

## 0617 | ABAP

DATA OBJECT | DATA 값을 참고하여 값을 저장할 수 있는 변수

FIELD

STRUCTURE

TABLE 


\*FIELD + STRUCTURE + TABLE 의 조합으로 이루어진 OBJECT도 있다.

DATA TYPE | 프로그램에서 사용할 수 있는 DATA TYPE을 정의

LOCAL	프로그램 내에서 정의한 DATA TYPE*
GLOBAL	모든 ABAP 프로그램 내에서 사용할 수 있는 DATA TYPE**
STANDARD	PREDEFINED ABAP TYPE   기본 데이터 타입

\*프로그램 내에서 PREDEFINED ABAP TYPE을 이용해 LOCAL TYPE 생성.

\*\*ABAP DICTIONARY DATA TYPE 은 프로그램 내에서 TYPE 구문 사용 가능.

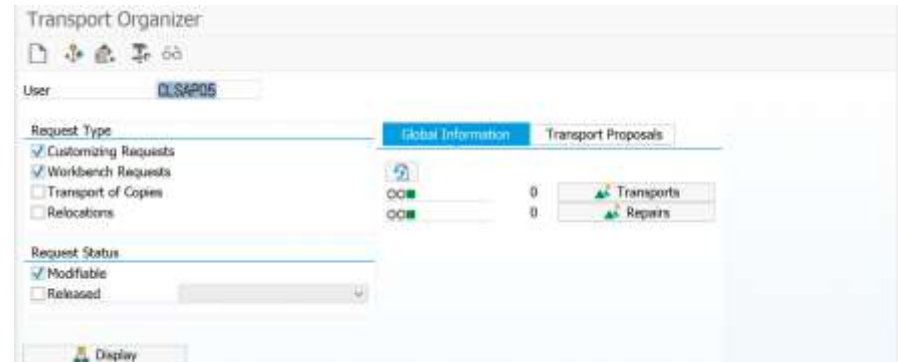
STANDARD |

COMPLETE	길이가 정해져 있음	D T   STRING
INCOMPLETE	길이가 정해져 있지 않음	C N P

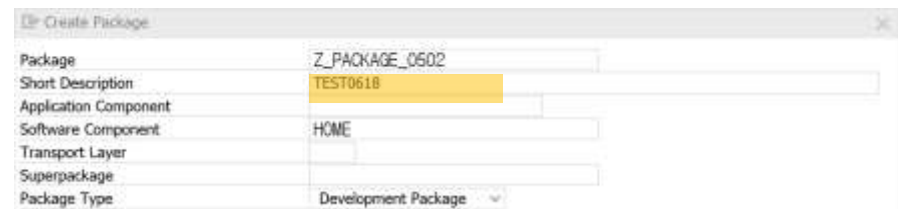
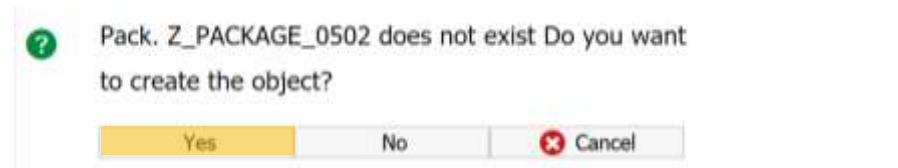
INTERNAL TABLE | ABAP 핵심 기술로 단일 프로그램 내에서 정의하여 사용할 수 있는 LOCAL TABLE로 프로그램 개발 및 유지보수를 좀 더 쉽고 편리하게 할 수 있다.

## SE09 | CTS | CHANGE AND TRANSACTION SYSTEM

DEVELOPMENT SYSTEM을 PRODUCTION SYSTEM으로 이관하는 것을 의미.



## SE80 | PACKAGE | PACKAGE 를 생성하자



Prompt for local Workbench request

Package: Z\_PACKAGE\_0502

Request: DDSK901347 Workbench request

Short Description: CLSAP05 TEST

Own Requests

\*만약 REQUEST 가 존재하지 않는 경우 새롭게 생성해서 진행한다.

SE80 | 0501 | 어제 배운 내용을 복습해보자

*\*FIELD*

```
DATA : GV_VAL1 TYPE C,
        GV_VAL2 TYPE C,
        GV_VAL3 TYPE C,
        GV_VAL4 TYPE C,
        GV_VAL5 TYPE C.
```

*\*LOCAL DATA TYPE*

```
TYPES T_TYPE1 TYPE C LENGTH 8.
TYPES T_TYPE2 TYPE N LENGTH 5.
TYPES T_TYPE3 TYPE P LENGTH 3 DECIMALS 2.
TYPES T_TYPE4 TYPE N LENGTH 5.
TYPES T_TYPE5 TYPE N LENGTH 5.
```

*\*TYPE 형식을 가져와 FIELD 선언하기*

```
TYPES T_TYPE6 TYPE P LENGTH 3 DECIMALS 2.
```

```
DATA : GV_PERCENTAGE TYPE T_TYPE6.
```

*\*STR*

```
DATA: BEGIN OF GV_STR,
        GV_VAL1 TYPE C,
        GV_VAL2 TYPE C,
        GV_VAL3 TYPE C,
        GV_VAL4 TYPE C,
        GV_VAL5 TYPE C,
      END OF GV_STR.
```

*\*TYPE 형태로 STR*

```
TYPES: BEGIN OF TY_STR,
        GV_VAL6 TYPE C,
        GV_VAL7 TYPE C,
        GV_VAL8 TYPE C,
        GV_VAL9 TYPE C,
        GV_VAL10 TYPE C,
      END OF TY_STR.
```

*\*TABLE*

```
DATA: GV_TABLE LIKE TABLE OF GV_STR.
```

```
DATA: GV_TABLE2 LIKE GV_TABLE.
```

SE80 | 0502 | 어제 배운 내용을 복습해보자

*\*GLOBAL STRUCTURE 내의 FIELD를 참조하여 FIELD를 선언하자.*

*\*더블클릭으로 TYPE을 확인할 수 있다.*

```
DATA: GV_MAX TYPE ZSBC400FOCC-SEATSMAX,  
      GV_OCC TYPE ZSBC400FOCC-SEATSOCC,  
      GV_PERCENTAGE TYPE ZSBC400FOCC-PERCENTAGE,  
      GV_LENGTH TYPE I,  
      GV_STRING TYPE STRING.
```

*\*변수의 값을 지정해주자*

```
GV_MAX = 50.  
GV_OCC = 10.  
GV_STRING = 'QWERTY'.
```

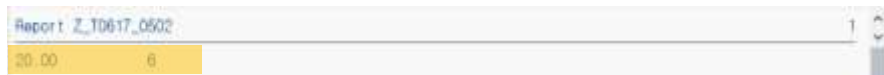
```
GV_PERCENTAGE = GV_OCC * 100 / GV_MAX.
```

*\*GV\_OCC인 10\*100 / GV\_MAX인 50 을 계산하면 20.00 이 된다.*

```
GV_LENGTH = STRLEN( GV_STRING ).
```

*\*지정한 값을 통해 GV\_STRING의 길이를 STRLEN으로 구한다.*

```
WRITE: GV_PERCENTAGE, GV_LENGTH.  
BREAK-POINT.
```



SE80 | 0503 | 0502 파일을 PARAMETER로 값을 지정해 연산을 수행하자.

