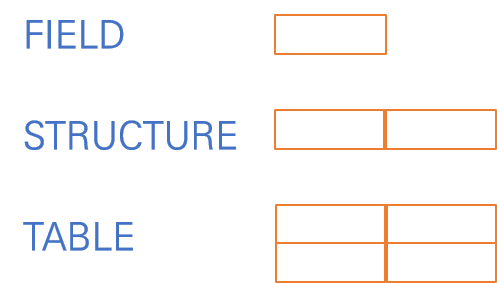
0617 | ABAP

DATA OBJECT | DATA 값을 참고하여 값을 저장할 수 있는 변수



\*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.

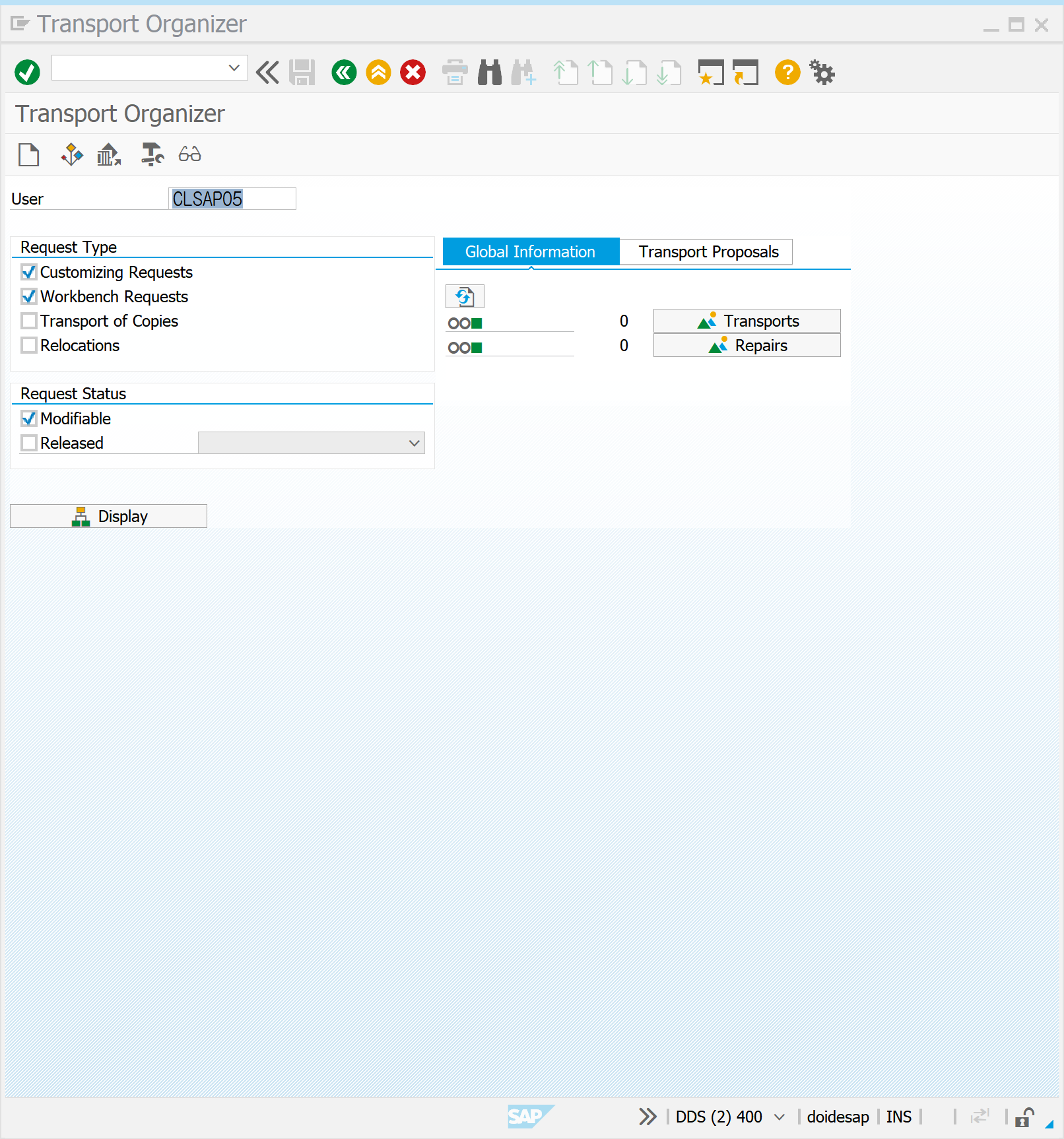
STANDARD |

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

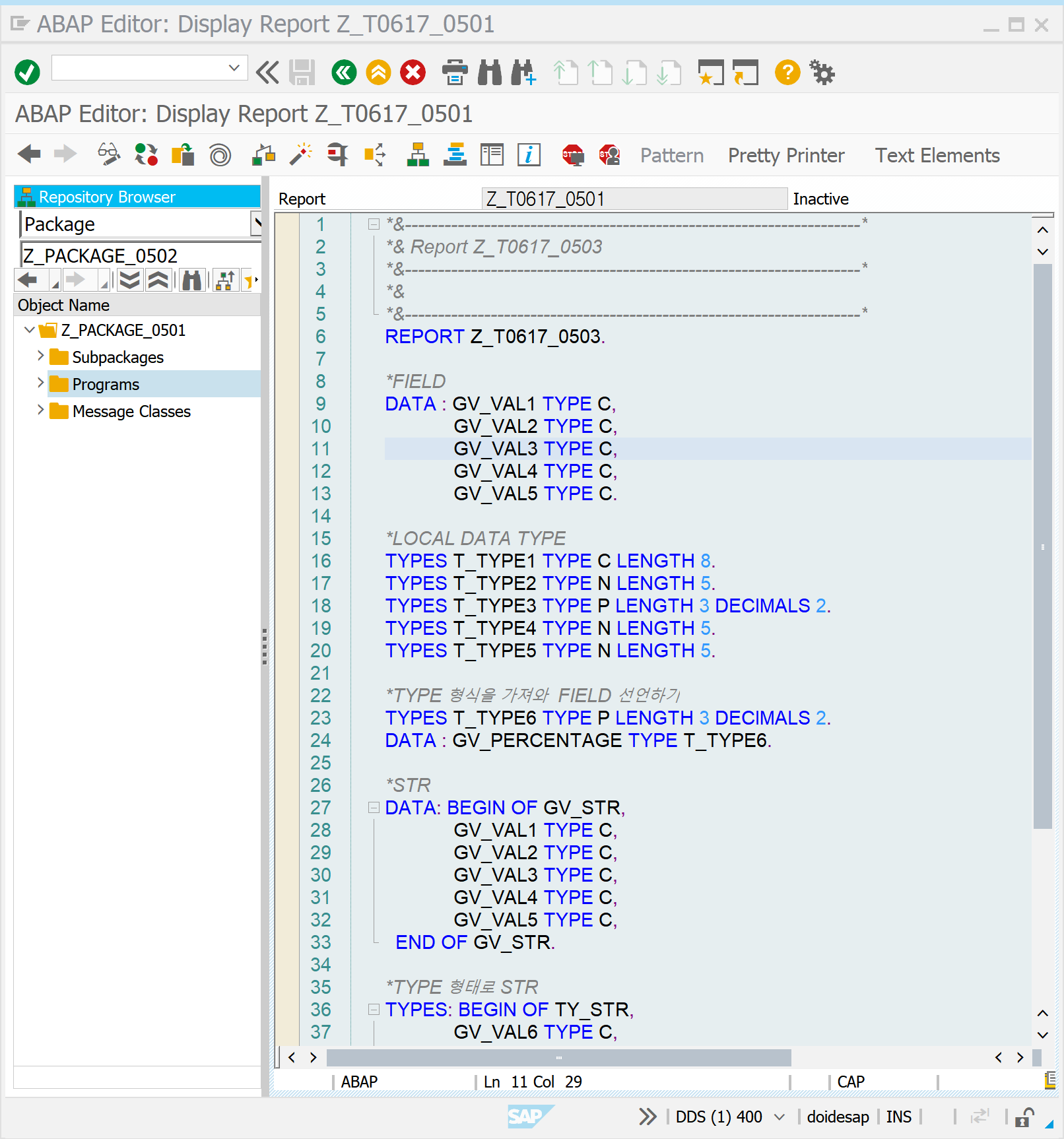
INTERNAL TABLE | ABAP 핵심 기술로 단일 프로그램 내에서 정의하여 사용할 수 있는 LOCAL TABLE로 프로그램 개발 및 유지보수를 좀 더 쉽고 편리하게 할 수 있다.

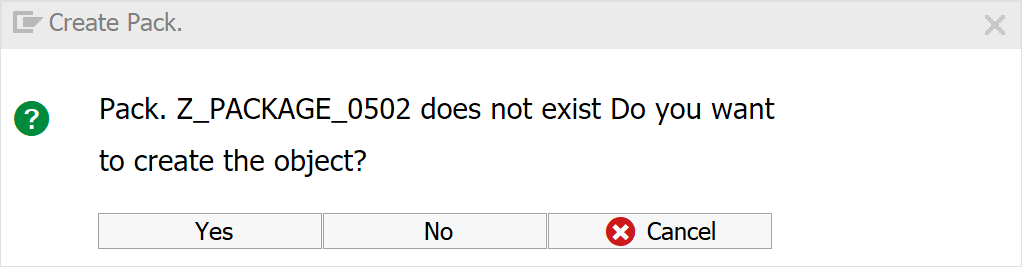
SE09 | CTS | CHANGE AND TRANSACTION SYSTEM

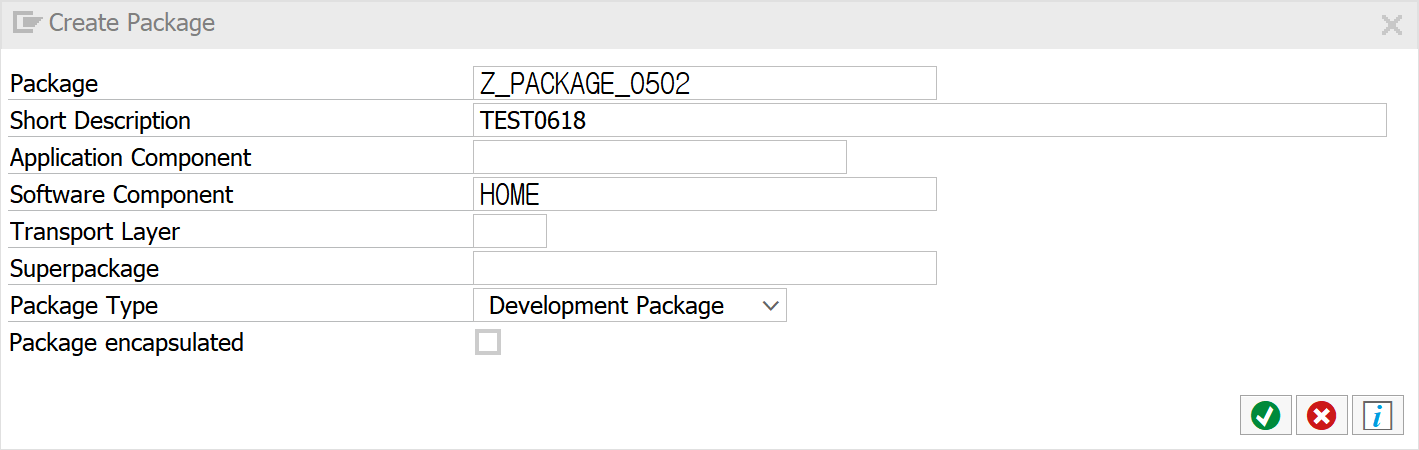
DEVELOPMENT SYSTEM을 PRODUCTION SYSTEM으로 이관하는 것을 의미.

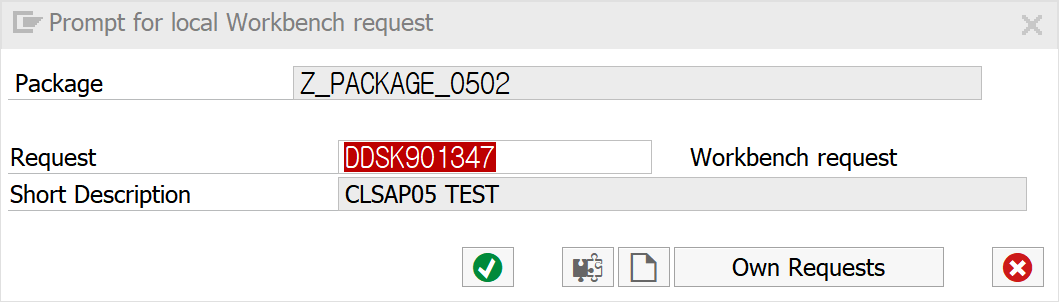


SE80 | PACKAGE | PACKAGE 를 생성하자









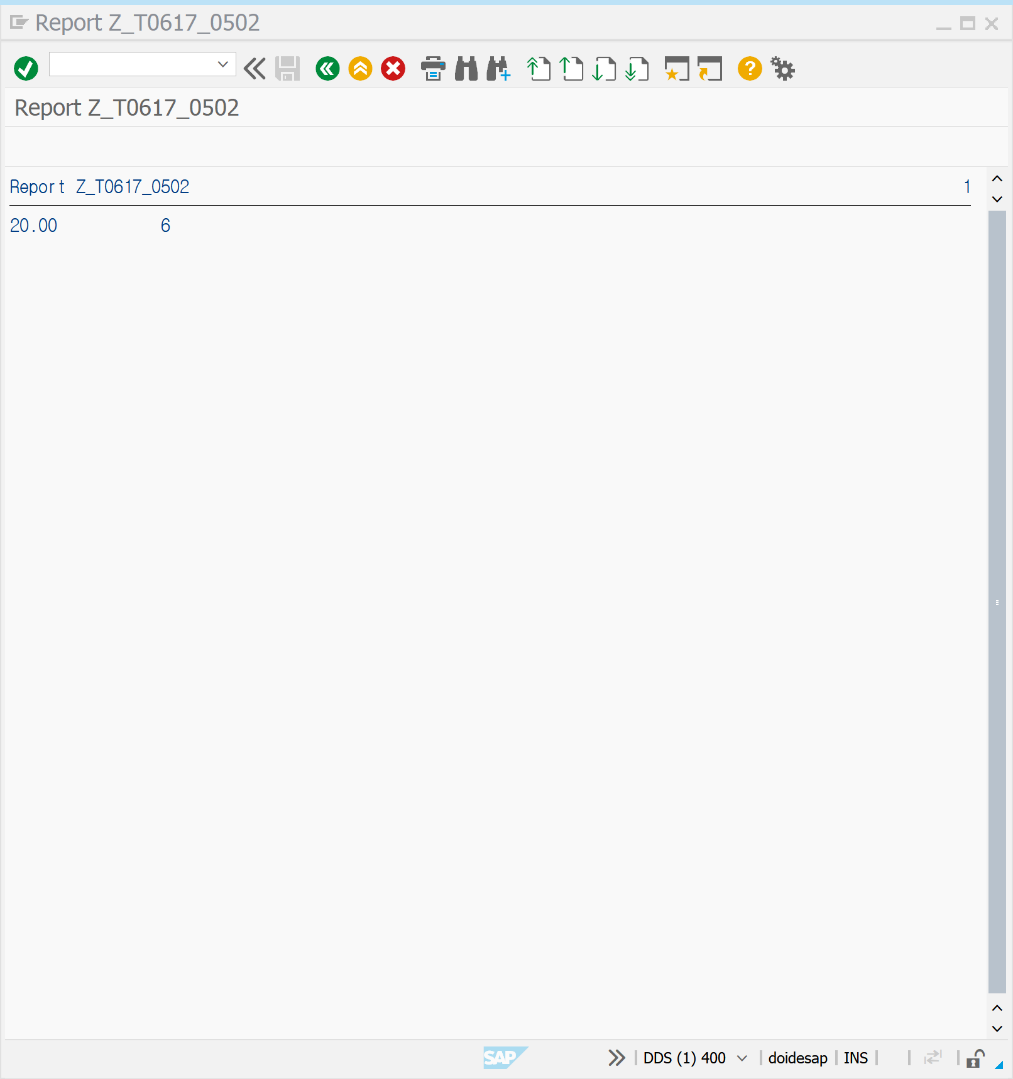
\*만약 REQUEST 가 존재하지 않는 경우 새롭게 생성해서 진행한다.

