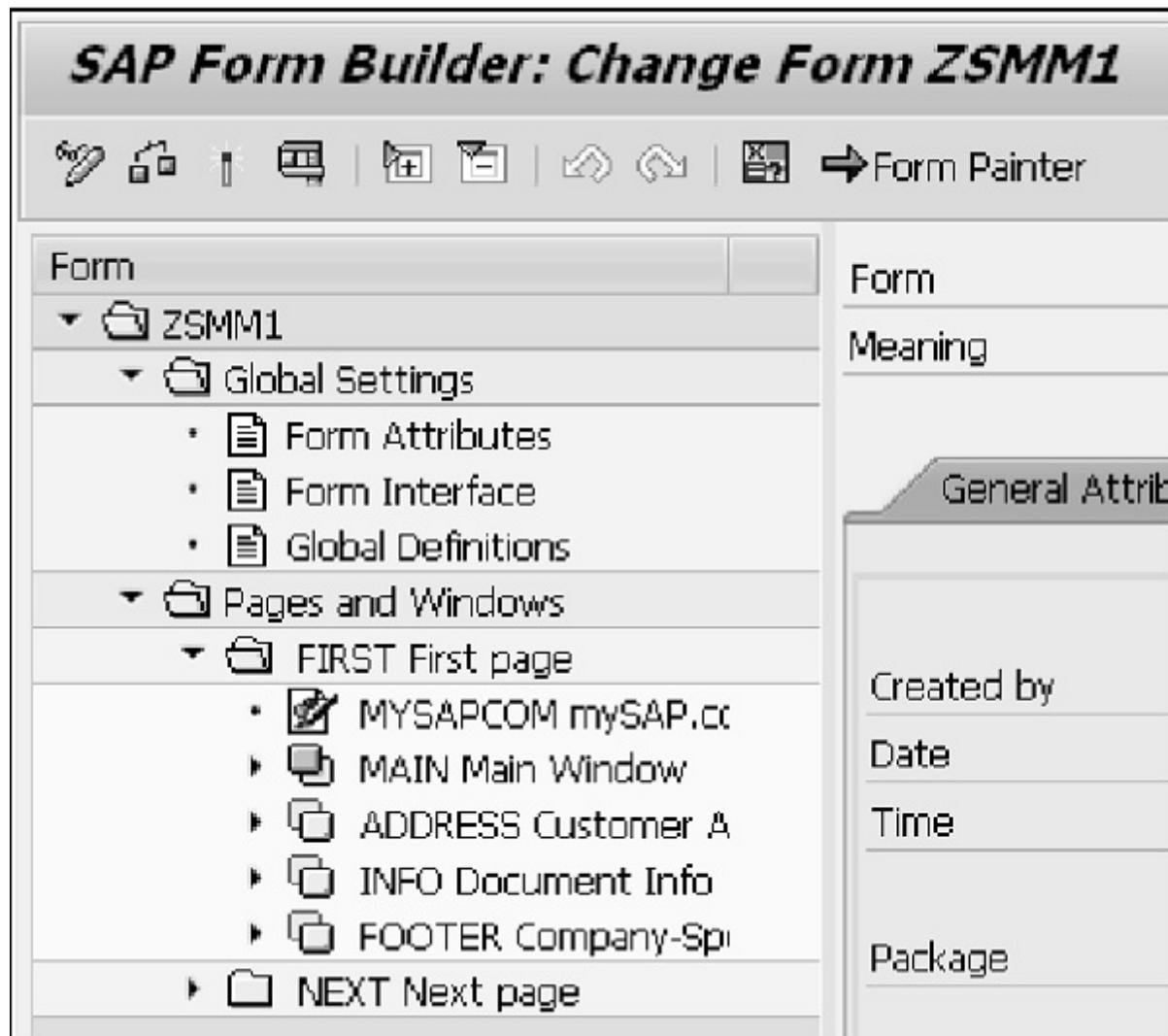


Step 4 – Click the Continue icon or press the ENTER key in the Copy Form or Text dialog box so that the ZSMM1 form is created as a copy of the predefined form SF_EXAMPLE_01.

Step 5 – Click the Save icon. The name of the form is displayed in the Form field on the initial screen of SAP Smart Forms.

Step 6 – Click the Create button on the initial screen of SAP Smart Forms. The ZSMM1 form appears in Form Builder.