*\*GLOBAL STRUCTURE 내의 FIELD를 참조하여 FIELD를 선언하자.*

*\*더블클릭으로 TYPE을 확인할 수 있다.*

```
DATA: GV_MAX TYPE ZSBC400FOCC-SEATSMAX,  
      GV_OCC TYPE ZSBC400FOCC-SEATSOCC,  
      GV_PERCENTAGE TYPE ZSBC400FOCC-PERCENTAGE,  
      GV_LENGTH TYPE I,  
      GV_STRING TYPE STRING.
```

*\*PARAMETERS로 값을 받아오자.*

```
PARAMETERS : PA_MAX TYPE ZSBC400FOCC-SEATSMAX,  
             PA_OCC TYPE ZSBC400FOCC-SEATSOCC,  
             PA_STR TYPE STRING.
```

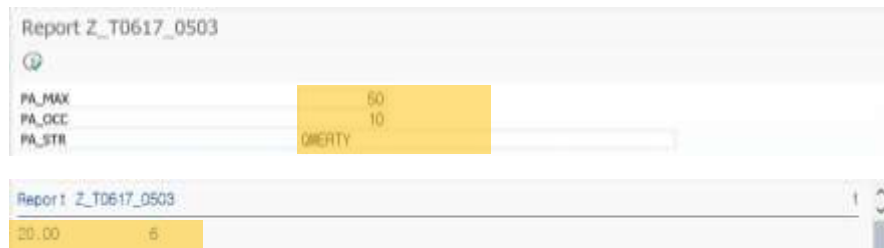
*\*PARAMETERS로 받아 온 값을 매칭시켜주자*

```
GV_MAX = PA_MAX.  
GV_OCC = PA_OCC .  
GV_STRING = PA_STR.
```

*\*연산을 수행하자*

```
GV_PERCENTAGE = GV_OCC * 100 / GV_MAX.  
GV_LENGTH = STRLEN( GV_STRING ).
```

```
WRITE: GV_PERCENTAGE, GV_LENGTH.
```



## INTERNAL TABLE | DYNAMIC DATA OBJECT

ABAP 핵심 기술로 단일 프로그램 내에서 정의하여 사용할 수 있는 LOCAL TABLE로 프로그램 개발 및 유지보수를 좀 더 쉽고 편리하게 할 수 있다. INTERNAL TABLE은 DYNAMIC DATA OBJECT 동적인 구조체 배열로, TYPE 구문을 기반으로 INITIAL SIZE 구문을 사용하여 테이블 크기만을 선언하고 MEMORY에 LOAD 하지 않음을 의미한다. 즉, TABLE의 형태인 크기만을 선언 하기 때문에 INSERT 또는 APPEND 를 사용하여 LINE이 추가될 때마다 MEMORY에 LOAD 한다. 이는 INITIAL SIZE 구문으로 실제로 MEMORY 공간을 할당하는 것이 아닌 RESERVE 예약을 하는 것임을 의미한다.

SE80 | 0504 | PARAMETER로 값을 지정해보자

PARAMETERS: PA\_NAME TYPE STRING.

DATA: GV\_STR TYPE STRING.

WRITE 'HELLO WORLD'.

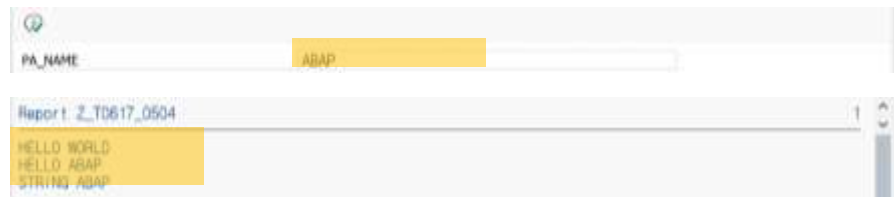
NEW-LINE.

WRITE: 'HELLO', PA\_NAME.

NEW-LINE.

GV\_STR = PA\_NAME.

WRITE: 'STRING', GV\_STR.

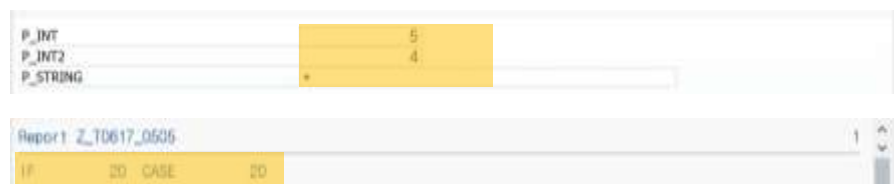


PARAMETERS | 사용자가 값을 입력하도록 INPUT FIELD 를 정의한다. PARAMETERS는 1개의 값만 입력 받을 수 있다.

## NUMERIC OPERATION | 연산자

+	더하기	ADD_TO_
-	빼기	SUBTRACT_FROM_
*	곱하기	MULTYPLT_BY_
/	나누기	DIVIDE_BY_
DIV	INTEGER 나누기의 몫	
MOD	INTEGER 나누기의 나머지	
**	제곱	

SE80 | 0505 | 연산자를 활용해보자



PARAMETERS: P\_INT TYPE I,  
P\_INT2 TYPE I,  
P\_STRING TYPE STRING.

```
DATA RESULT TYPE I.
```

```
IF P_STRING = '+'.
```

```
    RESULT = P_INT + P_INT2.
```

```
    WRITE : 'IF',RESULT.
```

```
ELSEIF P_STRING = '-'.
```

```
    RESULT = P_INT - P_INT2.
```

```
    WRITE : 'IF',RESULT.
```

```
ELSEIF P_STRING = '*'.
```

```
    RESULT = P_INT * P_INT2.
```

```
    WRITE : 'IF',RESULT.
```

```
ELSEIF P_STRING = '/'.
```

```
    RESULT = P_INT / P_INT2.
```

```
    WRITE : 'IF',RESULT.
```

```
ELSEIF P_STRING = '**'.
```

```
    RESULT = P_INT ** P_INT2.
```

```
    WRITE : 'IF',RESULT.
```

```
ELSEIF P_STRING = 'DIV'.
```

```
    RESULT = P_INT DIV P_INT2.
```

```
    WRITE : 'IF',RESULT.
```

```
ELSEIF P_STRING = 'MOD'.
```

```
    RESULT = P_INT MOD P_INT2.
```

```
    WRITE : 'IF',RESULT.
```

```
ENDIF.
```

```
*CASE
```

```
CASE P_STRING.
```

```
    WHEN '+'.
```

```
        RESULT = P_INT + P_INT2.
```

```
        WRITE : 'CASE',RESULT.
```

```
    WHEN '-'.
```

```
        RESULT = P_INT - P_INT2.
```

```
        WRITE : 'CASE',RESULT.
```

```
    WHEN '*'.
```

```
        RESULT = P_INT * P_INT2.
```

```
        WRITE : 'CASE',RESULT.
```

```
    WHEN '/'.
```

```
        RESULT = P_INT / P_INT2.
```

```
        WRITE : 'CASE',RESULT.
```

```
    WHEN '**'.
```

```
        RESULT = P_INT ** P_INT2.
```

```
        WRITE : 'CASE',RESULT.
```

```
    WHEN 'DIV'.
```

```
        RESULT = P_INT DIV P_INT2.
```

```
        WRITE : 'CASE',RESULT.
```

```
    WHEN 'MOD'.
```

```
        RESULT = P_INT MOD P_INT2.
```

```
        WRITE : 'CASE',RESULT.
```

```
ENDCASE.
```