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.

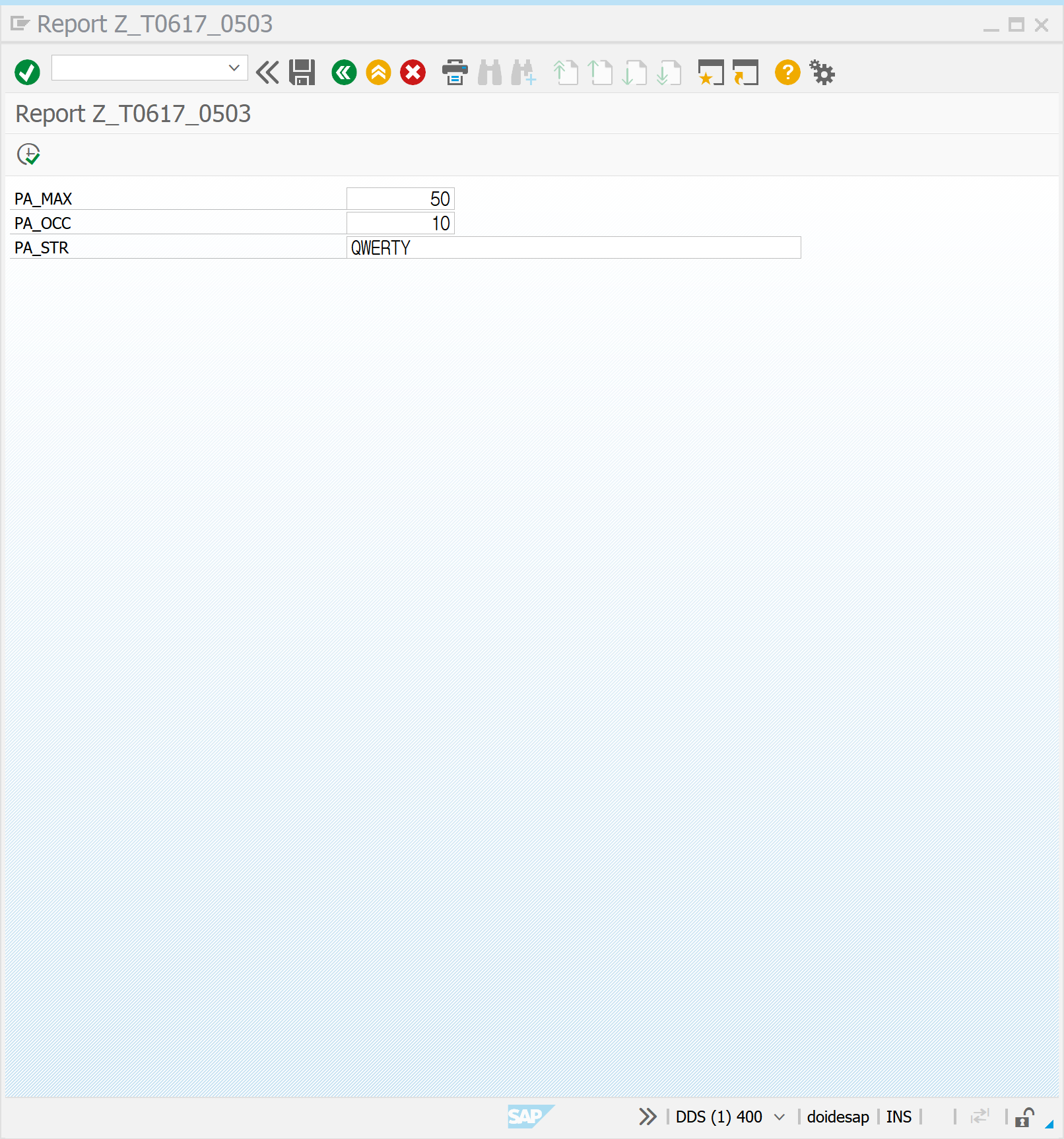
SE80 | 0502 | 어제 배운 내용을 복습해보자

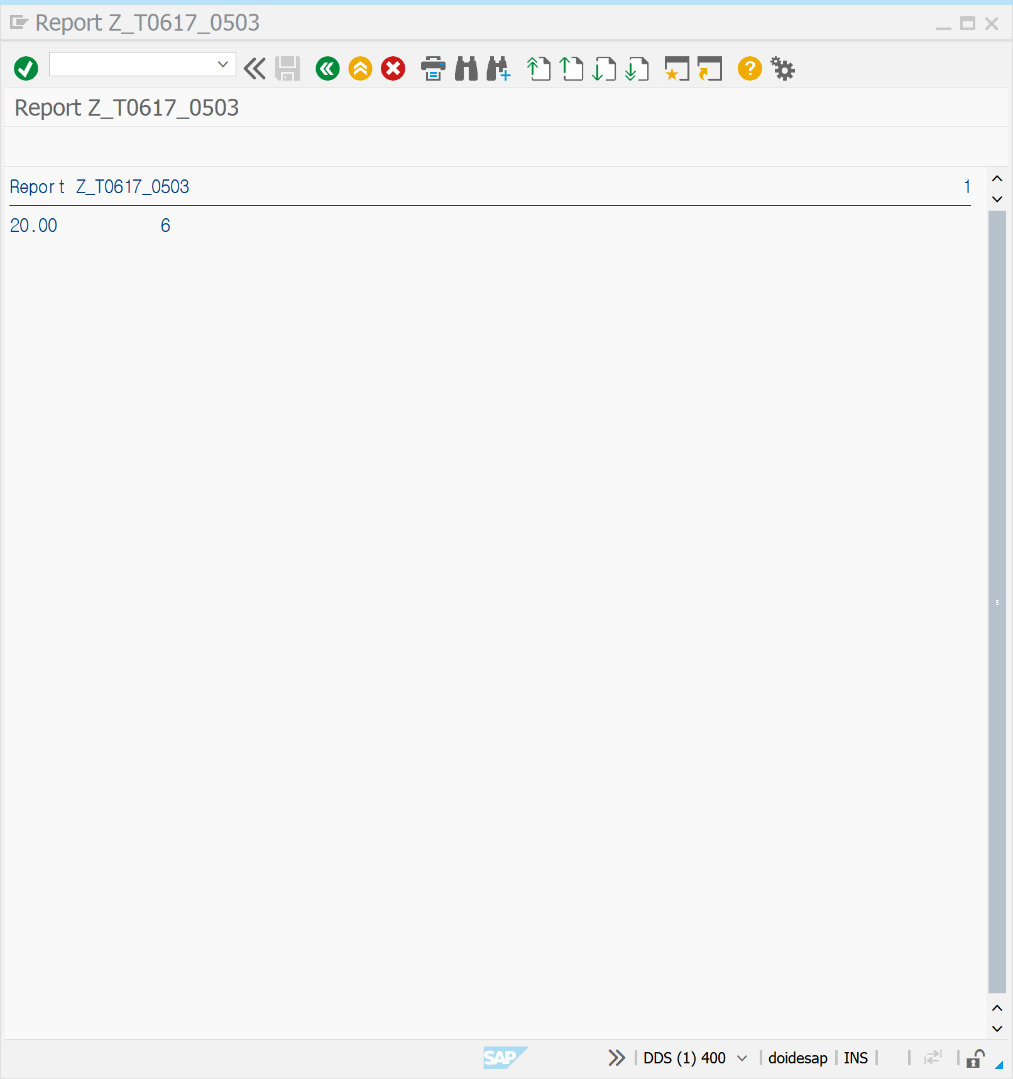
\*GLOBAL STRUCTURE 내의 FIELD를 참조하여 FIELD를 선언하자.  
\*더블클릭으로 TYPE을 확인할 수 있다.  
DATA: GV\_MAX TYPE ZSBC400FOCC-SEATSMAX,  
            GV\_OCC TYPE ZSBC400FOCC-SEATSOCC,  
            GV\_PERCENTAGE TYPE ZSBC400FOCC-PERCENTAGE,  
            GV\_LENGTH TYPE I,  
            GV\_STRING TYPE STRING.  
  
\*변수의 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.





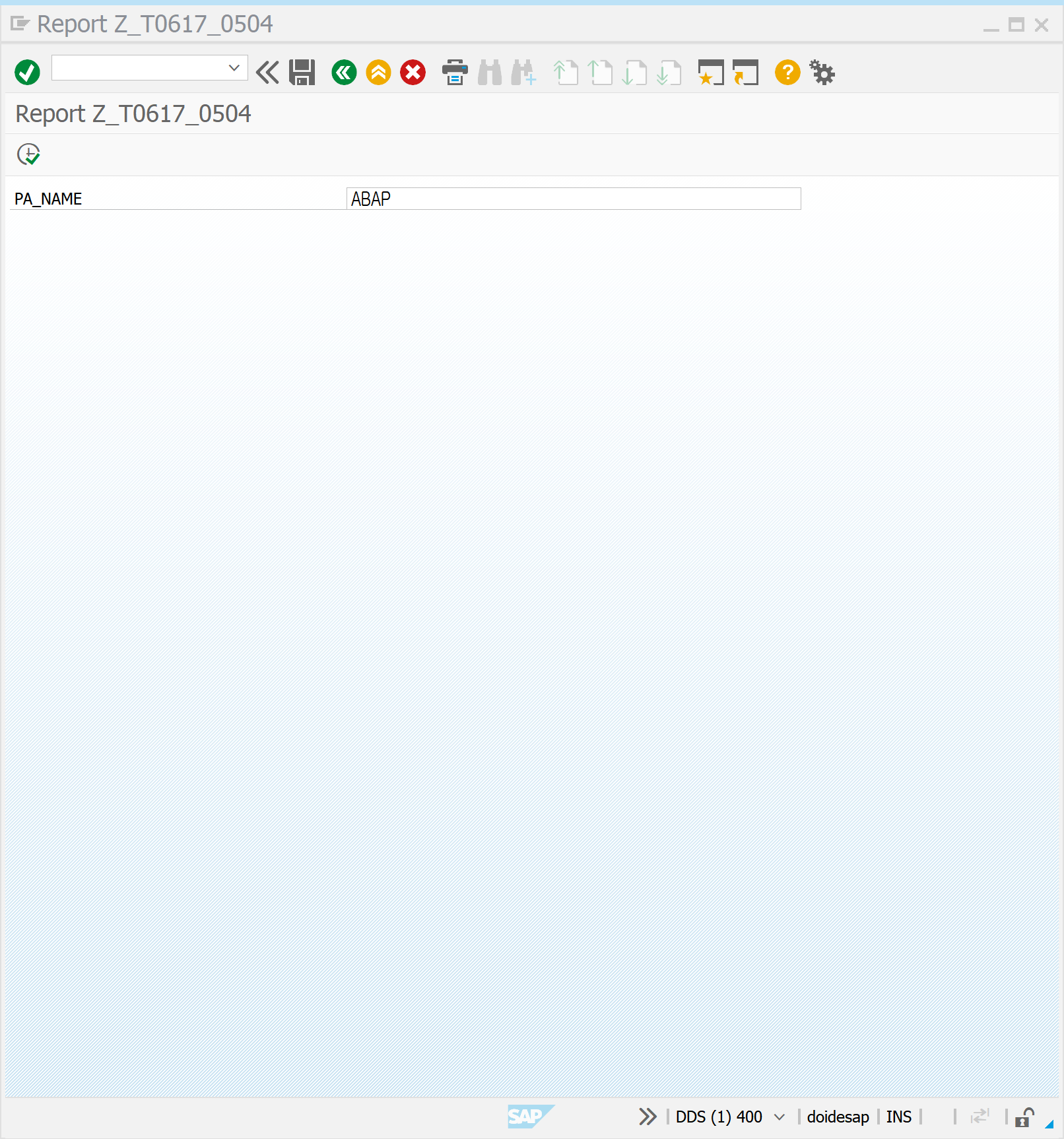
INTERNAL TABLE | DYNAMIC DATA OBJECT

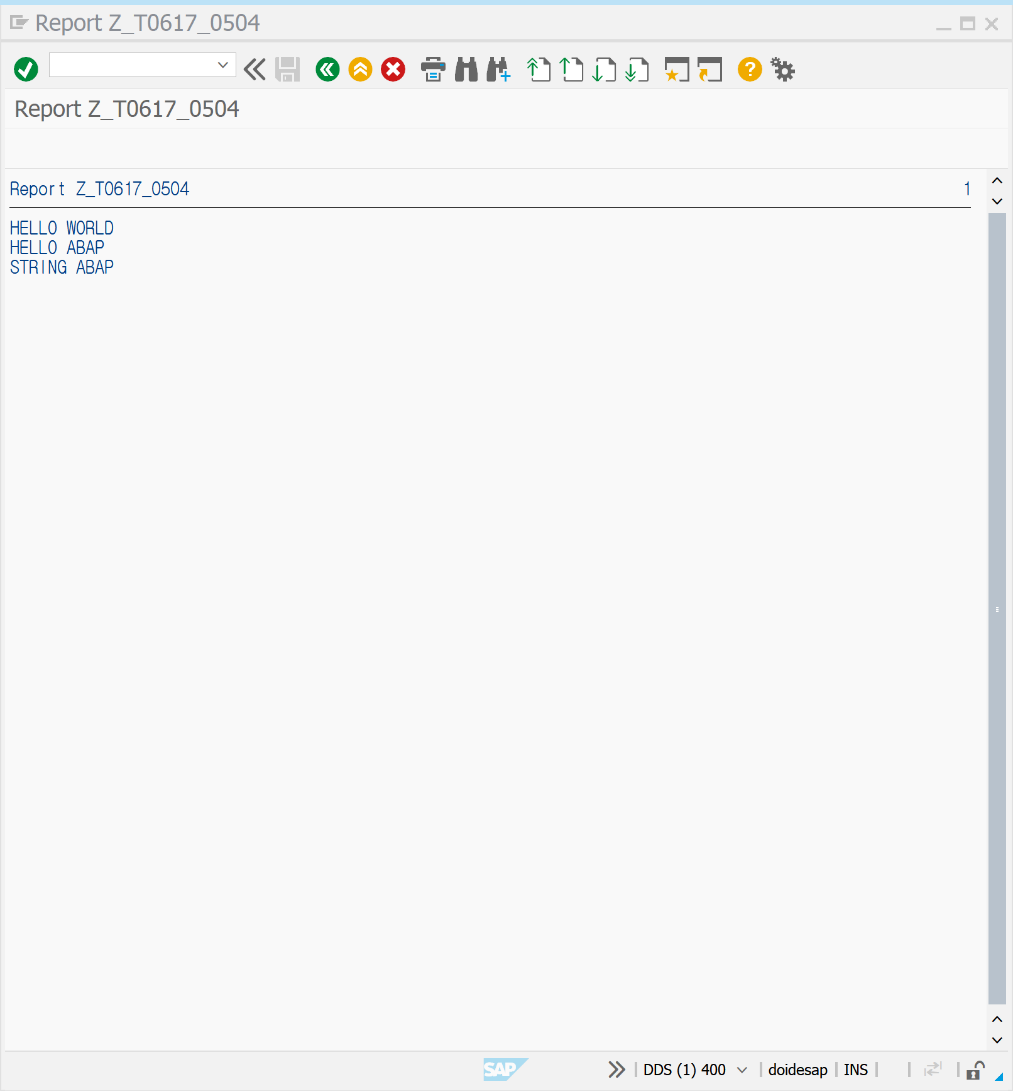
ABAP 핵심 기술로 단일 프로그램 내에서 정의하여 사용할 수 있는 LOCAL TABLE로 프로그램 개발 및 유지보수를 좀 더 쉽고 편리하게 할 수 있다. INTERNAL TABLE은 DYNAMIC DATA OBJECT 동적인 구조체 배열로, TYPE 구문을 기반으로 INTIAL SIZE 구문을 사용하여 테이블 크기만을 선언하고 MEMORY에 LOAD 하지 않음을 의미한다. 즉, TABLE의 형태인 크기만을 선언하기 때문에 INSERT 또는 APPEND 를 사용하여 LINE이 추가될 때마다 MEMORY에 LOAD 한다. 이는 INITIAL 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.