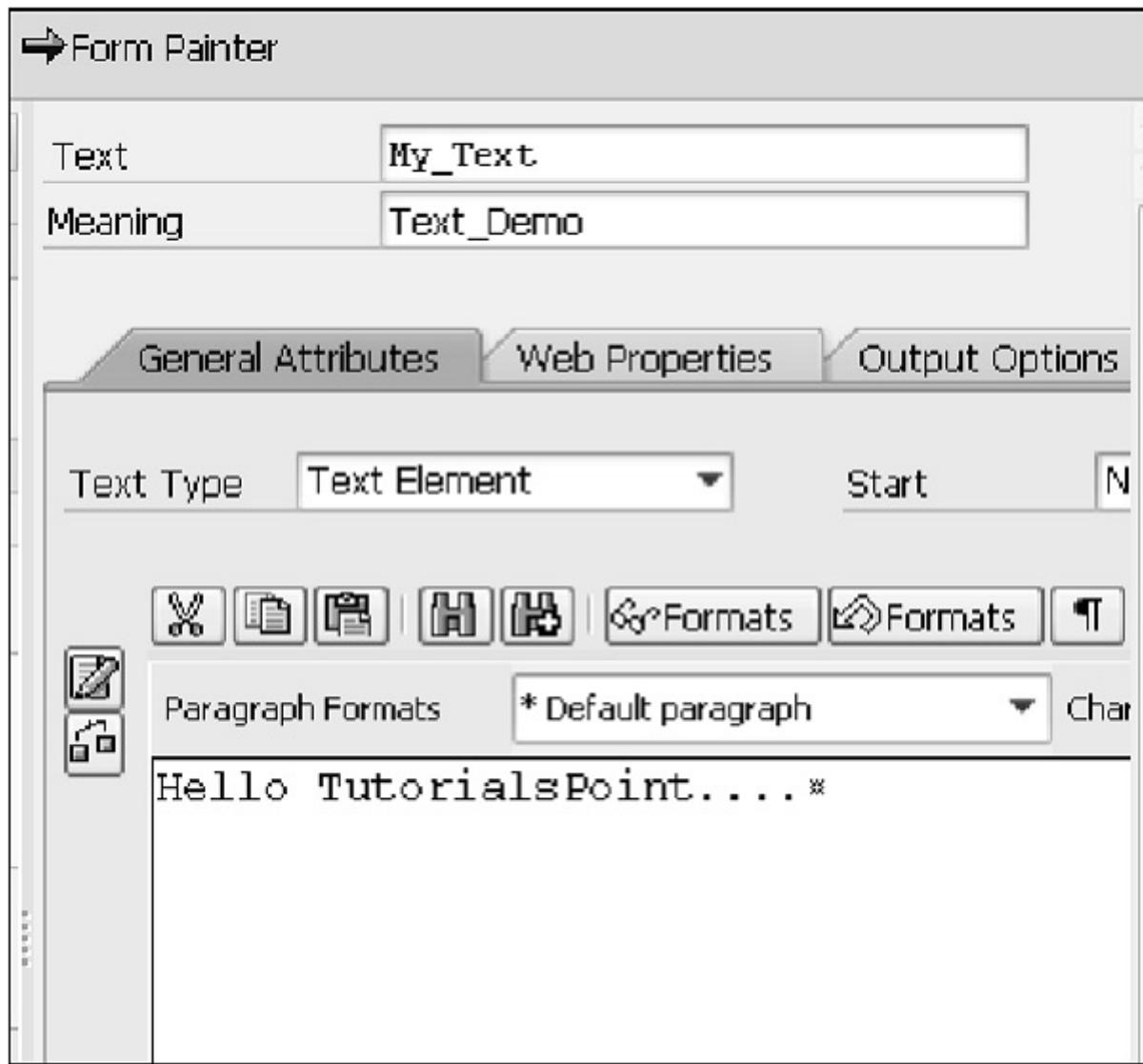
Step 7 – The first draft page is created with a MAIN window. All the components of the new form are based on the SF_EXAMPLE_01 predefined form. You can just click a node in the Navigation menu to view its content.



Creating a Text Node in the Form

Step 1 – Open a form in the change mode of the SAP Form Builder screen and right-click the Main Window option in the First Page node and select Create → Text from the context menu.

Step 2 – Modify the text in the Text field to 'My_Text' and the text in the Meaning field to 'Text_Demo'. Enter the text 'Hello TutorialsPoint.....' in the text-editing box in the center frame of Form Builder as shown in the following snapshot –



Step 3 – Click the Save button to save the node..

Step 4 – Activate and test the node by clicking the Activate and Test icons, respectively. The initial screen of Function Builder appears.

Step 5 – Activate and test the function module by clicking the Activate and Execute icons. The parameters of the function module are displayed in the initial screen of Function Builder.

Step 6 – Execute the function module by clicking the Execute icon. The Print dialog box appears.

Step 7 – Specify the output device as 'LP01' and click the Print preview button.

The above steps will produce the following output –

<u>IDES Holding Co. P.O. Box 9999, Cognac, CIV: XYY, SYY, United Kingdom</u>																	
Invoice																	
Administrative clerk	Mr. Jones																
Telephone	+1/2 12/99 10 99																
Telefax	+1/2 12/99 12 99																
Signed	39999 / 2000																
Customer number	00000000																
Date	21. 11.2015																
<p>Hello TutorialsPoint....</p> <p>Dear Sir or Madam,</p> <p>We would appreciate payment of the following invoice as soon as possible. Thank you for placing your confidence in us.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Car Line</th> <th>Flight</th> <th>Departure</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td>rie</td> <td>Date</td> <td></td> <td></td> </tr> <tr> <td>x</td> <td></td> <td></td> <td></td> </tr> <tr> <td colspan="2">Total</td> <td></td> <td></td> </tr> </tbody> </table> <p>Yours sincerely, IDES HOLDING AG</p>		Car Line	Flight	Departure	Price	rie	Date			x				Total			
Car Line	Flight	Departure	Price														
rie	Date																
x																	
Total																	

SAP ABAP - SAPscripts

The SAPscript tool of the SAP system can be used to build and manage business forms such as invoices and purchase orders. The SAPscript tool provides numerous templates that simplify the designing of a business form to a great extent.

The SAP system comes with standard SAPscript forms that are delivered with the SAP standard client (generally as client 000). Following are a few examples of standard SAPscript forms delivered with client 000 –

S.No.	Form Name & Description
1	RVORDER01 Sales Order Confirmation Form
2	RVDELNOTE Packing List
3	RVINVOICE01 Invoice
4	MEDRUCK Purchase Order
5	F110_PRENUM_CHCK Prenumbered Check

The structure of a SAPscript form consists of 2 main components –

Content – This can be either text (business data) or graphics (company logo).