SE80 | 0506 | 비교 연산자를 활용해 보자

```
DATA GV_STR TYPE STRING.  
GV_STR = 'ABC'.
```

*\*&& 문자열을 한 문자열로 만들 수 있다.*

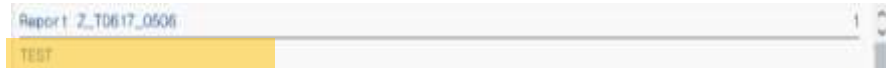
*\*\*문자열 비교 연산자에는 CA가 있다.*

*\*\*CA는 포함한다면 을 의미한다.*

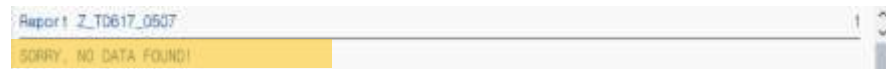
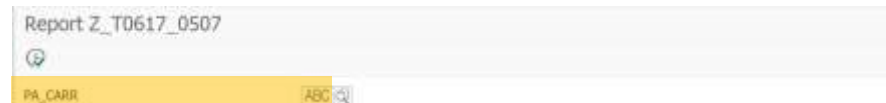
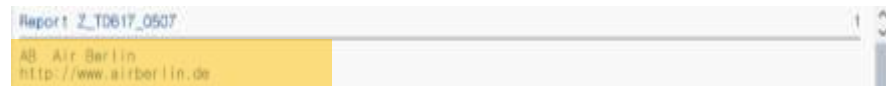
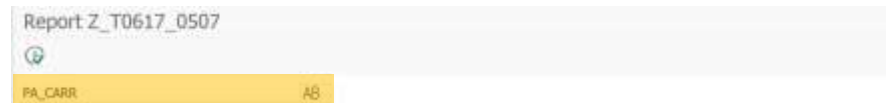
```
IF 'LIT1' && GV_STR CA 'ABC'.
```

*\*즉 LIT1과 GV\_STR을 한 문자열로 만든 LIT1ABC에 ABC가 포함 된다면 을 의미한다.*

```
WRITE: /'TEST'.  
ENDIF.
```



SE80 | 0507 | SY-SUBRC를 통해 로직 수행 여부를 확인하자



SE80 | 0507 | SY-SUBRC를 통해 로직 수행 여부를 확인하자

```
PARAMETERS PA_CARR TYPE SCARR-CARRID.  
DATA GS_SCARR TYPE SCARR.
```

```
SELECT SINGLE * FROM SCARR  
                INTO GS_SCARR  
                WHERE CARRID = PA_CARR.
```

*\*SY-SUBRC 는 정상적으로 로직이 수행되는지 확인하며 TRUE인 경우 0을 반환한다.*

```
IF SY-SUBRC = 0.
```

```
NEW-LINE.
```

```
WRITE : GS_SCARR-CARRID,  
        GS_SCARR-CARRNAME,  
        GS_SCARR-URL.
```

```
ELSE.
```

```
WRITE 'SORRY, NO DATA FOUND!'.
```

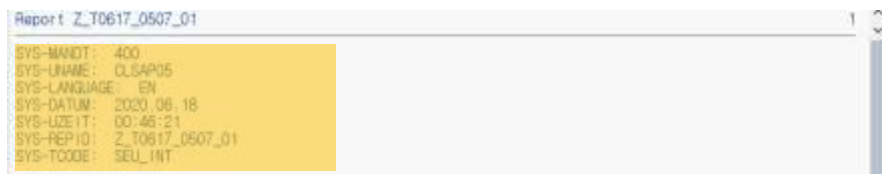
```
ENDIF.
```

SE80 | 0507\_01 | SYSTEM FIELD에 대해 알아보자.

SYS-MANDT	시스템 클라이언트 번호
SYS-UNAME	시스템 로그인 아이디
SY-LANGU	시스템 언어
SYS-DATUM	시스템 현재   LOCAL 날짜
SY-UZEIT	시스템 현재   LOCAL 시간
SY-REPID	시스템 ABAP PROGRAM 이름
SY-CODE	시스템 DO 또는 WHILE LOOP 숫자

\*SYSTEM FIELD

WRITE: / 'SYS-MANDT: ', SY-MANDT, / 'SYS-UNAME: ', SY-UNAME, / 'SYS-LANGUAGE: ', SY-LANGU,  
/ 'SYS-DATUM: ', SY-DATUM, / 'SYS-UZEIT: ', SY-UZEIT, / 'SYS-REPID: ', SY-REPID, / 'SYS-TCODE: ', SY-TCODE.

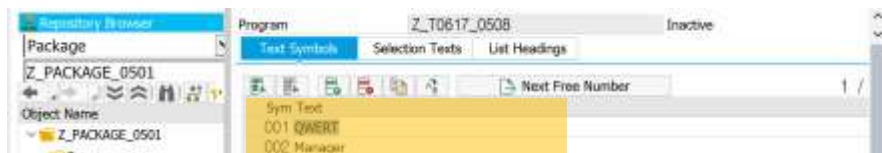


Report Z_T0617_0507_01
SYS-MANDT: 400
SYS-UNAME: CLSAP05
SYS-LANGUAGE: EN
SYS-DATUM: 2020.06.18
SYS-UZEIT: 00:46:21
SYS-REPID: Z_T0617_0507_01
SYS-TCODE: SEU_INT

SE80 | 0508 | SYSTEM FIELD에 대해 알아보자.

\*TEX-001을 더블 클릭 후 TEXT를 입력한다.

\*MANAGER을 더블 클릭하면 자동으로 값이 입력된다.