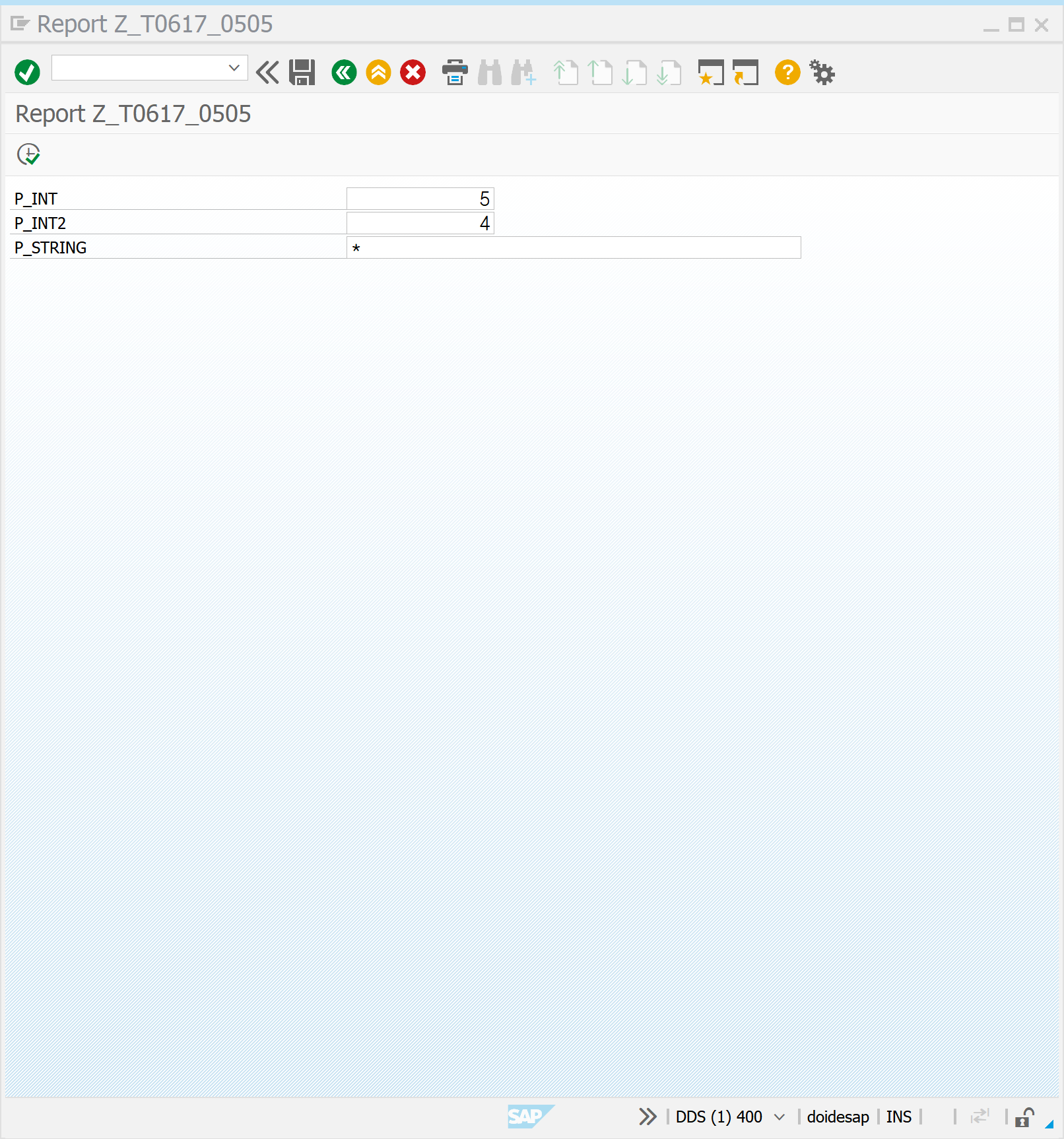


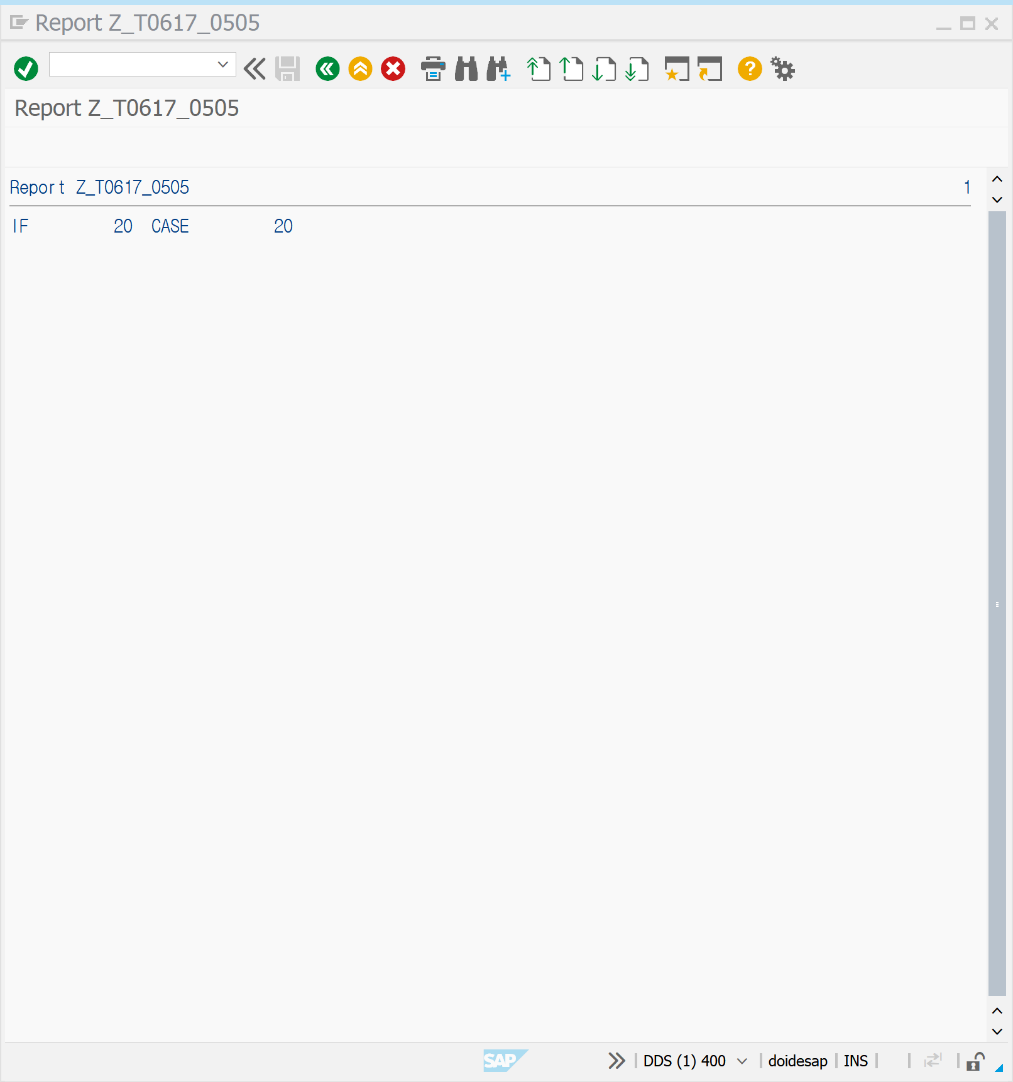
PARAMETERS | 사용자가 값을 입력하도록 INPUT FIELD 를 정의한다. PARAMETERS는 1개의 값만 입력 받을 수 있다.

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

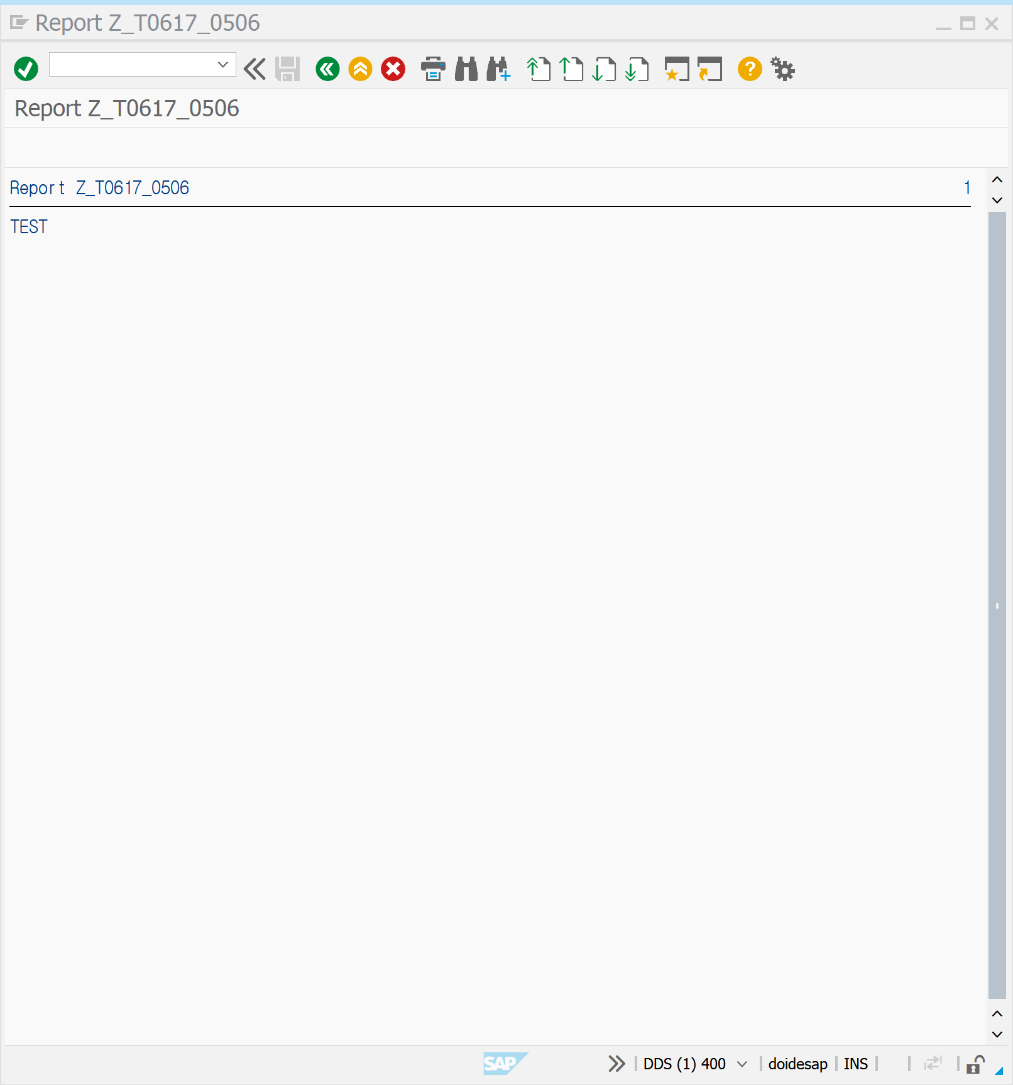




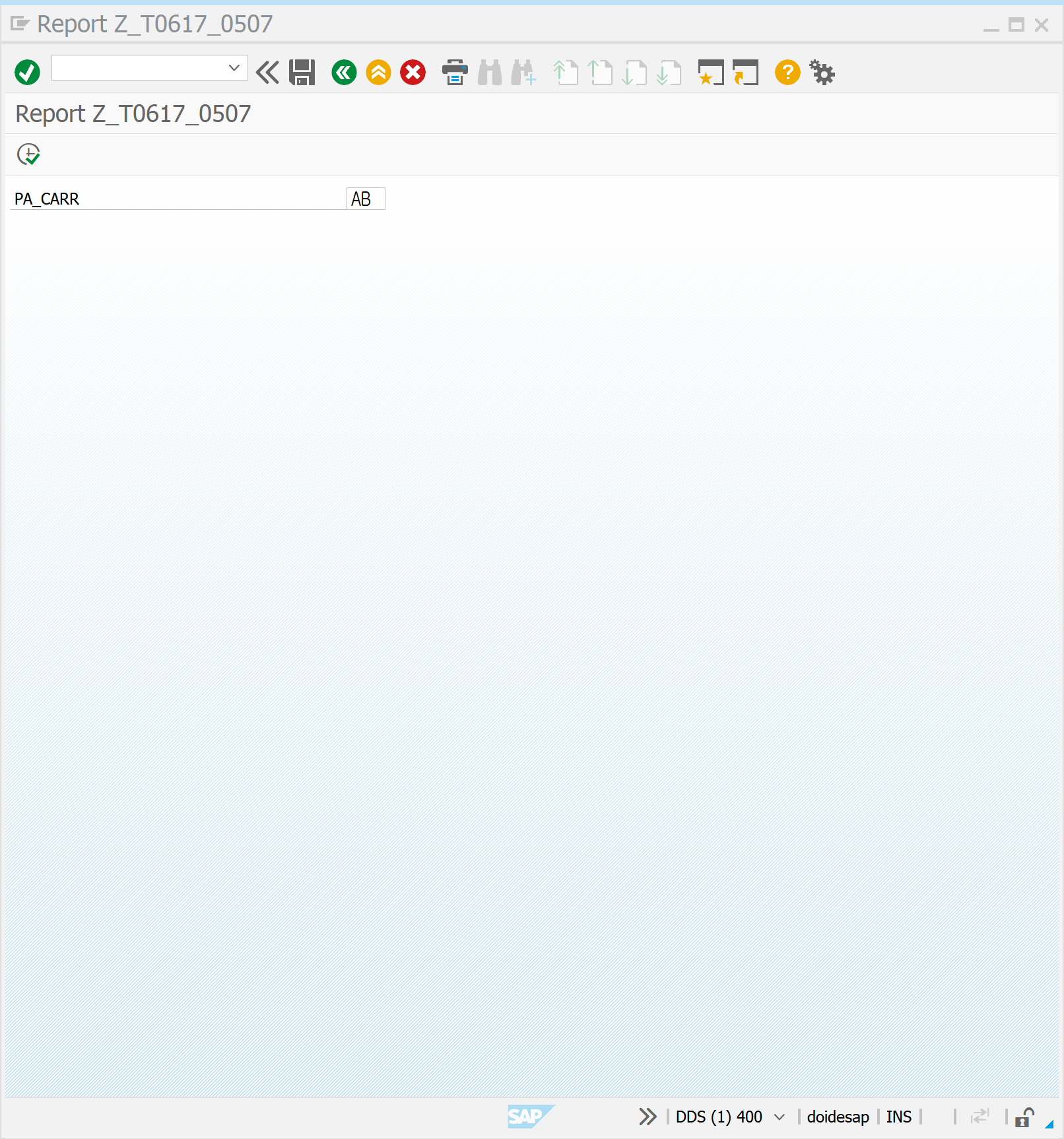
PARAMETERS: P\_INT TYPE I,  
                     P\_INT2 TYPE I,  
                     P\_STRING TYPE STRING.  
DATA RESULT TYPE I.  
  