Layout – This is defined by a set of windows in which the form content appears.

SAPscript – Form Painter Tool

The Form Painter tool provides the graphical layout of a SAPscript form and various functionalities to manipulate the form. In the following example, we are going to create an invoice form after copying its layout structure from a standard SAPscript form RVINVOICE01, and display its layout by accessing the Form Painter tool.

Step 1 – Open the Form Painter. You may request the screen either by navigating the SAP menu or by using the SE71 transaction code.

Step 2 – In the Form Painter, request screen, enter a name and language for a SAPscript form in the Form and Language fields, respectively. Let's enter 'RVINVOICE01' and 'EN' respectively in these fields.

Form RVINVOICE01 Language EN

Subobjects

- Header
- Page Layout
- Paragraph Formats
- Character Formats
- Documentation

Display Change

Step 3 – Select the Page Layout radio button in the Sub objects group box.

Step 4 – Select Utilities → Copy from Client to create a copy of the RVINVOICE01 form. The 'Copy Forms Between Clients' screen appears.

Step 5 – In the 'Copy Forms Between Clients' screen, enter the original name of the form, 'RVINVOICE01', in the Form Name field, the number of the source client '000' in the Source Client field, and the name of the target form as 'ZINV_01' in the Target Form field. Make sure that other settings remain unchanged.

Form Name RVINVOICE01

Source Client 000

Target Form ZINV_01

Original Language Only

Flow Trace

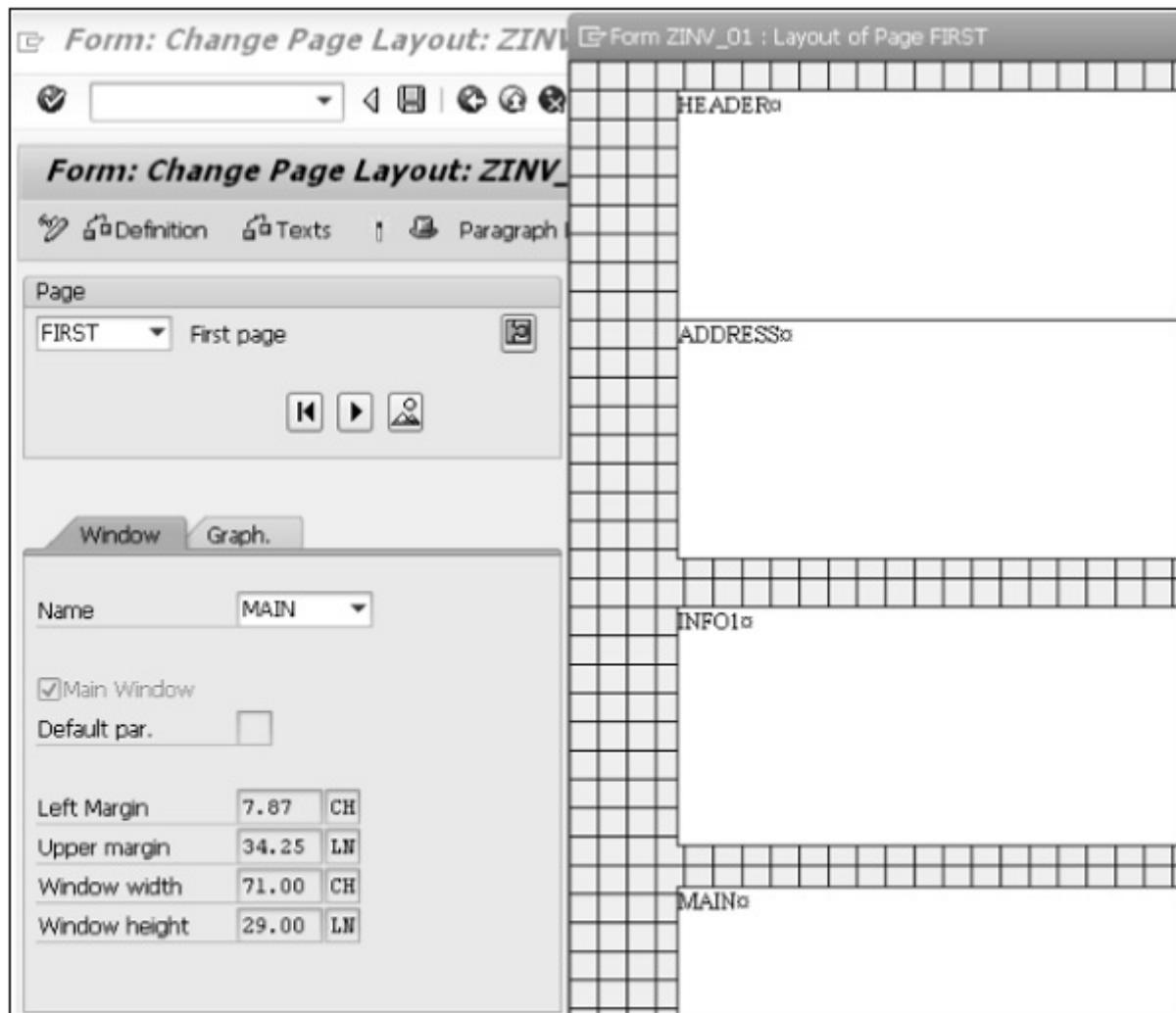
Step 6 – Next, click the Execute icon in the 'Copy Forms Between Clients' screen. The 'Create Object Directory Entry' dialog box appears. Click the Save icon.