Repository Browser	Program Z_T0617_0508
Package Z_PACKAGE_0501	Text Symbols
Object Name Z_PACKAGE_0501	Selection Tests
	List Headings
	Next Free Number
	Sym Text
	001 QWERT
	002 Manager

\*TEXT SYMBOL 은 똑같은 구문을 반복적으로 사용할 때

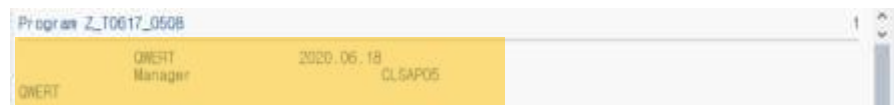
WRITE: /15 TEXT-001, "텍스트 입력해서 문구사용"

35 SY-DATUM, "날짜"

/15 'Manager'(002),

45 SY-UNAME. "사용자 이름"

WRITE:/ TEXT-001.



Program Z_T0617_0508
QWERT Manager
2020.06.18
CLSAP05

DEBUGGING | 오류를 찾아 수정하는 일

모든 화면	COMMAND 창에서 /H 입력 후 다음단계 진행
모든 화면	SYSTEM   UTILITIES   DEBUG ABAP
SE80	PROGRAM   TEST   DEBUGGING*
SE38	PROGRAM   TEST   DEBUGGING**

\*프로그램 또는 트랜잭션을 열어 메뉴를 활용한다.

\*\*프로그램 이름 입력 후 디버깅 버튼 클릭 | SOURCE CODE 조회 후 사용

DEBUGGING | 다음 순서로 이동 단축 키

F5	한 단계씩 다음 단계로 이동
F6	한 단계씩 다음 단계로 이동*
F7	현재 실행중인 SUBROUTINE을 빠져나오고 다음 단계 이동
F8	프로그램을 실행, 다음 중단점 또는 관찰점까지 실행

\*SUBROUTINE을 만나면 실행 후 다음 단계로 이동한다. 즉, PERFORM 구문에  
서 F6를 입력하면 FORM구문의 DEBUGGING을 건너뛰고, 다음라인으로 이동  
하며 F5를 입력하면 FORM 구문으로 이동한다.

SE80 | 0509 | DEBUGGING에 대해 알아보자.

*\*\*만약 DEBUGGING을 하고 싶을 때 실행 후 PARAMETER의 값을 넣은 다음 T-CODE창에 /H를 입력하면 된다.*

*\*\*F5를 누르면 한 줄 한 줄 DEBUGGING이 실행된다.*

*\*\*\*DEBUGGING MODE에서 값을 변경하여 값에 따른 결과값을 확인할 수 있다.*

*\*\*WATCH POINT 반복문의 경우 코드 내에서 중단하여 결과값을 확인할 수 있다.*

*\*\*\*\*ZBC400\_DED\_DEBUG\_EXERCISE PROGRAM을 검색해 확인해보자.*

PARAMETERS : P\_INT TYPE I,  
P\_INT2 TYPE I.

DATA RESULT TYPE I.

RESULT = P\_INT - P\_INT2.

*\*/F*

IF RESULT = 0.

WRITE 'RESULT = 0 '.

ELSEIF RESULT = 1.

WRITE 'RESULT = 1 '.

ELSE.

WRITE 'RESULT != 0 '.

ENDIF.

*\*CASE*

CASE RESULT.

WHEN 0.

WRITE 'RESULT = 0 '.

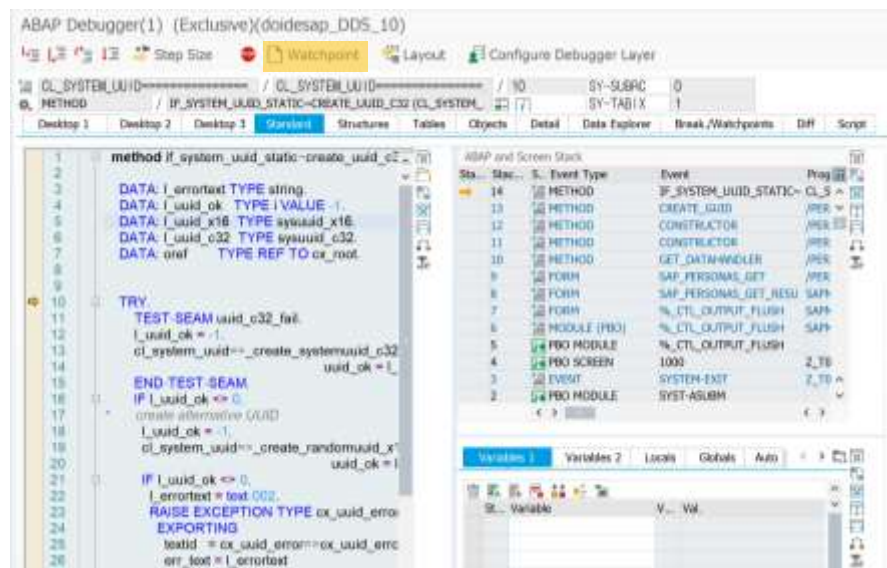
WHEN 1.

WRITE 'RESULT = 1 '.

WHEN OTHERS.

WRITE 'RESULT != 0 '.

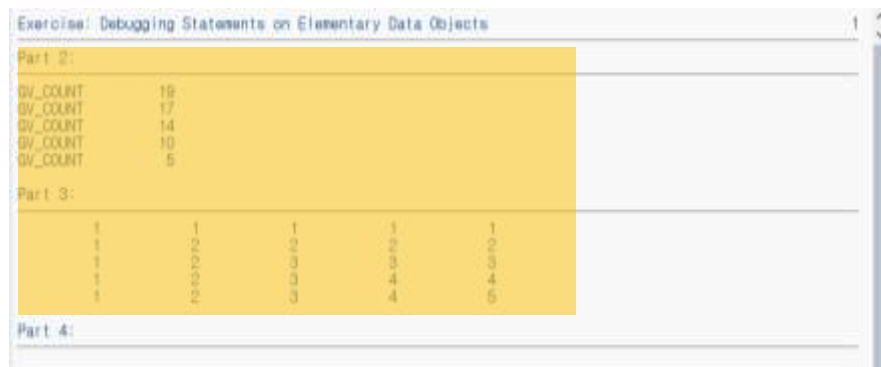
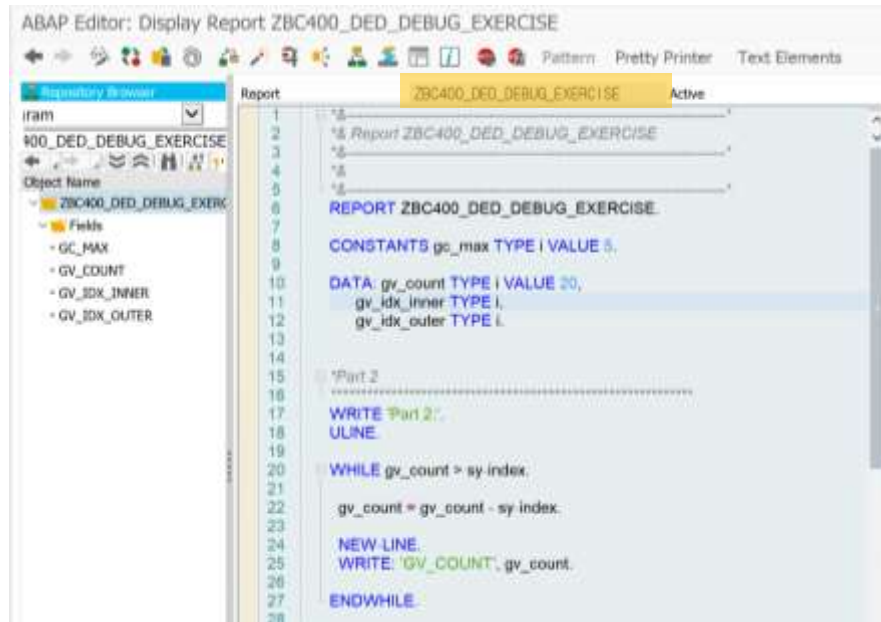
ENDCASE.