IF P\_STRING = '+'.  
      RESULT = P\_INT + P\_INT2.  
      WRITE : 'IF',RESULT.  
ELSEIF P\_STRING = '-'.  
      RESULT = P\_INT - P\_INT2.  
      WRITE : 'IF',RESULT.  
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.

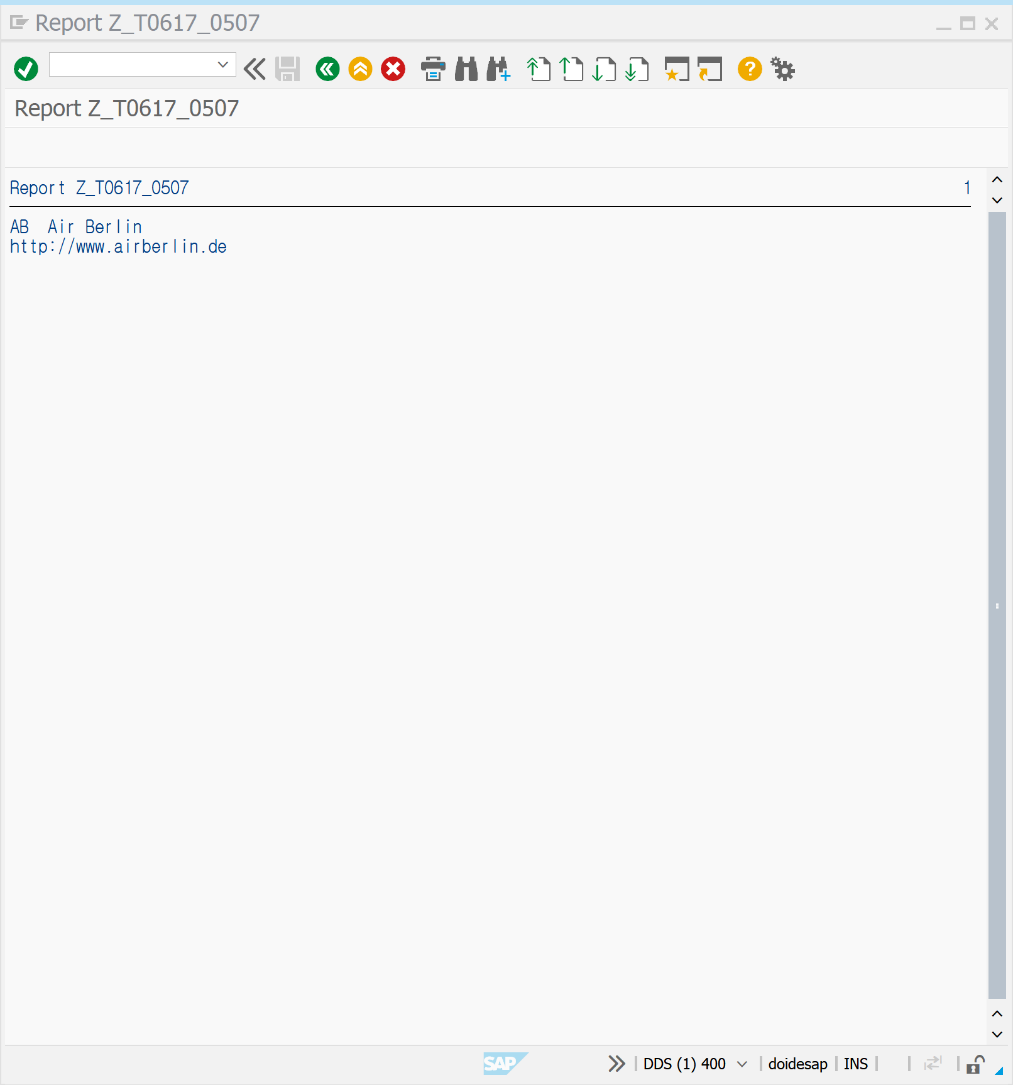
SE80 | 0506 | 비교 연산자를 활용해 보자

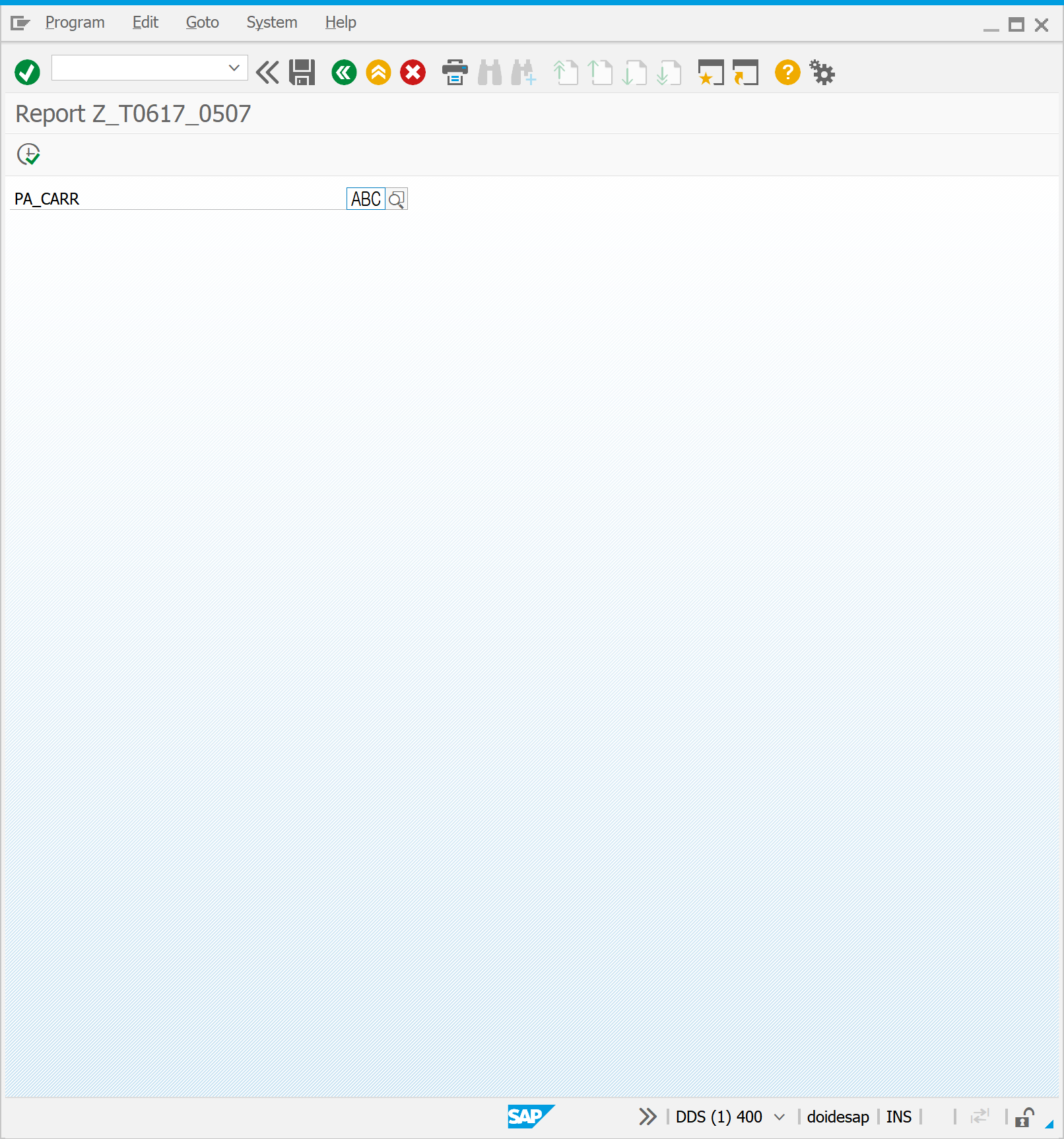
DATA GV\_STR TYPE STRING.  
GV\_STR = 'ABC'.  
  
\*&& 문자열을 한  문자열로 만들 수 있다.  
\*\*문자열 비교 연산자에는 CA가 있다.  
\*\*CA는 포함한다면 을 의미한다.  
IF 'LIT1' && GV\_STR CA 'ABC'.  
\*즉 LIT1과 GV\_STR을 한 문자열로 만든 LIT1ABC에 ABC가 포함 된다면 을 의미한다.  
     WRITE: /'TEST'.  
ENDIF.