The ZINV_01 form is copied from the RVINVOICE01 form and displayed in the 'Copy Forms Between Clients screen' as depicted in the following snapshot –



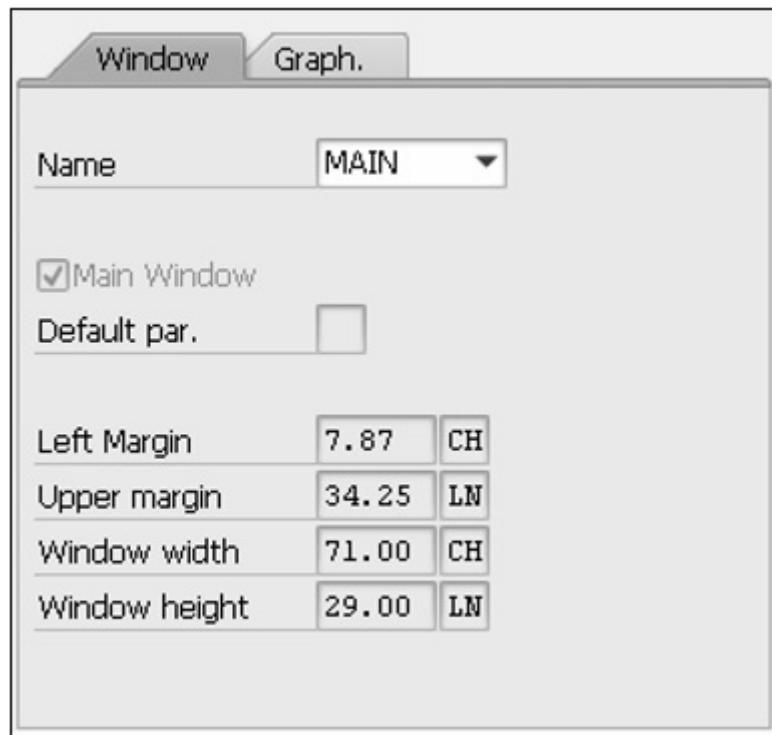
Step 7 – Click the back icon twice and navigate back to the Form Painter: Request screen, which contains the name of the copied form ZINV_01.

Step 8 – After clicking the Display button, the 'Form ZINV_01: Layout of Page FIRST' window and the 'Form: Change Page Layout: ZINV_01' screen appears as shown in the following screenshot.



Step 9 – The 'Form ZINV_01: Layout of Page FIRST' window shows the initial layout of the form. The layout of the form contains five windows: HEADER, ADDRESS, INFO, INFO1, and MAIN. The description of these windows can be accessed in PC Editor.

For instance, by just selecting the MAIN window and clicking the Text icon in the 'Form: Change Page Layout: ZINV_01' screen, you can view all the margin values as shown in the following screenshot –