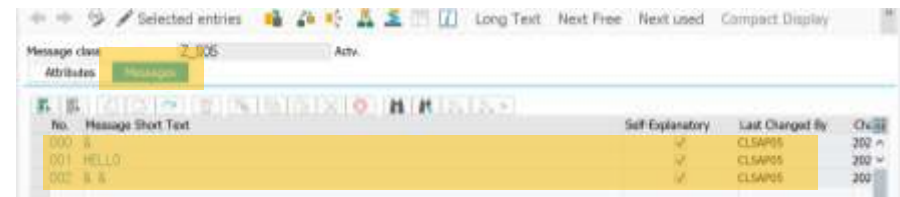
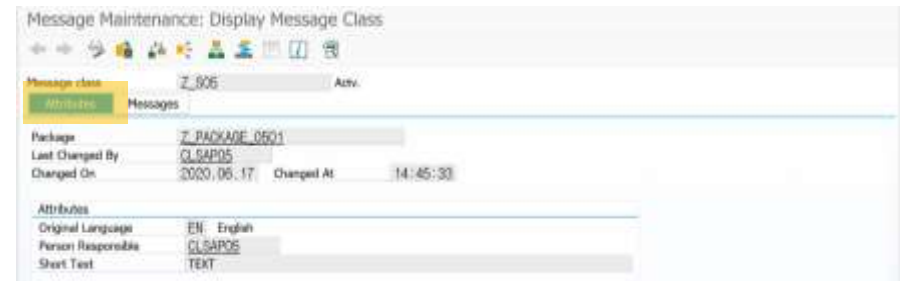
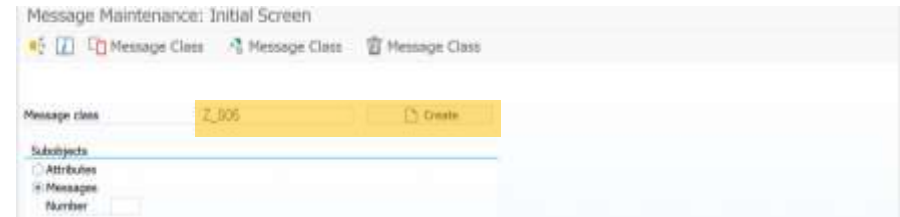
*\*WATCHPOINTS* | 관찰점은 필드 값이 변경되는 시점에서 프로그램이 정지되는 기능을 제공한다. 관찰점의 리스트를 조회할 수 있다.



## \*ZBC400\_DED\_DEBUG\_EXERCISE PROGRAM



## SE91 | MESSAGE 조회 및 입력 | 다양한 MESSAGE를 생성해보자



\*내가 생성한 PACKAGE와 연결될 수 있도록 PACKAGE 입력을 꼭 하자!

\*\*& DMS MESSAGE WITH 에서 사용하는 PARAMETERS로 002와 같이 & & 인 경우 MESSAGE S002 WITH '1' '2' 로 사용할 수 있다.

\*\*\*ABAP 프로그램 화면 하단에 MESSAGE를 출력하려면 리포트 선언 첫 문장에 MESSAGE-ID 를 기술한다. 형태는 다음과 같다.

MESSAGE S01 WITH <FIELD1> "" <FIELD4>

\*\*\*\*이때 S와 같은 MESSAGE TYPE은 다음과 같은 종류가 있다.

## MESSAGE | 메시지 타입

E	메시지 바   입력 값에 대한 체크 시 발생하는 에러 메시지
W	메시지 바   ENTER를 누르면 다음 프로세스를 진행한다. 화면의 로직을 멈추고 경고 메시지 형태로 출력한다.
I	팝업 윈도우   ENTER를 누르면 다음 프로세스를 수행한다.
S	성공 메시지
A	팝업 윈도우   <STOP> BUTTON 이 윈도우 창 안에 있다. <STOP> 을 누르면 프로그램 SESSION 이 종료된다.
X	SHORT DUMP라고 하며, DUMP화면과 함께 프로그램 종료

## SE80 | 0510 | MESSAGE를 확인해보자.

\*SE91 에서 지정한 DIALOG MESSAGE를 확인해보자.

\*회사에서 자재 생성시 개발자마다 다양하게 TEXT를 CUSTOM할 수 있어 통일성이 떨어질 수 있다.

\*이때 MESSAGE CLASS 를 사용해 통일성을 준다.

MESSAGE I000(Z\_S05) WITH 'TEST'.

\*실행하면 DIALOG에서 TEST가 출력된다.

MESSAGE I002(Z\_S05) WITH 'TEST' 'HELLO'.

\*실행하면 DIALOG 에서 TEST HELLO 가 출력된다.

MESSAGE S002(Z\_S05) WITH 'SUCCESS'.

\*실행하면 상태표시줄에 SUCCESS가 출력된다.

MESSAGE E002(Z\_S05) WITH 'ERROR'.

\*실행하면 상태표시줄에 ERROR가 출력된다.

\*ENTER 입력 시 프로세스를 종료한다.

MESSAGE W002(Z\_S05) WITH 'WARNING'.

\*실행하면 상태표시줄에 WARNING가 출력된다.

\*ENTER 입력 시 경고를 하며 프로세스를 종료하지 않는다.

MESSAGE A002(Z\_S05) WITH 'TERMINATION'.

\*실행하면 DIALOG 에서 TEST HELLO 가 출력된다.

MESSAGE X002(Z\_S05) WITH 'SHORTDUMP'.

\*실행하면 EXCEPTION DOCUMENT가 출력된다.

MESSAGE 'ERROR' TYPE 'E' DISPLAY LIKE 'W'.

\*ERROR를 출력하는 ERROR MESSAGE를 WARNING TYPE으로 보여준다.

MESSAGE 'ERROR' TYPE 'S' DISPLAY LIKE 'E'.