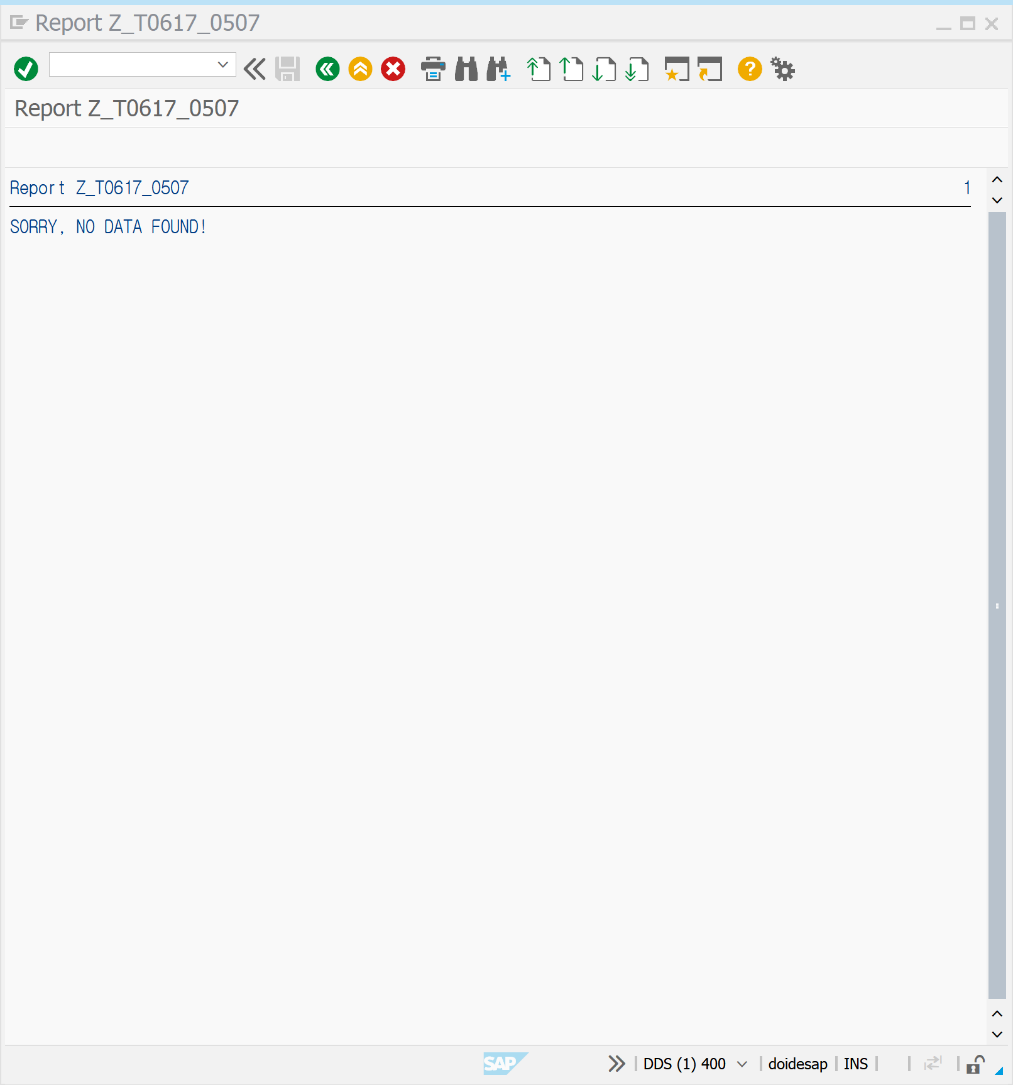


SE80 | 0507 | SY-SUBRC를 통해 로직 수행 여부를 확인하자









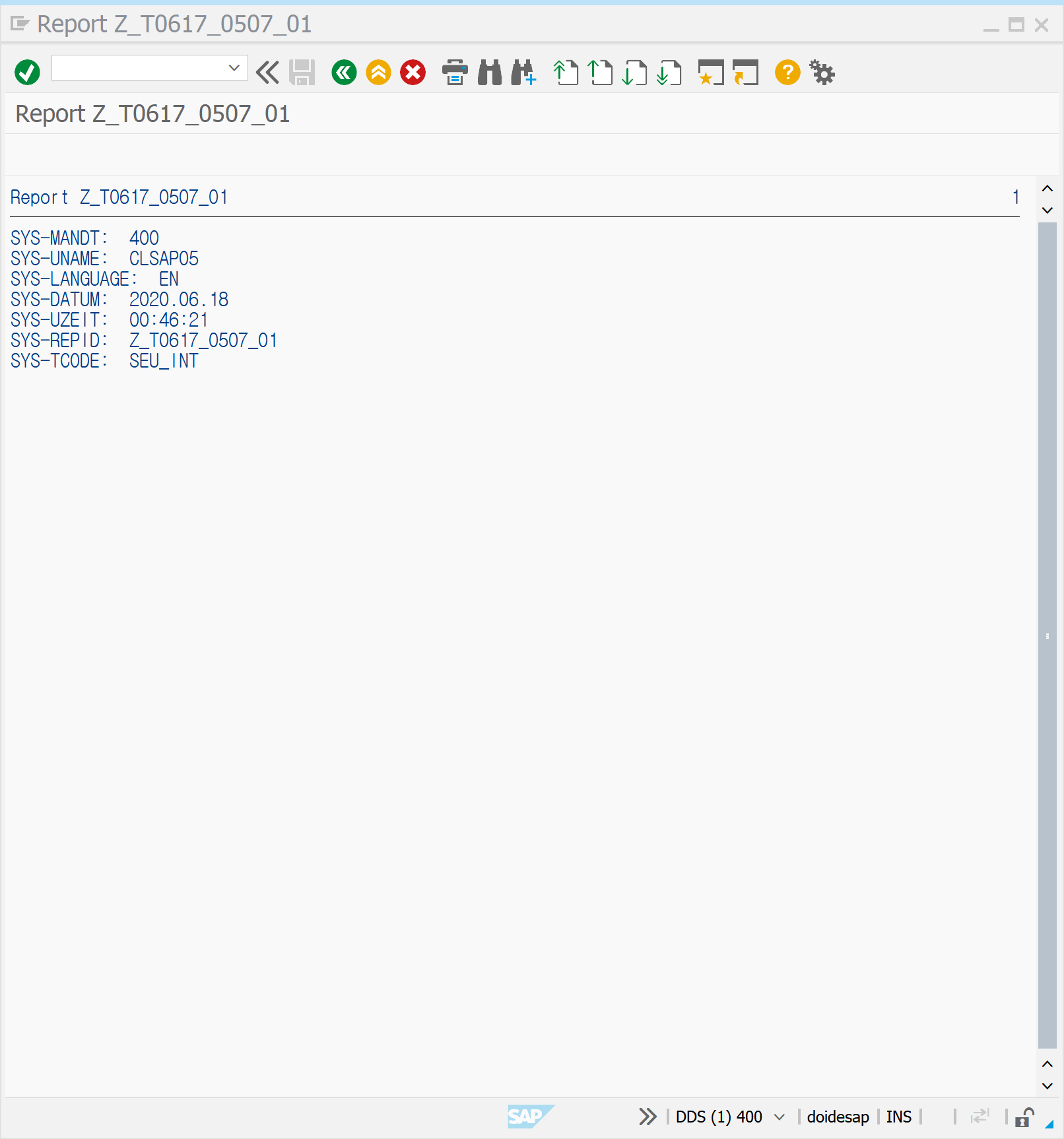
SE80 | 0507 | SY-SUBRC를 통해 로직 수행 여부를 확인하자

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 |

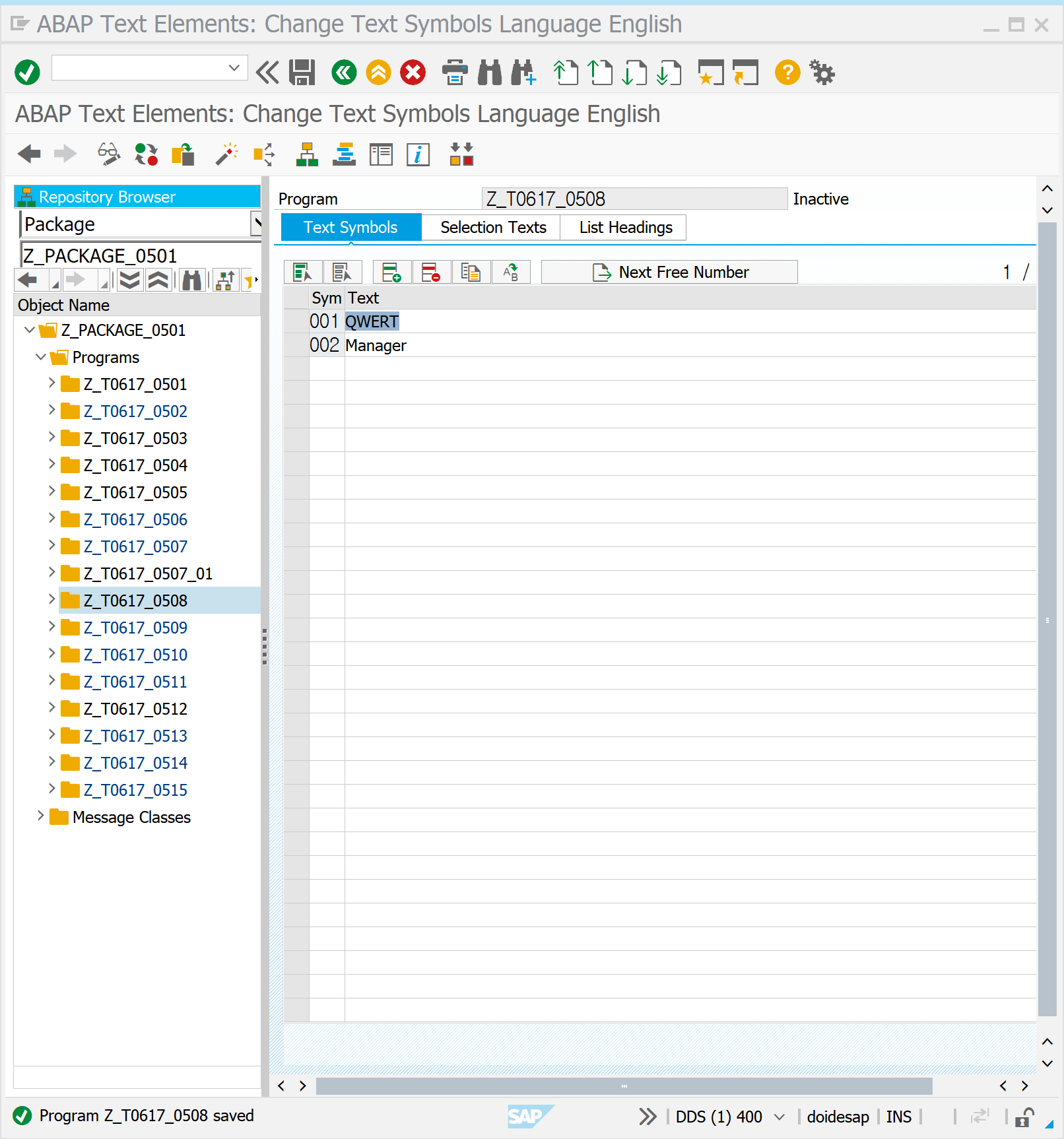
\*SYSTEM FIELD  
WRITE: / 'SYS-MANDT: ', SY-MANDT, / 'SYS-UNAME: ', SY-UNAME,  / 'SYS-LANGUAGE: ', SY-LANGU,  
             / 'SYS-DATUM: ', SY-DATUM,  / 'SYS-UZEIT: ', SY-UZEIT, / 'SYS-REPID: ', SY-REPID,/ 'SYS-TCODE: ', SY-TCODE.



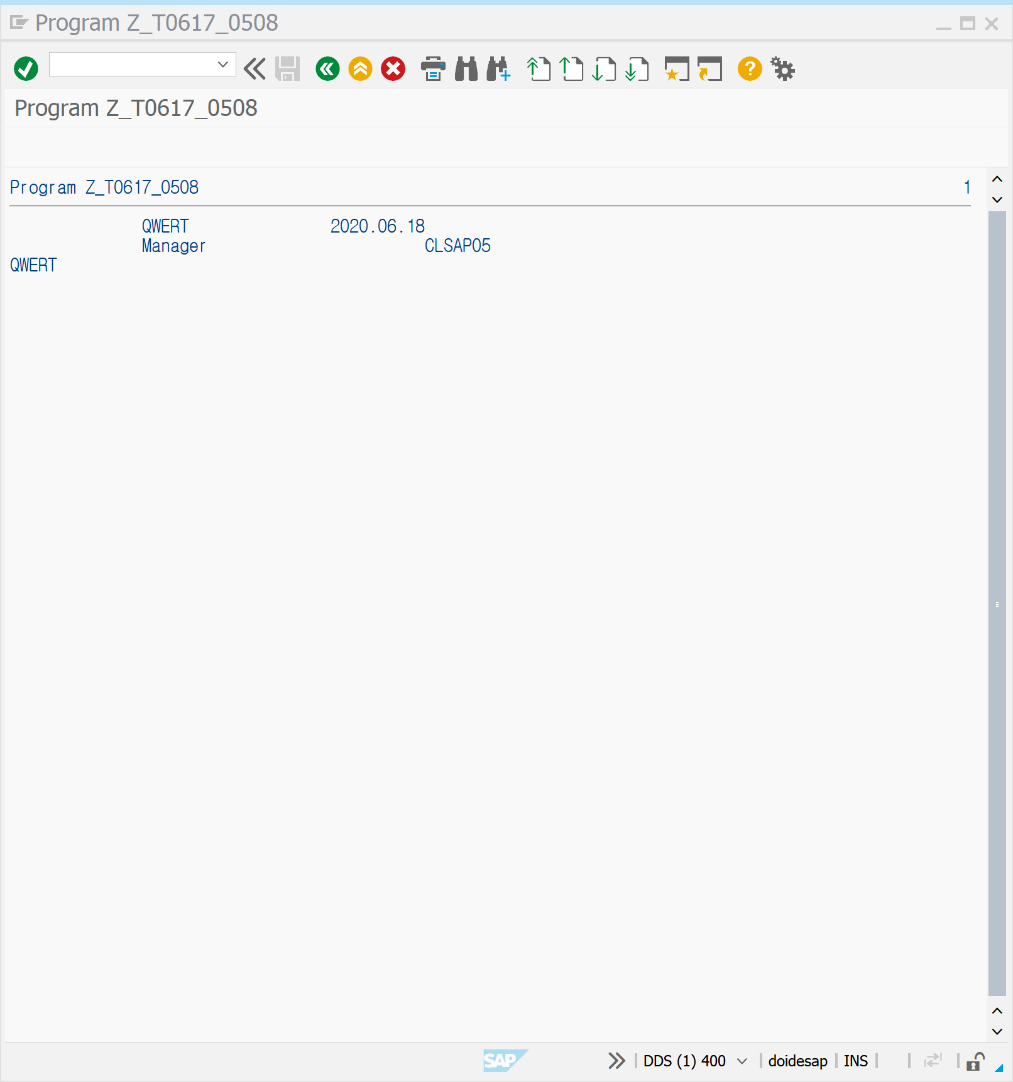
SE80 | 0508 | SYSTEM FIELD에 대해 알아보자.

\*TEX-001을 더블 클릭 후 TEXT를 입력한다.

\*MANAGER을 더블 클릭하면 자동으로 값이 입력된다.



\*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.



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

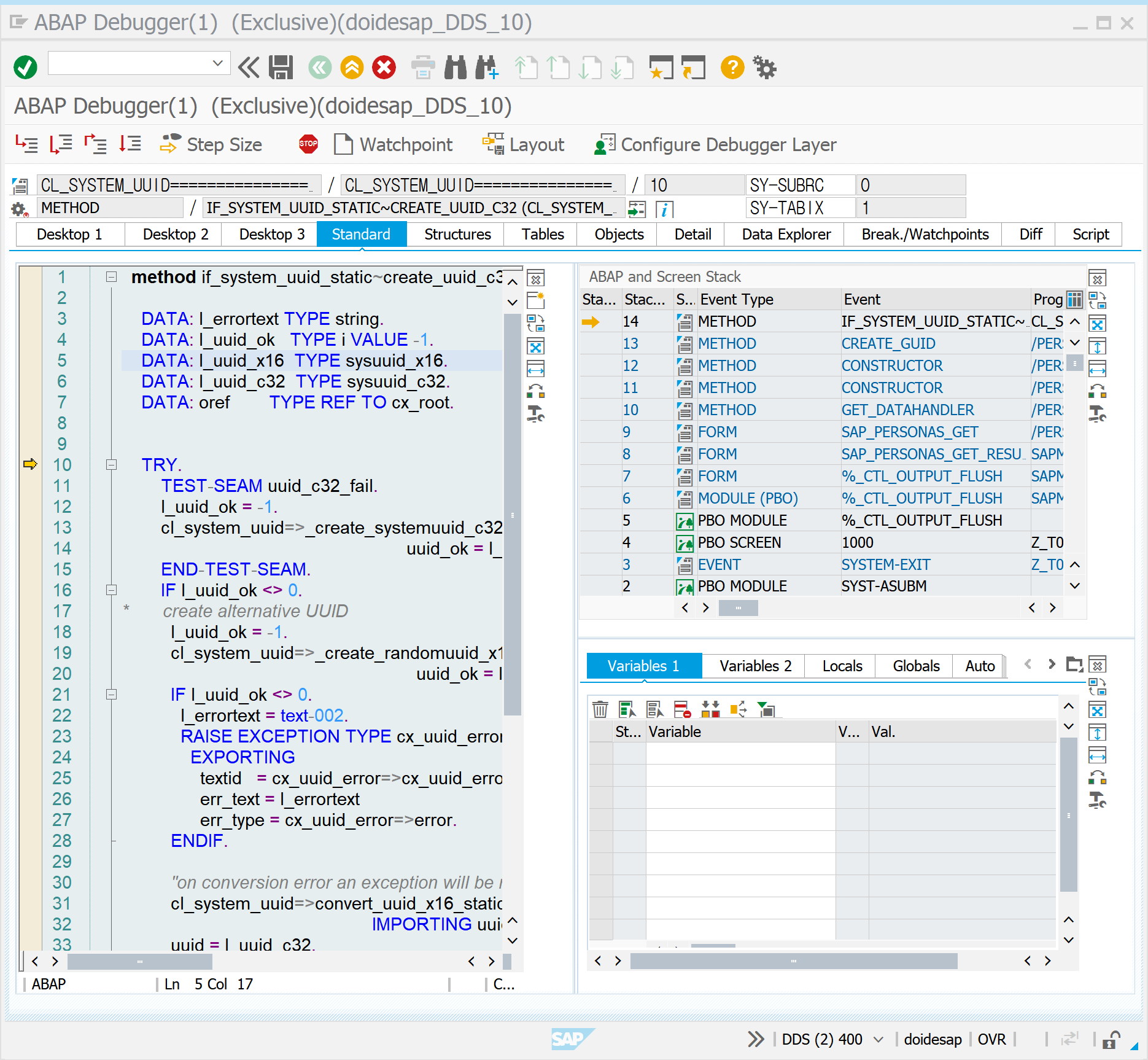
DEBUGGING | 다음 순서로 이동 단축 키

|  |  |
| --- | --- |
| 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 |

\*SUBLOUTINE을 만나면 실행 후 다음 단계로 이동한다. 즉, PERFORM 구문에서 F6를 입력하면 FORM구문의 DEBUGGING을 건너뛰고, 다음라인으로 이동하며 F5를 입력하면 FORM 구문으로 이동한다.