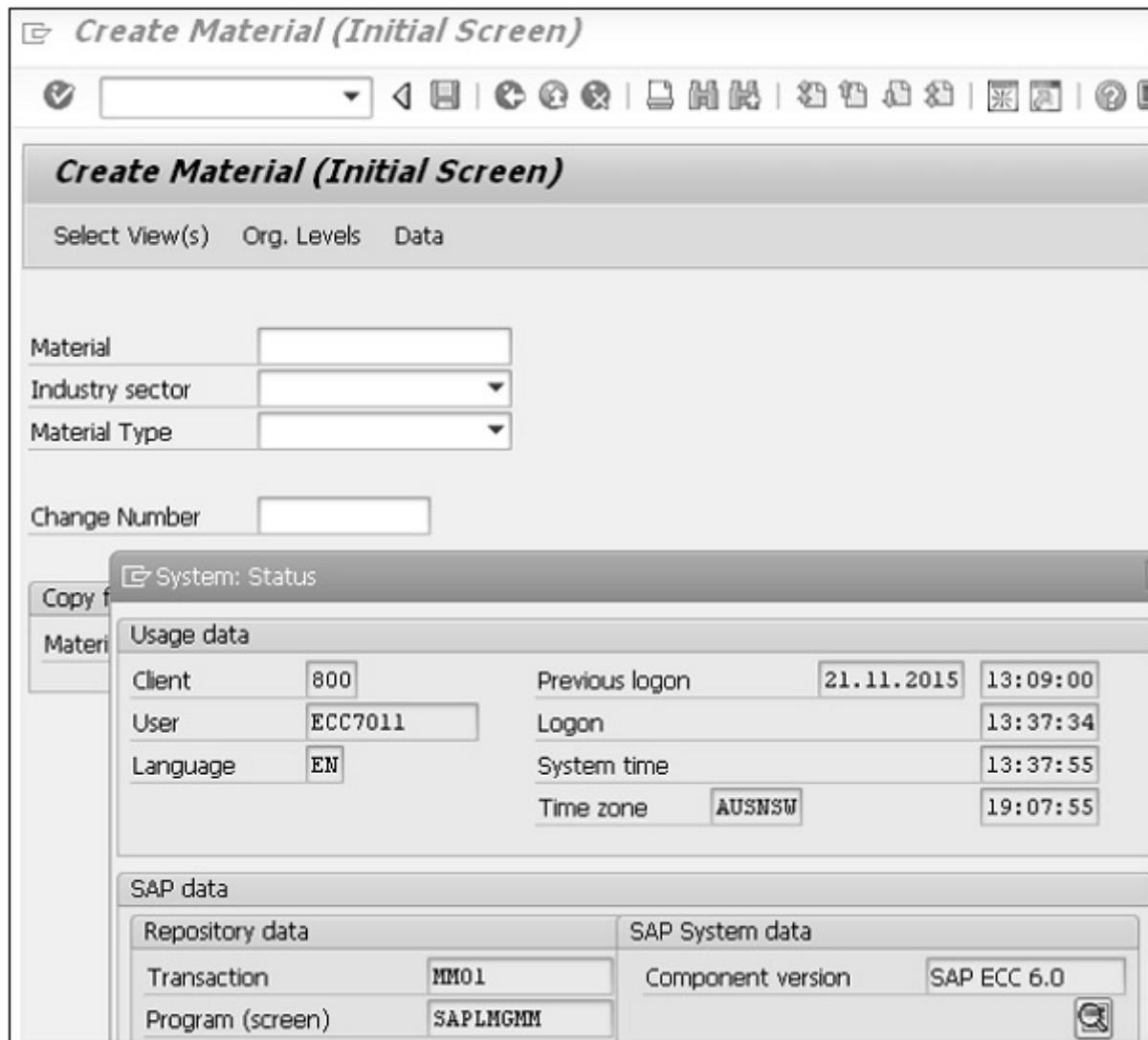


SAP ABAP - Customer Exits

Customer exits could be considered as hooks to the SAP standard programs. We do not need an access key to write the code and there is no need to modify the SAP standard program. These exits don't have any functionality and they are empty. Business logic could be added in order to meet various client requirements. However, Customer Exits are not available for all programs.

Customer Exits for Standard Transactions

Following are the steps to find customer exits as far as standard transactions are concerned. Let's identify customer exits available in MM01 (Material Master Creation).



Step 1 – Go to transaction MM01 and identify the program name of MM01 by going to Menu bar → System → Status as shown in the above screenshot.

Step 2 – Get the program name from the popup screen. The program name is 'SAPLMGMM'.

Step 3 – Go to transaction SE38, enter the program name and click Display.

Step 4 – Navigate to Go to → Properties and find out the package of this program name.

ABAP Editor: Display FunctionPool SAPLMGMM

```

1   * ****
2   | * System-defined Include-files.
3   |
4   | INCLUDE LMGMMTOP.      " Global Data
5   | INCLUDE LMGMMUXX.      " Function Modules
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

```

The package name is 'MGA'.

Step 5 – Go to transaction code SMOD that is usually used to identify customer exits. Navigate to Utilities → Find (or) you may directly press Ctrl + F on the transaction code SMOD.

Step 6 – After going to the 'Find Exits' screen, enter the package name we got earlier and press F8 (Execute) button.

Repository Info System: Find Exits

Exit name	<input type="text"/>	<input type="button" value="→"/>
Short text	<input type="text"/>	<input type="button" value="→"/>
Package	MGA	<input type="button" value="→"/>
Application Component	<input type="text"/>	<input type="button" value="→"/>
Settings		
Maximum No. of Hits	200	

The above steps produce the following output with the list of exits available in the Material Master Creation.

Repository Info System: Exits Find (3 Hits)

Exit name	Short text
<input type="checkbox"/> MGA00001	Material Master (Industry): Checks and Enhancements
<input type="checkbox"/> MGA00002	Material Master (Industry): Number Assignment
<input type="checkbox"/> MGA00003	Material Master (Industry and Retail): Number Display

SAP ABAP - User Exits

User exits are used in an extraction if the standard SAP extractors do not provide the expected data or the required functionality, for instance in authorizations or time checks. User exits are commonly used in Sales and Distribution (SD) modules. There are many exits provided by SAP in the areas of sales, transportation, shipping and billing. A user exit is designed to make some changes when standard SAP is not capable of fulfilling all the requirements.

To be able to access what exits are available in each area of sales, go to IMG using this path: IMG → Sales and Distribution → System Modifications → User Exits. The documentation for each exit in the areas of SD is explained thoroughly.

For instance, if you want to find user exits in Sales Document Processing (contract, quotation or sales order), follow the path mentioned above and continue to expand the node User Exits in

Sales → User Exits. Click on icon documentation to see all user exits available in Sales Document Processing.

S.No.	User Exit & Description
1	USEREXIT_FIELD_MODIFICATION Used to modify screen attributes.
2	USEREXIT_SAVE_DOCUMENT Helps in performing operations when the user hits Save.
3	USEREXIT_SAVE_DOCUMENT_PREPARE Very useful to check input fields, put any value in the field or show a popup to users and to confirm the document.
4	USEREXIT_MOVE_FIELD_TO_VBAK Usado quando as alterações do cabeçalho do usuário são movidas para a área de trabalho do cabeçalho.
5	USEREXIT_MOVE_FIELD_TO_VBAP Usado quando as alterações de itens do usuário são movidas para a área de trabalho de itens SAP.

Uma User Exit tem a mesma finalidade que as Customer Exits, mas estão disponíveis apenas para o módulo SD. A saída é implementada como uma chamada para um Módulo de Função. As saídas de usuário são modificações nos programas padrão SAP.