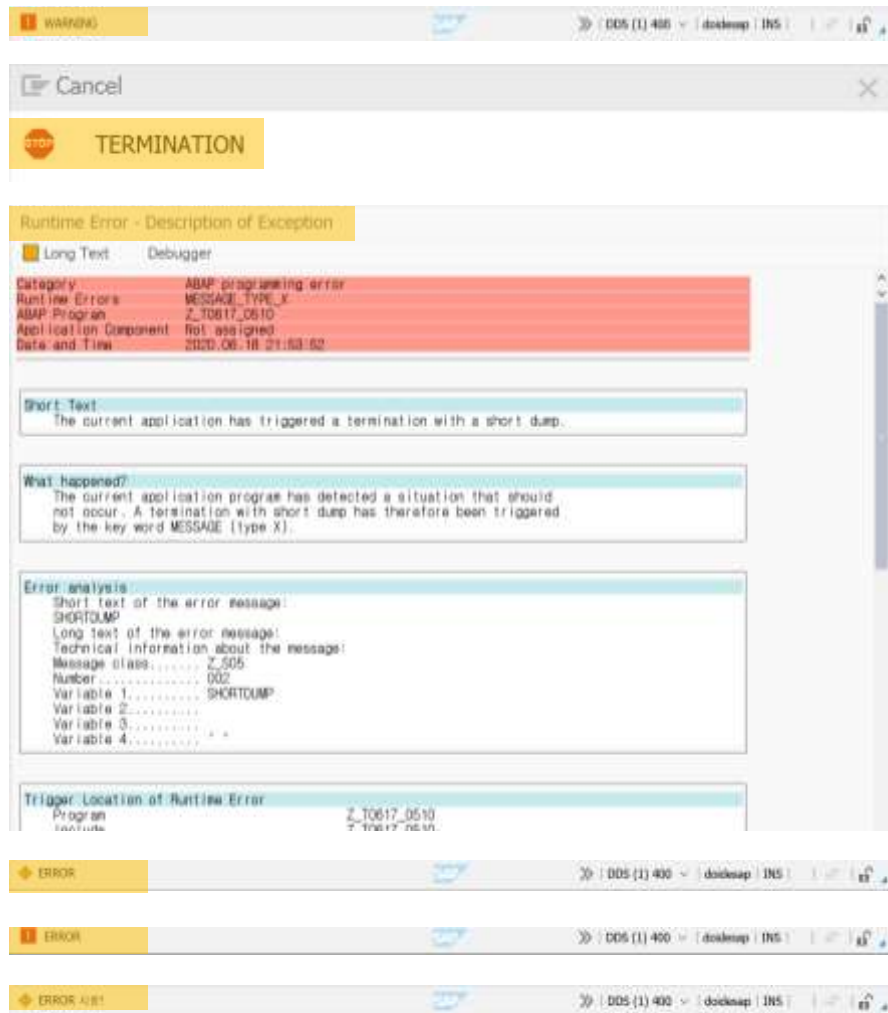
\*ERROR를 출력하는 ERROR MESSAGE를 DIALOG 화면에서 ERROR TYPE으로 보여준다.

\*\*제일 많이 사용하는 경우

MESSAGE TEXT-005 TYPE 'E' DISPLAY LIKE 'W'.

\*"대신에 TEXT SYMBOL이 올 수도 있다.





## 순환 반복 구문 |

ABAP에서 사용할 수 있는 순환 반복 구문은 DO ~ ENDDO, WHILE ~ ENDWHILE, LOOP ~ ENDLOOP 이 해당된다. 순환 구문 내에서 EXIT 명령을 만나면 순환 구문을 빠져나오며, CONTINUE 명령을 만나면 이후 스크립트를 수행하지 않으며 다음 순환을 실행한다. CHECK 명령은 값을 비교하여 TRUE 일 경우에만 이후 구문을 수행하고 거짓이면 다음 순환을 실행한다.

## DO ~ END DO

순환 횟수를 지정할 수 있는 구문으로 횟수 미지정시 무한 LOOP를 수행한다.

현재 순환 횟수는 시스템 변수 SY-INDEX에 저장된다.

DO 3 TIMES. ~~~ ENDDO.

## WHILE ~ ENDWHILE

WHILE 구문의 다음 표현식이 참이면 반복 순환을 지속하며 현재 순환 횟수는 시스템 변수 SY-INDEX에 저장된다.

WHILE GV\_FLAG = 'X'. ~~~ ENDWHILE.

## LOOP ~ ENDLOOP

INTERNAL TABLE의 라인을 차례대로 WORK AREA 또는 HEADER LINE으로 이동하는 순환 구문이다. 현재 순환 횟수는 시스템 변수 SY-TABIX에 저장된다. SY-TABIX는 INTERNAL TABLE의 라인 번호이다.

## SE80 | 0511 | 순환 반복 구문에 대해 알아보기

*\*DO WHILE 무한 반복문으로 탈출하기 위한 조건을 기재한다.*

*\*SELECT와 ENDSELECT는 특정 조건을 선택하기 위해 사용한다.*

*\*LOOP AT END LOOP 는 INTERNAL TABLE 을 ROW 만큼 반복한다.*

*\*DO*

*\*\*IF 문을 통해 조건문을 탈출한다.*

*\*\*\*F8 DEBUGGING을 통해 확인해보자.*

DATA GV\_INT1 TYPE I .

DO.

GV\_INT1 = GV\_INT1 + 1.

IF GV\_INT1 = 4.

EXIT.

ENDIF.

ENDDO.

WRITE:/ GV\_INT1.

*\*DO 구문에 조건을 걸으려면 TIMES를 사용한다.*

DO 5 TIMES.

WRITE:/ SY-INDEX.

ENDDO.

*\*WHILE*

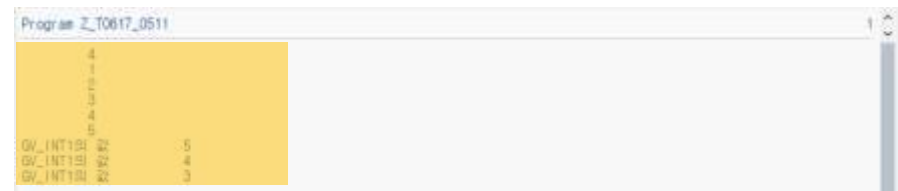
GV\_INT1 = 6.

WHILE GV\_INT1 > 3.

GV\_INT1 = GV\_INT1 - 1.

WRITE:/ 'GV\_INT1의 값' , GV\_INT1.

ENDWHILE.



## SUBROUTINE |

SUBROUTINE은 FORM으로 시작하여 END FORM으로 종료되는 구문을 의미하며, 스크립트의 모듈화, 재사용, 구조화를 주목적으로 한다. ABAP 프로그램에서는 PERFORM 구문을 이용한 SUBROUTINE으로 유사한 기능을 제공한다.이 외에 PARAMETER 값을 주고받을 수 있는 FUNCTION MODULE이 존재한다. 모듈화는 의미 있는 기능들을 모아 놓은 프로그램 블록을 의미하며 재사용이 가능한 특성을 가진다. 스크립트가 길면 유지보수 하기 어려우므로, 재사용의 목적이 아니더라도 기능별로 블록화 하여 프로그램을 구조화하는 것이 바람직하다. SUBROUTINE은 FORM으로 시작하여 END FORM으로 종료되는 구문을 의미한다. FORM은 프로그램의 내부 및 외부에서 호출할 수 있다.