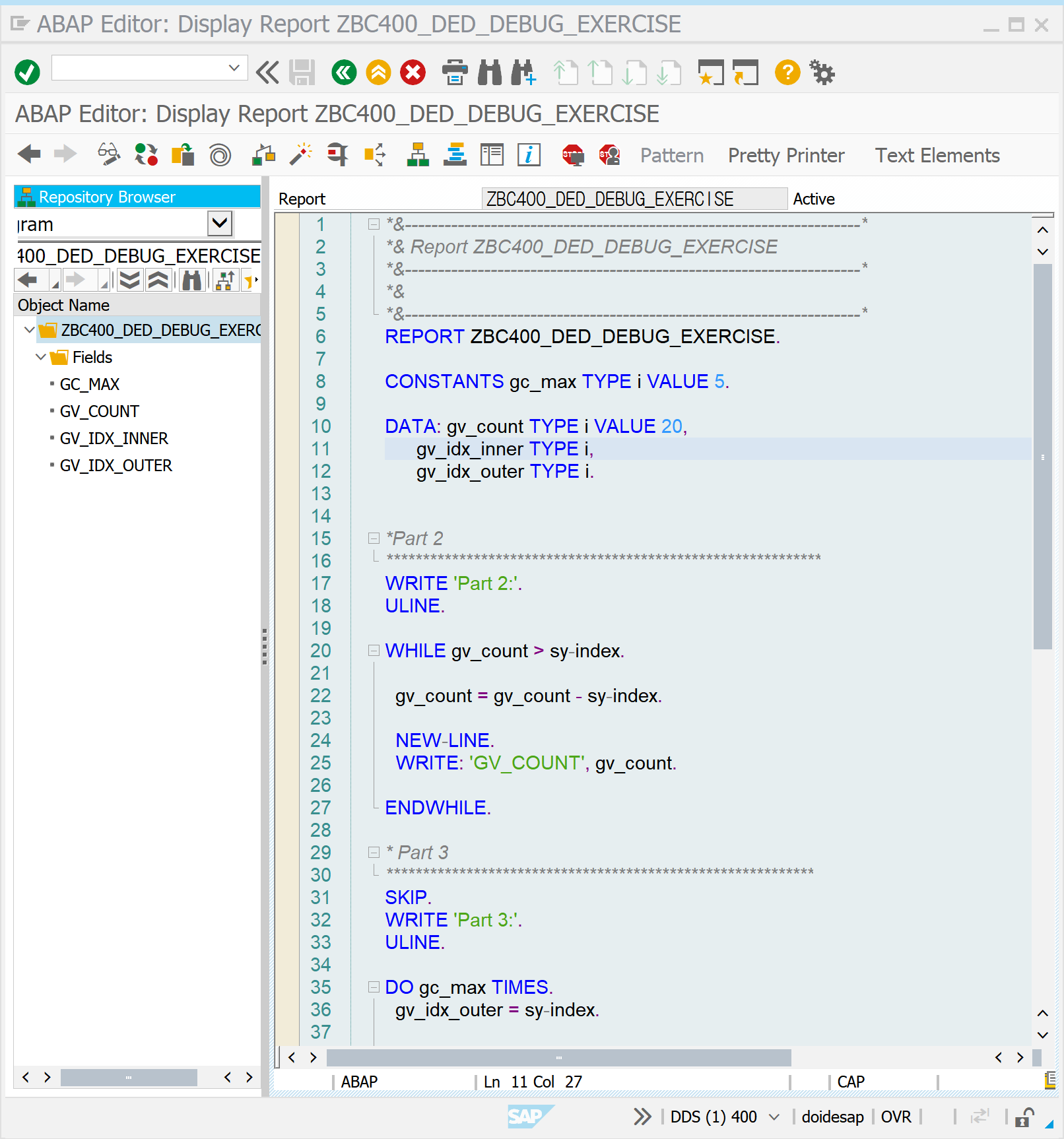
SE80 | 0509 | DEBUGGING에 대해 알아보자.

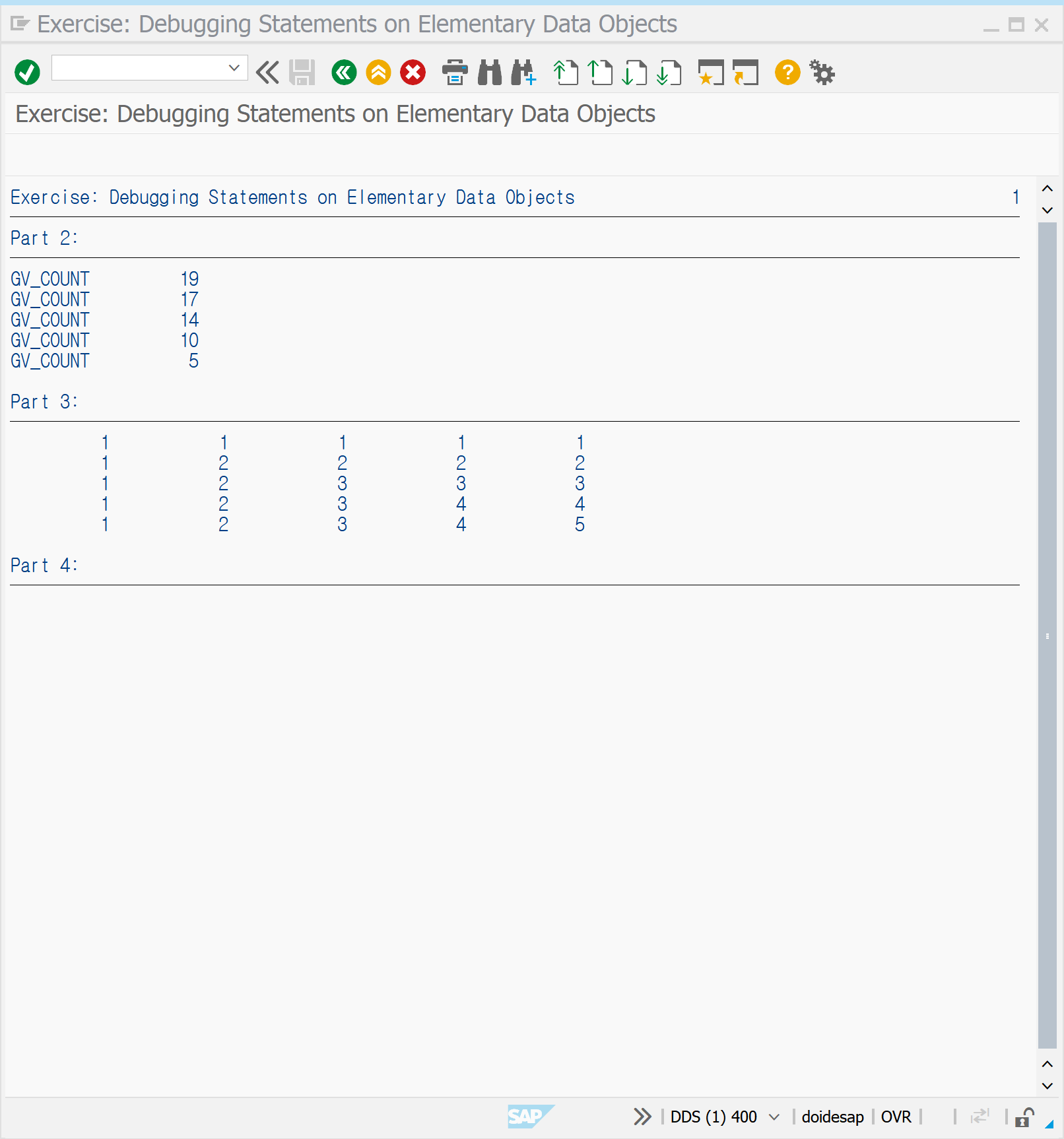
\*\*만약 DEBUGGING을 하고 싶을 때 실행 후 PARAMETER의 값을 넣은 다음 T-CODE창에  /H를 입력하면 된다.  
\*\*F5를 누르면 한 줄 한 줄 DEBUGGING이 실행된다.  
\*\*\*DEBUGGING MODE에서 값을 변경하여 값에 따른 결과값을 확인할 수 있다.  
\*\*WATCH POINT 반복문의 경우 코드 내에서 중단하여 결과값을 확인할 수 있다.  
\*\*\*\*ZBC400\_DED\_DEBUG\_EXERCISE PROGRAM을 검색해 확인해보자.  
  
PARAMETERS : P\_INT TYPE I,  
                            P\_INT2 TYPE I.  
  
DATA RESULT TYPE I.  
  
RESULT = P\_INT - P\_INT2.  
  
\*IF  
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.



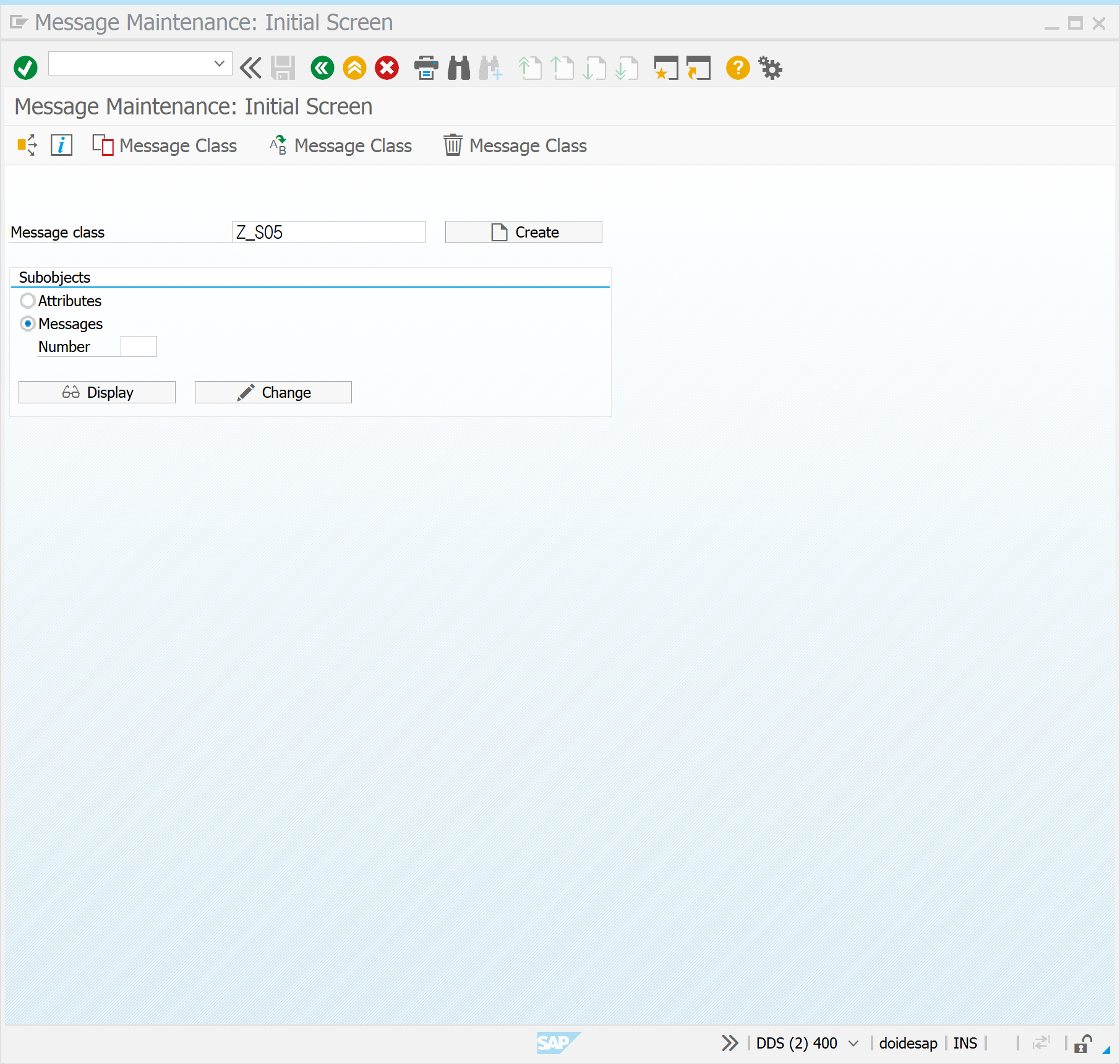
\*WATCHPOINTS | 관찰점은 필드 값이 변경되는 시점에서 프로그램이 정지되는 기능을 제공한다. 관찰점의 리스트를 조회할 수 있다.