Exemplo

```

REPORT ZUSEREXIT1.

TABLES:
  TSTC, TSTCT,
  TADIR, TRDIR, TFDIR, ENLFDIR,
  MODSAPT, MODACT.

DATA:
  JTAB LIKE TADIR OCCURS 0 WITH HEADER LINE,
  field1(30),

```

```
v_devclass LIKE TADIR-devclass.
```

PARAMETERS:

```
P_TCODE LIKE TSTC-tcode OBLIGATORY.
```

```
SELECT SINGLE *
FROM TSTC
WHERE tcode EQ P_TCODE.
```

```
IF SY-SUBRC EQ 0.
  SELECT SINGLE *
  FROM TADIR
```

```
WHERE pgmid = 'R3TR' AND
      object = 'PROG' AND
      obj_name = TSTC-pgmna.
```

```
MOVE TADIR-devclass TO v_devclass.
```

```
IF SY-SUBRC NE 0.
  SELECT SINGLE *
  FROM TRDIR
  WHERE name = TSTC-pgmna.
```

```
IF TRDIR-subc EQ 'F'.
  SELECT SINGLE *
  FROM TFDIR
  WHERE pname = TSTC-pgmna.
```

```
SELECT SINGLE *
FROM ENLFDIR
WHERE funcname = TFDIR-funcname.
```

```
SELECT SINGLE *
FROM TADIR
WHERE pgmid = 'R3TR' AND
      object = 'FUGR' AND
      obj_name EQ ENLFDIR-area.
MOVE TADIR-devclass TO v_devclass.
```

```
ENDIF.
```

```
ENDIF.
```

```
SELECT *
FROM TADIR
INTO TABLE JTAB
```

```

WHERE pgmid = 'R3TR' AND
      object = 'SMOD' AND
      devclass = v_devclass.

SELECT SINGLE *
  FROM TSTCT
 WHERE sprsl EQ SY-LANGU AND
       tcode EQ P_TCODE.

FORMAT COLOR COL_POSITIVE INTENSIFIED OFF.
WRITE:(19) 'Transaction Code - ',
           20(20) P_TCODE,
           45(50) TSTCT-ttext.

SKIP.

IF NOT JTAB[] IS INITIAL.
  WRITE:(95) SY-ULINE.
  FORMAT COLOR COL_HEADING INTENSIFIED ON.

  WRITE:/1 SY-VLINE,
        2 'Exit Name',
        21 SY-VLINE ,
        22 'Description',
        95 SY-VLINE.

  WRITE:(95) SY-ULINE.
  LOOP AT JTAB.
    SELECT SINGLE * FROM MODSAPT
    WHERE sprsl = SY-LANGU AND
          name = JTAB-obj_name.

    FORMAT COLOR COL_NORMAL INTENSIFIED OFF.
    WRITE:/1 SY-VLINE,
          2 JTAB-obj_name HOTSPOT ON,
          21 SY-VLINE ,
          22 MODSAPT-modtext,
          95 SY-VLINE.

  ENDLOOP.

  WRITE:(95) SY-ULINE.
  DESCRIBE TABLE JTAB.
  SKIP.
  FORMAT COLOR COL_TOTAL INTENSIFIED ON.
  WRITE:/ 'No of Exits:' , SY-TFILL.

ELSE.

```

```

FORMAT COLOR COL_NEGATIVE INTENSIFIED ON.
WRITE:/95 'User Exit doesn't exist'.
ENDIF.

ELSE.

FORMAT COLOR COL_NEGATIVE INTENSIFIED ON.
WRITE:/95 'Transaction Code Does Not Exist'.
ENDIF.

```

```

AT LINE-SELECTION.
GET CURSOR FIELD field1.
CHECK field1(4) EQ 'JTAB'.
SET PARAMETER ID 'MON' FIELD sy-lisel+1(10).
CALL TRANSACTION 'SMOD' AND SKIP FIRST SCREEN.

```

Durante o processamento, insira o código de transação 'ME01' e pressione o botão F8 (Executar). O código acima produz a seguinte saída -

Transaction Code - ME01		Maintain Source List
Exit Name	Description	
AMPL0001	User subscreen for additional data on AMPL	
LMEDR001	Enhancements to print program	
LMELA002	Adopt batch no. from shipping notification when posting a GR	
LMELA010	Inbound shipping notification: Transfer item data from IDOC	
LMEQR001	User exit for source determination	
LMEXF001	Conditions in Purchasing Documents Without Invoice Receipt	
LWSUS001	Customer-Specific Source Determination in Retail	
M06B0001	Role determination for purchase requisition release	
M06B0002	Changes to comm. structure for purchase requisition release	
M06B0003	Number range and document number	
M06B0004	Number range and document number	
M06B0005	Changes to comm. structure for overall release of requis. doc.	
M06E0004	Changes to communication structure for release purch. doc.	
M06E0005	Role determination for release of purchasing documents	
ME590001	Grouping of requisitions for PO split in ME59	
MEETA001	Define schedule line type (backlog, immed. req., preview)	
MEFLD004	Determine earliest delivery date f. check w. GR (only PO)	
MELAB001	Gen. forecast delivery schedules: Transfer schedule implem.	
MEQUERY1	Enhancement to Document Overview ME21N/ME51N	
MEVME001	WE default quantity calc. and over/underdelivery tolerance	
MM06E001	User exits for EDI inbound and outbound purchasing documents	
MM06E003	Number range and document number	
MM06E004	Control import data screens in purchase order	
MM06E005	Customer fields in purchasing document	