SE80 | 0512 | MODULE을 생성하고 호출해보자.

*\*MODULE*

*\*공통적인 코드를 모아 모듈로 만드는 것.*

*\*모듈화 하면 한번에 수정이 가능하며, 타 사용자의 코드 이해가 쉬움.*

*\*유지보수 하는데 편리하다.*

*\*호출할 때 PERFORM을 사용한다.*

*\*PERFORM에 선언한 리스트 이름을 더블 클릭하면 FORM 화면을 불러와 자동 생성한다.*

PERFORM WRITE\_LIST.

PERFORM WRITE\_LIST.

FORM WRITE\_LIST.

WRITE / 'LIST OF AIRLINES'.

SKIP.

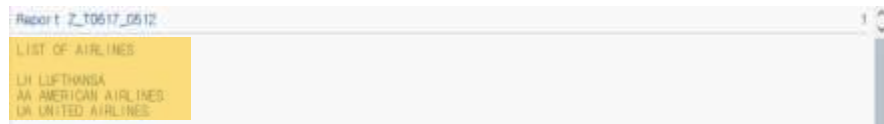
WRITE / 'LH LUFTHANSA'.

WRITE / 'AA AMERICAN AIRLINES'.

WRITE / 'UA UNITED AIRLINES'.

SKIP. ULINE.

ENDFORM.



SE80 | 0513 | 복습해보자

*\*STANDARD FIELD선언하는 것*

*\*\*LOCAL 은 TYPES를 사용하여 선언하는 것.*

*\*FIELD*

DATA F1 TYPE C.

DATA F2 TYPE C.

DATA F3 TYPE C.

*\*LOCAL*

TYPES T1 TYPE C.

TYPES T2 TYPE C.

TYPES T3 TYPE C.

DATA F4 TYPE T1.

DATA F5 TYPE T2.

DATA F6 TYPE T3.

*\*GLOBAL*

DATA G7 TYPE S\_CARR\_ID.

DATA G8 TYPE S\_CONN\_ID.

DATA G9 LIKE G7.

*\*STR*

```
DATA: BEGIN OF STR1,  
      F10 TYPE C,  
      F11 TYPE C,  
      F12 TYPE C,  
END OF STR1.
```

*\*LOCAL 참조하여 생성*

```
DATA: BEGIN OF STR2,  
      F13 TYPE T1,  
      F14 TYPE T2,  
      F15 TYPE T3,  
END OF STR2.
```

*\*GLOBAL 참조 하여 생성*

```
DATA: BEGIN OF STR3,  
      F16 LIKE G7,  
      F17 LIKE G8,  
      F18 LIKE G9,  
END OF STR3.
```

SE80 | 0514 | 복습해보자

*\*STRUCTURE STANDARD*

```
DATA: BEGIN OF GS_STR,  
      F1 TYPE C,  
      F2 TYPE I,  
      F3 TYPE N,  
END OF GS_STR.
```

*\*STRUCTURE LOCAL*

```
TYPES: BEGIN OF TY_STR,  
      T1 TYPE C,  
      T2 TYPE I,  
      T3 TYPE N,  
END OF TY_STR.
```

*\*LOCAL STRUCTURE 참조*

```
DATA GS_STR2 TYPE TY_STR.
```

*\*GLOBAL STRUCTURE 참조*

```
DATA GS_STR3 TYPE SCARR.
```

*\* LOCAL STRUCTURE을 참조하여 TABLE 생성*

```
DATA GT_TABLE TYPE TABLE OF TY_STR.
```

DATA GT\_TABLE2 LIKE GT\_TABLE.

*\*GLOBAL STRUCTURE을 참조하여 TABLE 생성*

DATA GT\_TABLE TYPE TABLE OF SCARR.

*\*GLOBAL TABLE을 참조하여 TABLE 생성*

DATA GT\_TABLE TYPE ZBC400\_T\_FLIGHTS.

## SELECT-OPTIONS |

PARAMETERS가 하나의 값만 입력 받을 수 있는 INPUT FIELD인 반면, SELET-OPTIONS는 2개의 INPUT 필드를 통해 다양한 조건 값인 SELECT CRITERIA를 입력 받을 수 있다.

## SELECTION-SCREEN |

PARAMETER와 SELECTION-OPTION을 사용하면 ABAP 프로그램이 자동으로 필드 내역과 길이를 조절하여 화면인 SELECTION-SCREEN을 생성한다. 만약, 시스템이 생성하는 화면을 커스텀하고 싶으면 SELECTOION-SCREEN 구문을 이용하면 된다.

SELECTION-SCREEN BEGIN OF LINE  
SELECTION-SCREEN END OF LINE

\*파라미터를 여러 개 묶어 한 라인으로 생성하며 라인에서 SELEC-OPTIONS, SELECT-SCREEN SKIP N 구문을 사용할 수 없다.

SELECTION-SCREEN SKIP N

빈 라인을 N개 삽입한다.

SELECTION-SCREEN BEGIN OF BLOCK B1.  
SELECTION-SCREEN END OF BLOCK B1.

PARAMETER, SELECT-OPTIONS 등 블록을 형성한다.

WITH FRAME | 프레임을 추가한다.

TITLE T1 | 프레임의 TITLE을 추가한다.

NO INTERVALS | 블록안의 SELECT-OPTIONS의 LOW 값만 보인다.

## SE80 | 0515 | SELECT-OPTIONS를 수행해보자

*\*WITH FRAME 을 통해 상자를 지정해줄 수 있다.*

*\*TITLE TEXT SYMBOLIC TEXT를 활용하여 TITLE을 지정할 수 있다.*

*\*TEXT ELEMENT 를 통해 SELECTION TEXT에서 입력 받는 곳의 NAME 를 지정할 수 있다.*

*\*SELECT-OPTION은 MULTI SELECTION 여부를 선택할 수 있다. (FROM TO 형태)*

*\*SELECT-OPTION은 DATA OBJECT를 참조하여 생성한다.*

*\*DEFAULT 는 해당 파라미터의 기본 값을 지정해 준다.*

*\*OBLIGATORY 는 필수 입력 값을 지정해 준다.*

*\*AS CHECKBOX는 CHECK BOX를 생성한다.*

\**RADIO GROUP에서 반드시 하나의 값은 CHECK를 해야 한다.*

PA\_RADI3 RADIOBUTTON GROUP RAD1.

SELECTION-SCREEN END OF BLOCK B1.