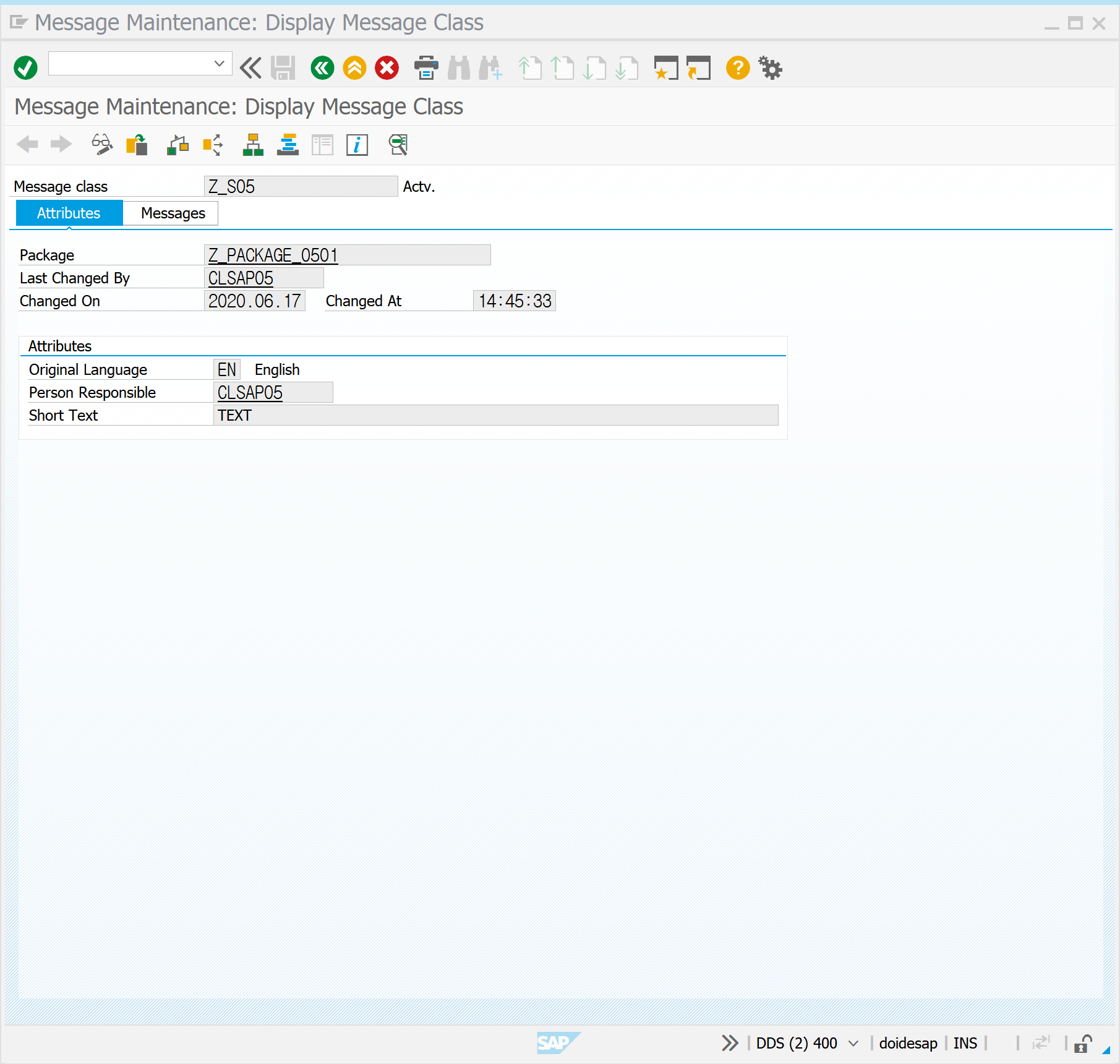
\*ZBC400\_DED\_DEBUG\_EXERCISE PROGRAM

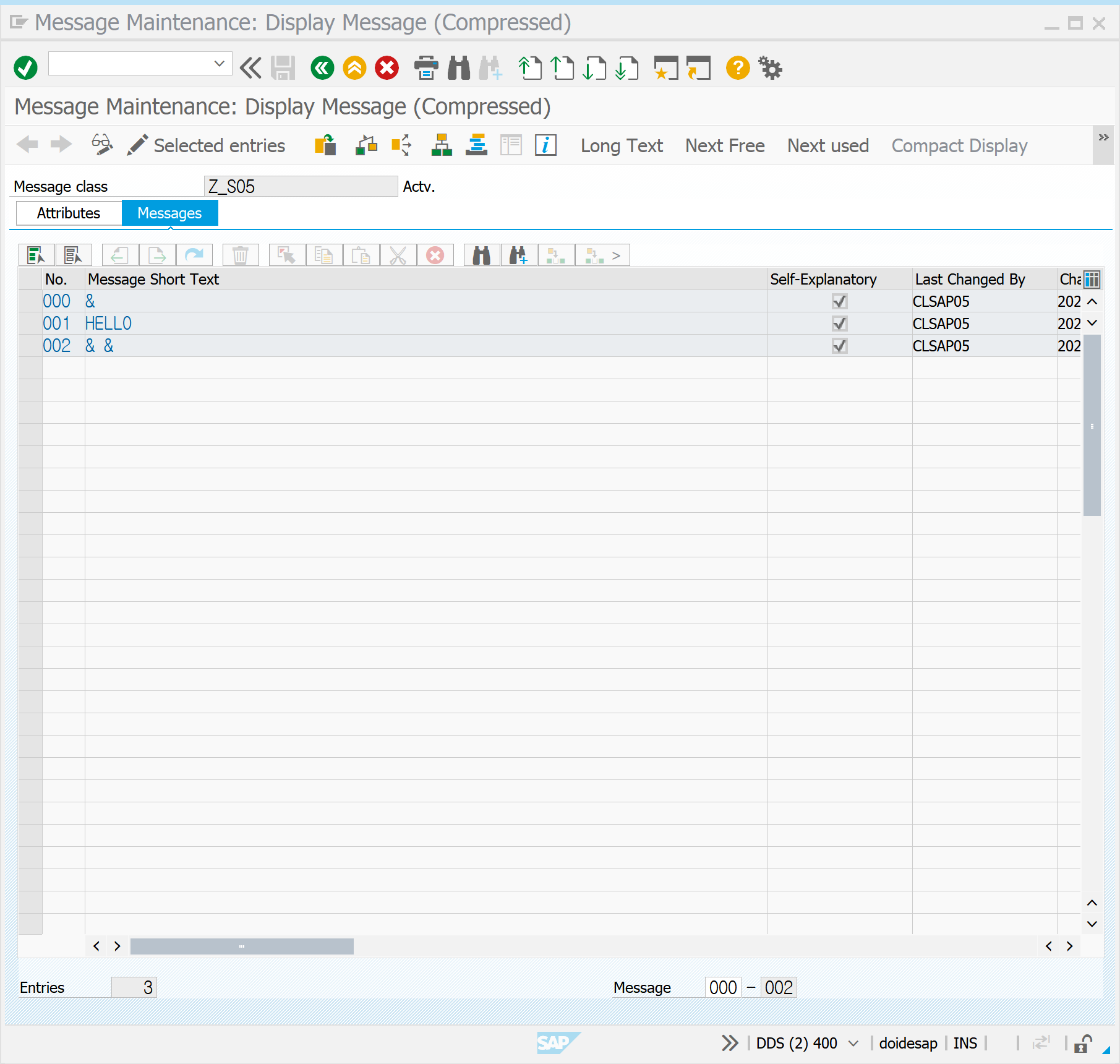
**



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>

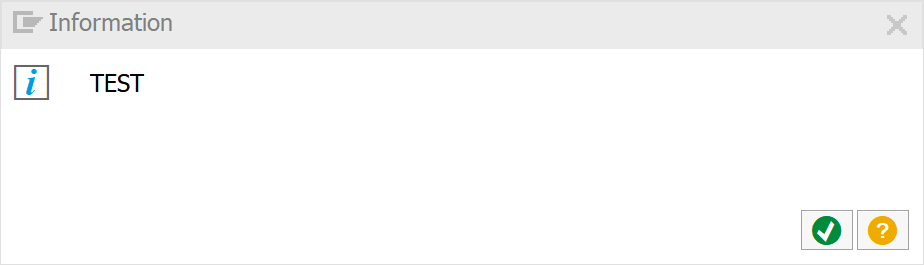
\*\*\*\*이때 S와 같은 MESSAGE TYPE은 다음과 같은 종류가 있다.

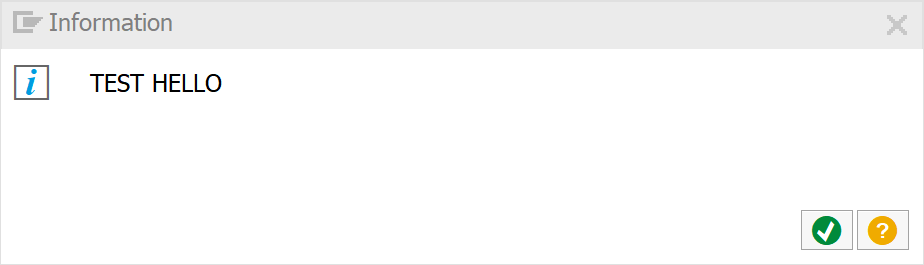
MESSAGE | 메시지 타입

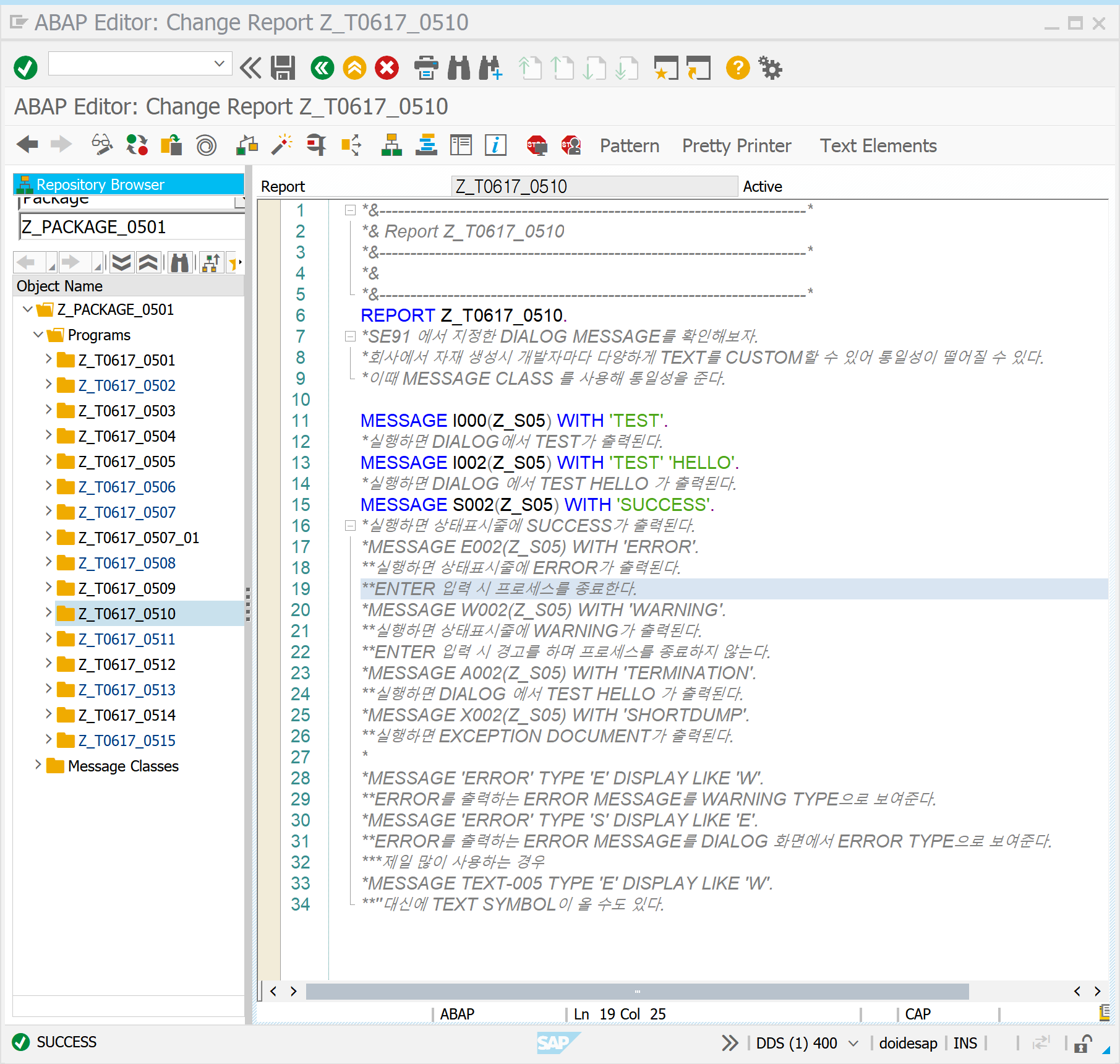
|  |  |
| --- | --- |
| E | 메시지 바 | 입력 값에 대한 체크 시 발생하는 에러 메시지 |
| W | 메시지 바 | ENTER를 누르면 다음 프로세스를 진행한다.  화면의 로직을 멈추고 경고 메시지 형태로 출력한다. |
| I | 팝업 윈도우 | ENTER를 누르면 다음 프로세스를 수행한다. |
| S | 성공 메시지 |
| A | 팝업 윈도우 | <STOP> BUTTON 이 윈도우 창 안에 있다.  <STOP> 을 누르면 프로그램 SESSION 이 종료된다. |
| X | SHORT DUMP라고 하며, DUMP화면과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

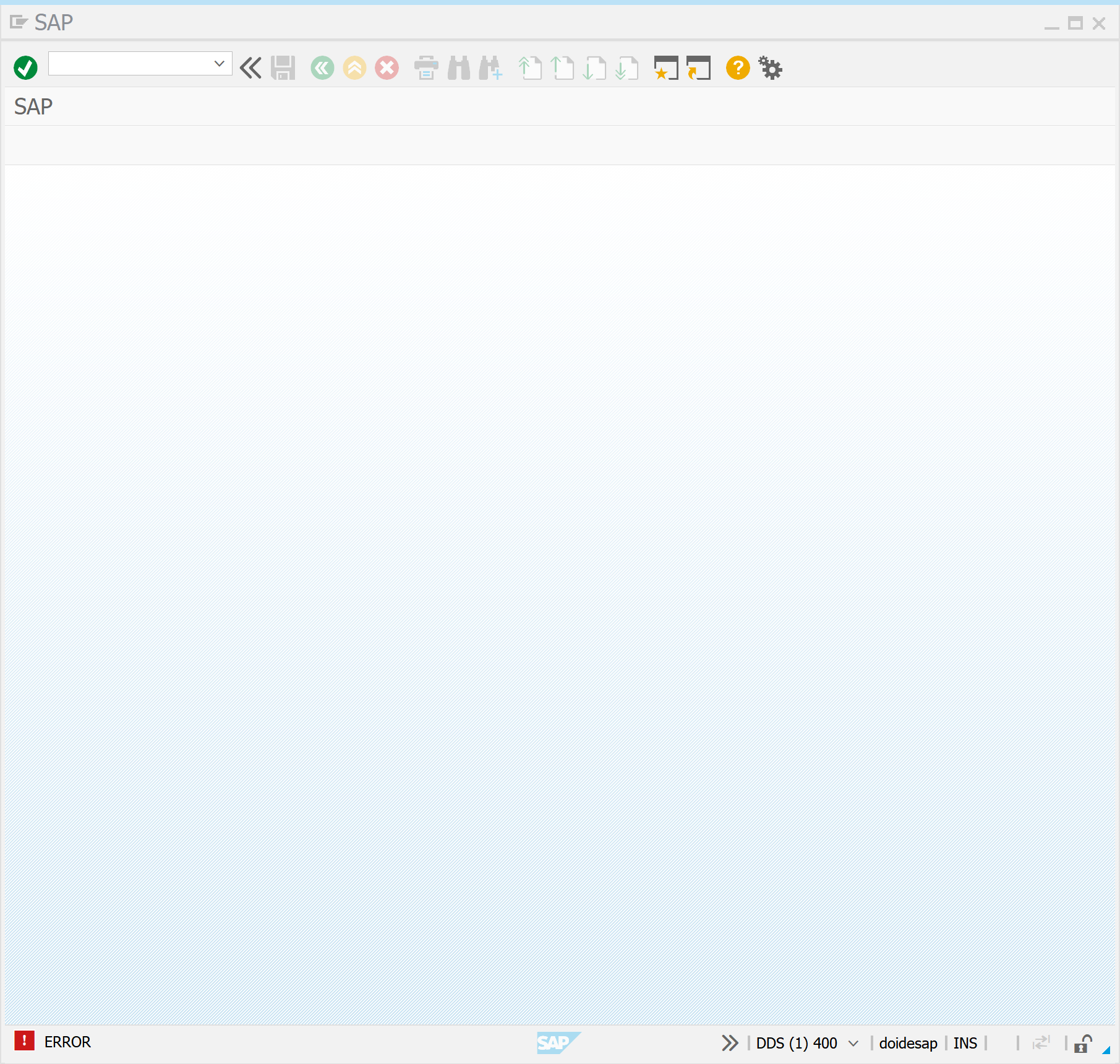
SE80 | 0510 | MESSAGE를 확인해보자.