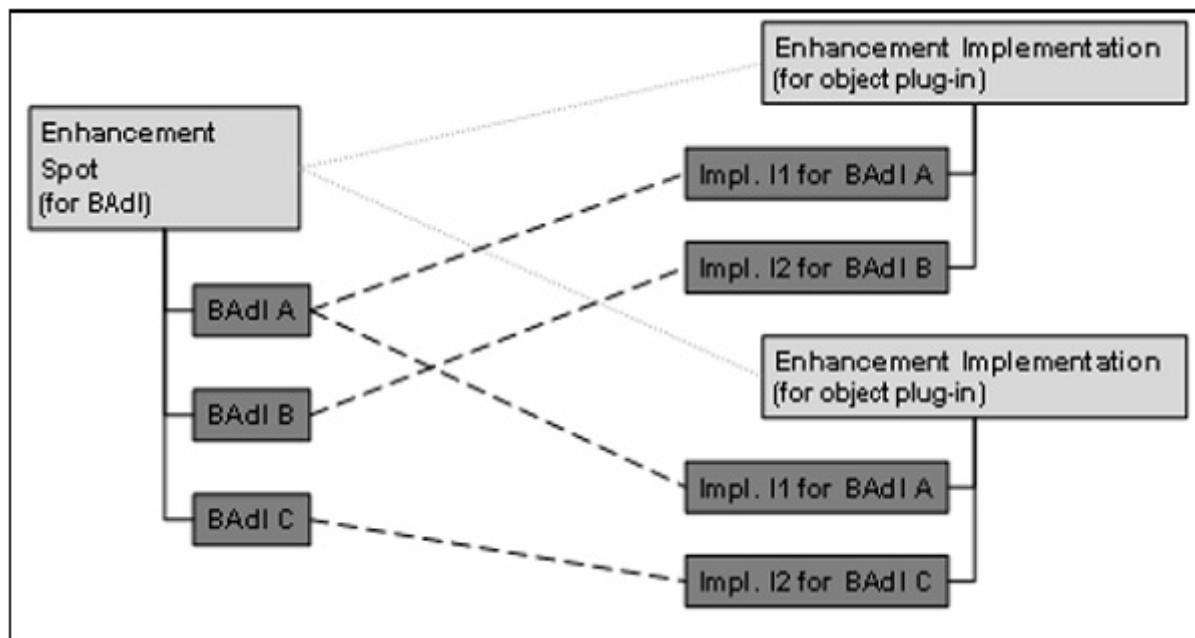
SAP ABAP - Suplementos de negócios

Em alguns casos, funções especiais precisam ser predefinidas em um aplicativo de software para aprimorar a funcionalidade de vários aplicativos. Existem muitos suplementos do Microsoft Excel para melhorar a funcionalidade do MS Excel. Da mesma forma, a SAP facilita algumas funções predefinidas, fornecendo **Business Add-Ins** conhecidos como BADI.

Um BADI é uma técnica de aprimoramento que facilita a um programador SAP, um usuário ou um setor específico adicionar algum código adicional ao programa existente no sistema SAP. Podemos usar lógica padrão ou customizada para melhorar o sistema SAP. Uma BADI deve primeiro ser definida e depois implementada para aprimorar a aplicação SAP. Ao definir uma BADI, uma interface é criada. BADI é implementado por esta interface, que por sua vez é implementada por uma ou mais classes de adaptadores.

A técnica BADI é diferente de outras técnicas de aprimoramento de duas maneiras -

- A técnica de aprimoramento pode ser implementada apenas uma vez.
- Esta técnica de aprimoramento pode ser usada por muitos clientes simultaneamente.



Você também pode criar BADIs de filtro, o que significa que as BADIs são definidas com base em dados filtrados, o que não é possível com técnicas de aprimoramento. O conceito de BADIs foi redefinido no SAP Release 7.0 com os seguintes objetivos -

- Aprimorar as aplicações padrão em um sistema SAP adicionando dois novos elementos na linguagem ABAP, que são 'GET BADI' e 'CALL BADI'.
- Oferecendo mais recursos de flexibilidade como contextos e filtros para aprimoramento de aplicações padrão em um sistema SAP.

Quando um BADI é criado, ele contém uma interface e outros componentes adicionais, como códigos de função para aprimoramentos de menu e de tela. Uma criação BADI permite que os clientes incluam suas próprias melhorias na aplicação SAP padrão. O aprimoramento, a interface e as classes geradas estão localizados em um namespace de desenvolvimento de aplicativo apropriado.

Conseqüentemente, um BADI pode ser considerado como uma técnica de aprimoramento que utiliza objetos ABAP para criar 'pontos predefinidos' nos componentes SAP. Esses pontos predefinidos são então implementados por soluções industriais individuais, variantes de país.

parceiros e clientes para atender às suas necessidades específicas. Na verdade, a SAP introduziu a técnica de aprimoramento BADI com a versão 4.6A, e a técnica foi reimplementada novamente na versão 7.0.