ABAP Editor: Change Report Z\_T0017\_0515

Package Explorer: Z\_PACKAGE\_0501

Report: Z\_T0017\_0515

Code:

```

1  *
2  * T0017_0515
3  *
4  *
5  *
6  * T0017_0515
7  *
8  *
9  * SYMBOLIC TEXT을 불러와서 TITLE을 지정함 = 50%
10 * SYT을 불러서 SELECTION TEXT인 시퀀스 조건 NAME을 세로
11 * TION = MULT SELECTION 시퀀스 조건 = 50% (FROM TO 시퀀스)
12 * TION = DATA OBJECT를 지정할 수 있음.
13 *
14 * IV을 불러와서 IV을 지정함.
15 *
16 * IOKX CHECK BOX를 지정함.
17 *
18 * TON GROUP = GROUP 조건을 세로 지정함.
19 *
20 * BUTTON = CHECK을 세로 지정함. DEFAULT 값을 세로 지정함.
21 *
22 *
23 * SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT TO1
24 * IS PA_INT1 TYPE (OBLIGATORY DEFAULT 1)
25 * PA_INT1 TYPE 1
26 * PA_CH_45 CHECKBOX DEFAULT 'X'
27 * PA_RAD1 RADIOBUTTON GROUP RAD1 DEFAULT 'X'
28 * PA_RAD2 RADIOBUTTON GROUP RAD1
29 * PA_RAD3 RADIOBUTTON GROUP RAD1
30 *
31 *
32 *
33 *
34 *
35 *
36 *
37 *
38 *
39 *
40 *
41 *
42 *
43 *
44 *
45 *
46 *
47 *
48 *
49 *
50 *
51 *
52 *
53 *
54 *
55 *
56 *
57 *
58 *
59 *
60 *
61 *
62 *
63 *
64 *
65 *
66 *
67 *
68 *
69 *
70 *
71 *
72 *
73 *
74 *
75 *
76 *
77 *
78 *
79 *
80 *
81 *
82 *
83 *
84 *
85 *
86 *
87 *
88 *
89 *
90 *
91 *
92 *
93 *
94 *
95 *
96 *
97 *
98 *
99 *
100 *
101 *
102 *
103 *
104 *
105 *
106 *
107 *
108 *
109 *
110 *
111 *
112 *
113 *
114 *
115 *
116 *
117 *
118 *
119 *
120 *
121 *
122 *
123 *
124 *
125 *
126 *
127 *
128 *
129 *
130 *
131 *
132 *
133 *
134 *
135 *
136 *
137 *
138 *
139 *
140 *
141 *
142 *
143 *
144 *
145 *
146 *
147 *
148 *
149 *
150 *
151 *
152 *
153 *
154 *
155 *
156 *
157 *
158 *
159 *
160 *
161 *
162 *
163 *
164 *
165 *
166 *
167 *
168 *
169 *
170 *
171 *
172 *
173 *
174 *
175 *
176 *
177 *
178 *
179 *
180 *
181 *
182 *
183 *
184 *
185 *
186 *
187 *
188 *
189 *
190 *
191 *
192 *
193 *
194 *
195 *
196 *
197 *
198 *
199 *
200 *
201 *
202 *
203 *
204 *
205 *
206 *
207 *
208 *
209 *
210 *
211 *
212 *
213 *
214 *
215 *
216 *
217 *
218 *
219 *
220 *
221 *
222 *
223 *
224 *
225 *
226 *
227 *
228 *
229 *
230 *
231 *
232 *
233 *
234 *
235 *
236 *
237 *
238 *
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *
258 *
259 *
260 *
261 *
262 *
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 *
271 *
272 *
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *
324 *
325 *
326 *
327 *
328 *
329 *
330 *
331 *
332 *
333 *
334 *
335 *
336 *
337 *
338 *
339 *
340 *
341 *
342 *
343 *
344 *
345 *
346 *
347 *
348 *
349 *
350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *
390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 *
439 *
440 *
441 *
442 *
443 *
444 *
445 *
446 *
447 *
448 *
449 *
450 *
451 *
452 *
453 *
454 *
455 *
456 *
457 *
458 *
459 *
460 *
461 *
462 *
463 *
464 *
465 *
466 *
467 *
468 *
469 *
470 *
471 *
472 *
473 *
474 *
475 *
476 *
477 *
478 *
479 *
480 *
481 *
482 *
483 *
484 *
485 *
486 *
487 *
488 *
489 *
490 *
491 *
492 *
493 *
494 *
495 *
496 *
497 *
498 *
499 *
500 *
501 *
502 *
503 *
504 *
505 *
506 *
507 *
508 *
509 *
510 *
511 *
512 *
513 *
514 *
515 *
516 *
517 *
518 *
519 *
520 *
521 *
522 *
523 *
524 *
525 *
526 *
527 *
528 *
529 *
530 *
531 *
532 *
533 *
534 *
535 *
536 *
537 *
538 *
539 *
540 *
541 *
542 *
543 *
544 *
545 *
546 *
547 *
548 *
549 *
550 *
551 *
552 *
553 *
554 *
555 *
556 *
557 *
558 *
559 *
560 *
561 *
562 *
563 *
564 *
565 *
566 *
567 *
568 *
569 *
570 *
571 *
572 *
573 *
574 *
575 *
576 *
577 *
578 *
579 *
580 *
581 *
582 *
583 *
584 *
585 *
586 *
587 *
588 *
589 *
590 *
591 *
592 *
593 *
594 *
595 *
596 *
597 *
598 *
599 *
600 *
601 *
602 *
603 *
604 *
605 *
606 *
607 *
608 *
609 *
610 *
611 *
612 *
613 *
614 *
615 *
616 *
617 *
618 *
619 *
620 *
621 *
622 *
623 *
624 *
625 *
626 *
627 *
628 *
629 *
630 *
631 *
632 *
633 *
634 *
635 *
636 *
637 *
638 *
639 *
640 *
641 *
642 *
643 *
644 *
645 *
646 *
647 *
648 *
649 *
650 *
651 *
652 *
653 *
654 *
655 *
656 *
657 *
658 *
659 *
660 *
661 *
662 *
663 *
664 *
665 *
666 *
667 *
668 *
669 *
670 *
671 *
672 *
673 *
674 *
675 *
676 *
677 *
678 *
679 *
680 *
681 *
682 *
683 *
684 *
685 *
686 *
687 *
688 *
689 *
690 *
691 *
692 *
693 *
694 *
695 *
696 *
697 *
698 *
699 *
700 *
701 *
702 *
703 *
704 *
705 *
706 *
707 *
708 *
709 *
710 *
711 *
712 *
713 *
714 *
715 *
716 *
717 *
718 *
719 *
720 *
721 *
722 *
723 *
724 *
725 *
726 *
727 *
728 *
729 *
730 *
731 *
732 *
733 *
734 *
735 *
736 *
737 *
738 *
739 *
740 *
741 *
742 *
743 *
744 *
745 *
746 *
747 *
748 *
749 *
750 *
751 *
752 *
753 *
754 *
755 *
756 *
757 *
758 *
759 *
760 *
761 *
762 *
763 *
764 *
765 *
766 *
767 *
768 *
769 *

```