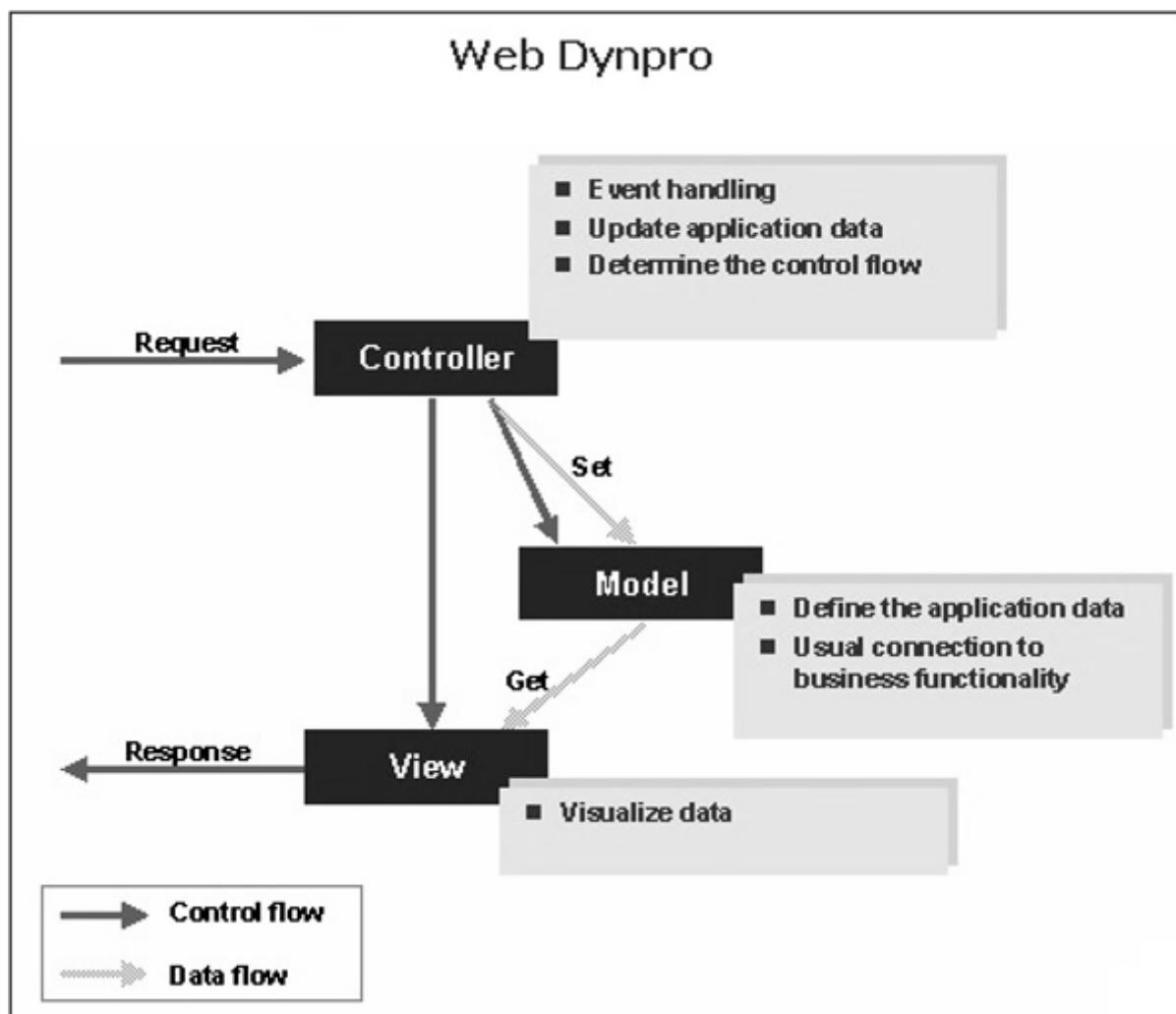
SAP ABAP - Web Dynpro

Web Dynpro (WD) para ABAP é a tecnologia de interface de usuário padrão SAP desenvolvida pela SAP AG. Pode ser usado no desenvolvimento de aplicativos baseados na web no ambiente SAP ABAP que utiliza ferramentas e conceitos de desenvolvimento SAP. Ele fornece uma interface de usuário web front-end para conectar-se diretamente aos sistemas back-end SAP R/3 para acessar dados e funções para geração de relatórios.

Web Dynpro for ABAP consiste em um ambiente de execução e um ambiente de desenvolvimento gráfico com ferramentas de desenvolvimento específicas que estão integradas no ABAP Workbench (transação: SE80).

Arquitetura da Web Dynpro

A ilustração a seguir mostra a arquitetura geral do Web Dynpro -



A seguir estão alguns pontos que você deve ter em mente em relação ao Web Dynpro -

- Web Dynpro é o modelo de programação SAP NetWeaver para interfaces de usuário.
- Todas as aplicações Web Dynpro são estruturadas de acordo com o modelo de programação Model View Controller (MVC).
- O modelo define uma interface para o sistema principal e a aplicação Web Dynpro pode ter acesso aos dados do sistema.
- A visualização é responsável por mostrar os dados no navegador web.
- O controlador reside entre a visualização e o modelo. O controlador formata os dados do modelo para serem exibidos na vista. Ele processa as entradas feitas pelo usuário e as retorna ao modelo.

Vantagens

Web Dynpro oferece as seguintes vantagens para desenvolvedores de aplicativos -

- A utilização de ferramentas gráficas reduz significativamente o esforço de implementação.
- Reutilização e melhor manutenção usando componentes.
- O layout e a navegação são facilmente alterados usando as ferramentas Web Dynpro.
- A acessibilidade da interface do usuário é suportada.
- Integração total no ambiente de desenvolvimento ABAP.

Componente e janela Web Dynpro

O componente é a unidade global do projeto do aplicativo Web Dynpro. A criação de um componente Web Dynpro é a etapa inicial no desenvolvimento de um novo aplicativo Web Dynpro. Depois que o componente é criado, ele atua como um nó na lista de objetos Web Dynpro. Você pode criar qualquer número de visualizações de componentes em um componente e montá-las em qualquer número de janelas Web Dynpro correspondentes.

Pelo menos uma janela do Web Dynpro está contida em cada componente do Web Dynpro. A janela Web Dynpro incorpora todas as visualizações exibidas no aplicativo web front-end. A janela é processada nos editores de janelas do ABAP Workbench.

Observação

- A visualização do componente exibe todos os detalhes administrativos do aplicativo, incluindo a descrição, o nome da pessoa que o criou, a data de criação e o pacote de desenvolvimento atribuído.
- A aplicação Web Dynpro é o objeto independente na lista de objetos do ABAP Workbench. A interação entre a janela e a aplicação é criada pela visualização da interface de uma determinada janela.