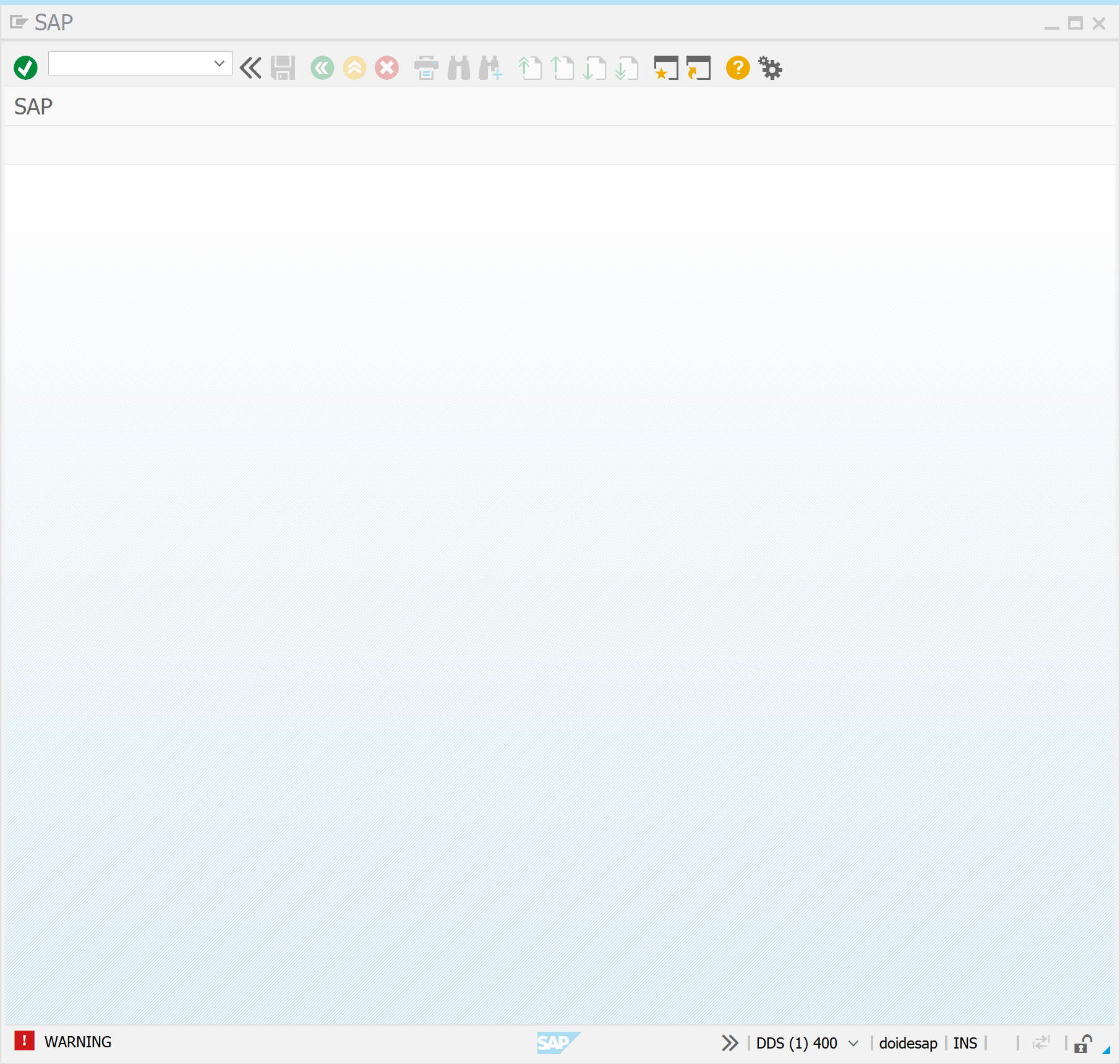
*\*SE91 에서 지정한 DIALOG MESSAGE를 확인해보자.*  
*\*회사에서 자재 생성시 개발자마다 다양하게 TEXT를 CUSTOM할 수 있어 통일성이 떨어질 수 있다.*  
*\*이때 MESSAGE CLASS 를 사용해 통일성을 준다.*  
  
MESSAGE I000(Z\_S05) WITH 'TEST'.  
*\*실행하면 DIALOG에서 TEST가 출력된다.*  
MESSAGE I002(Z\_S05) WITH 'TEST' 'HELLO'.  
*\*실행하면 DIALOG 에서 TEST HELLO 가 출력된다.*  
MESSAGE 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.*

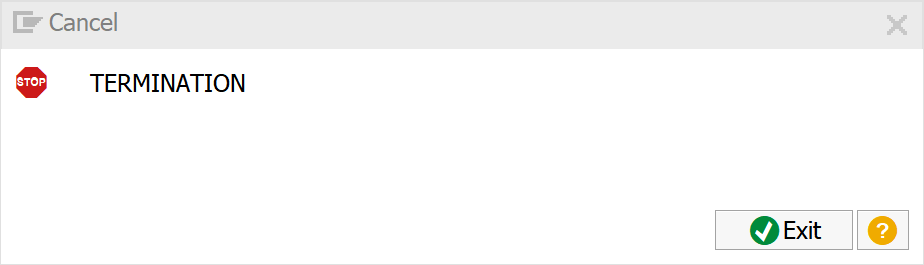


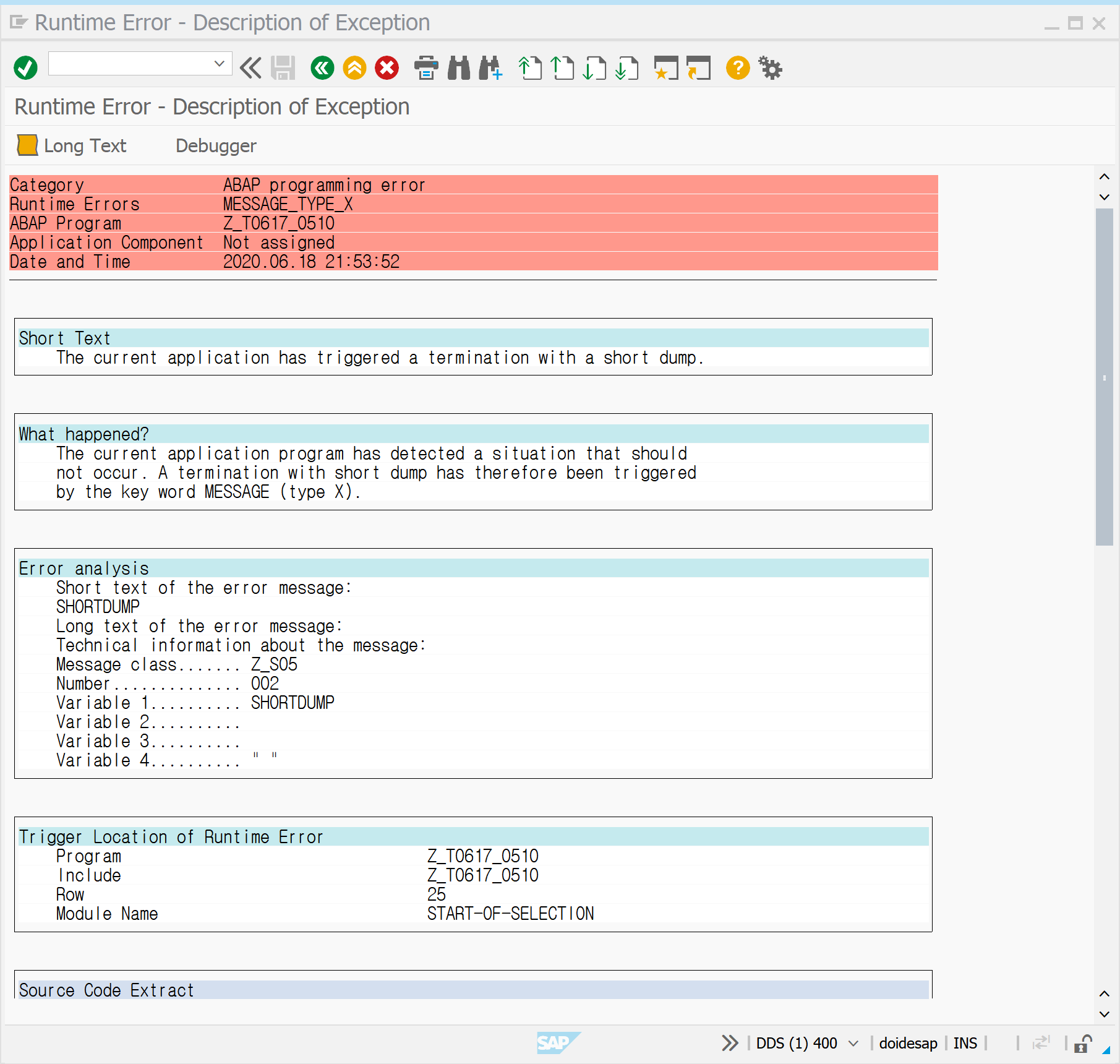


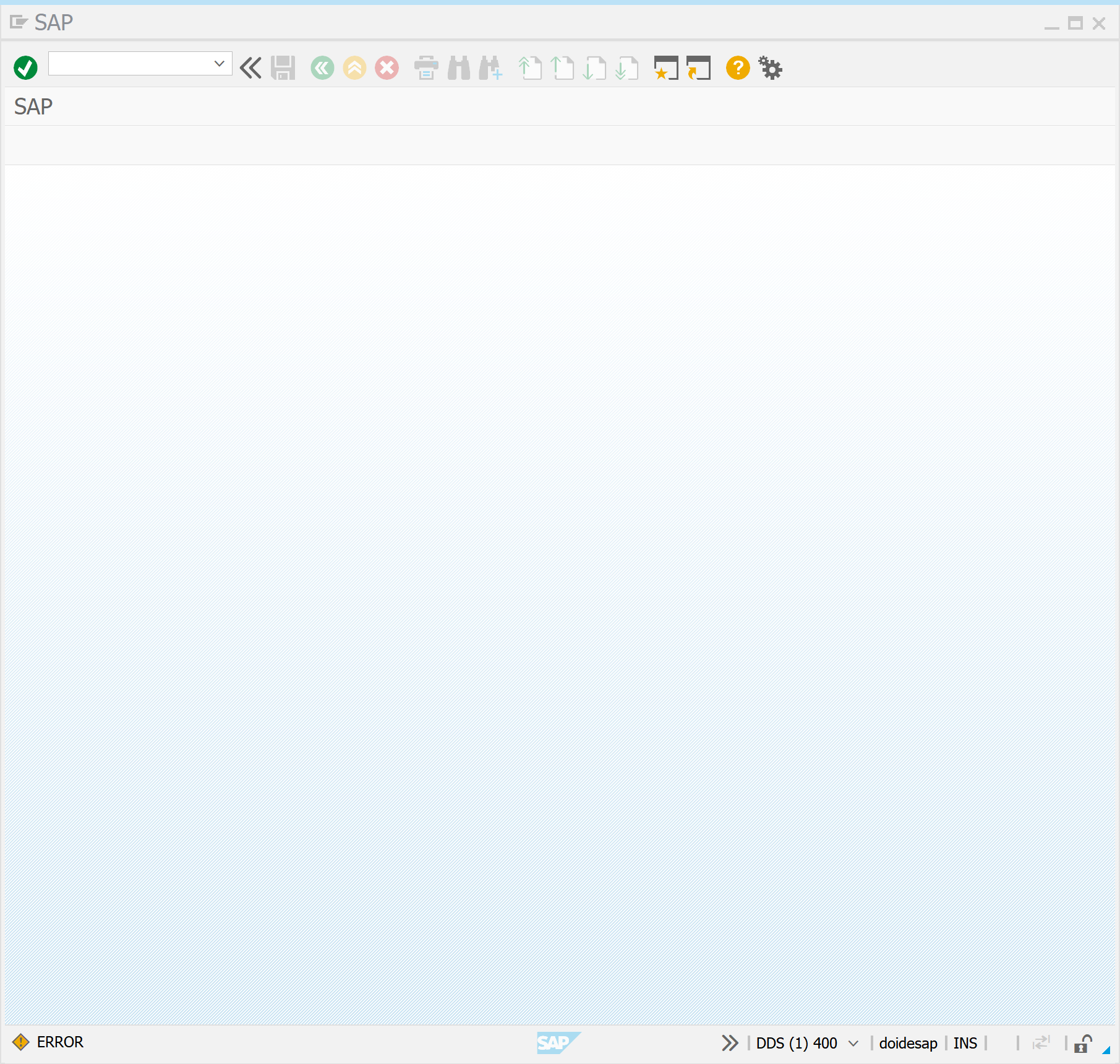


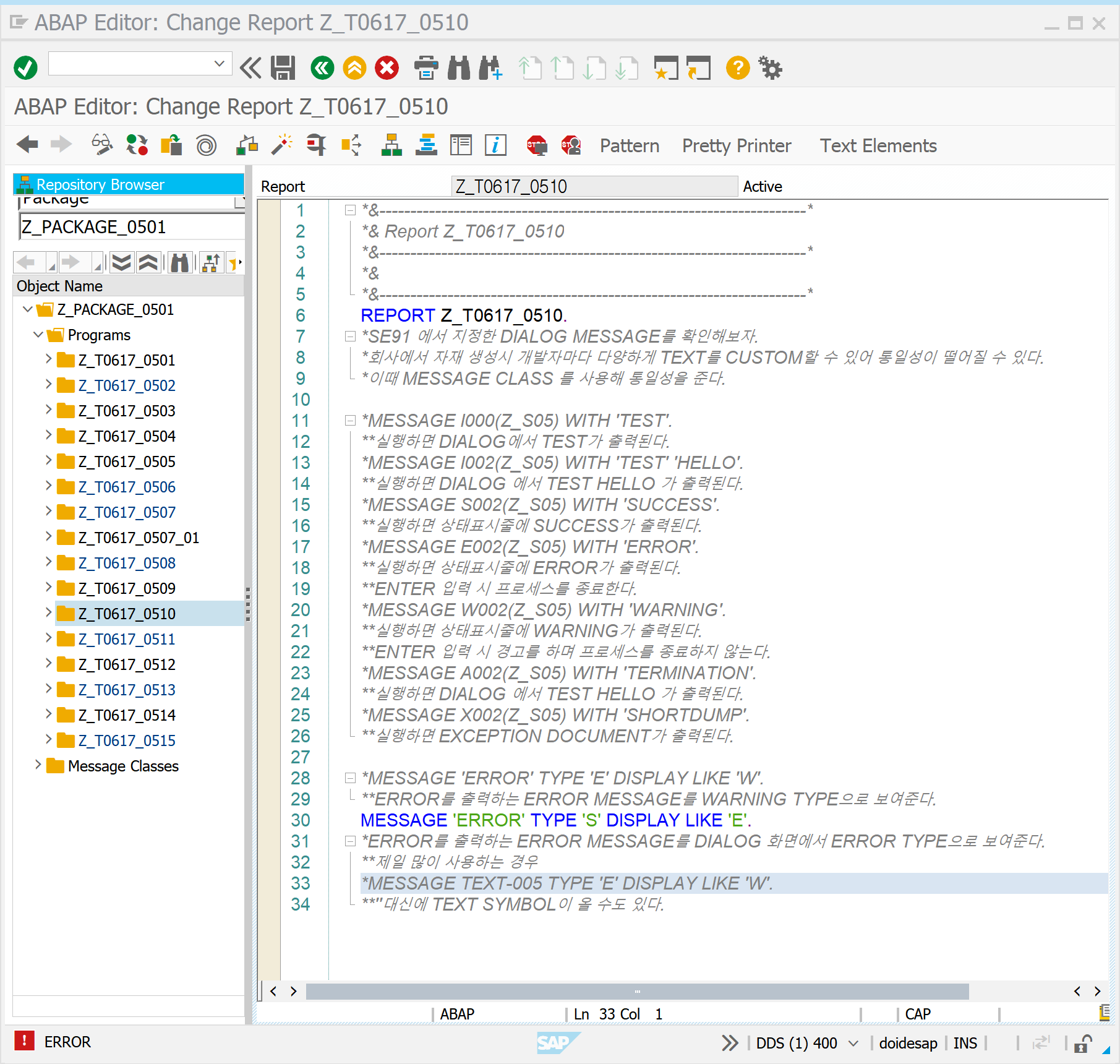


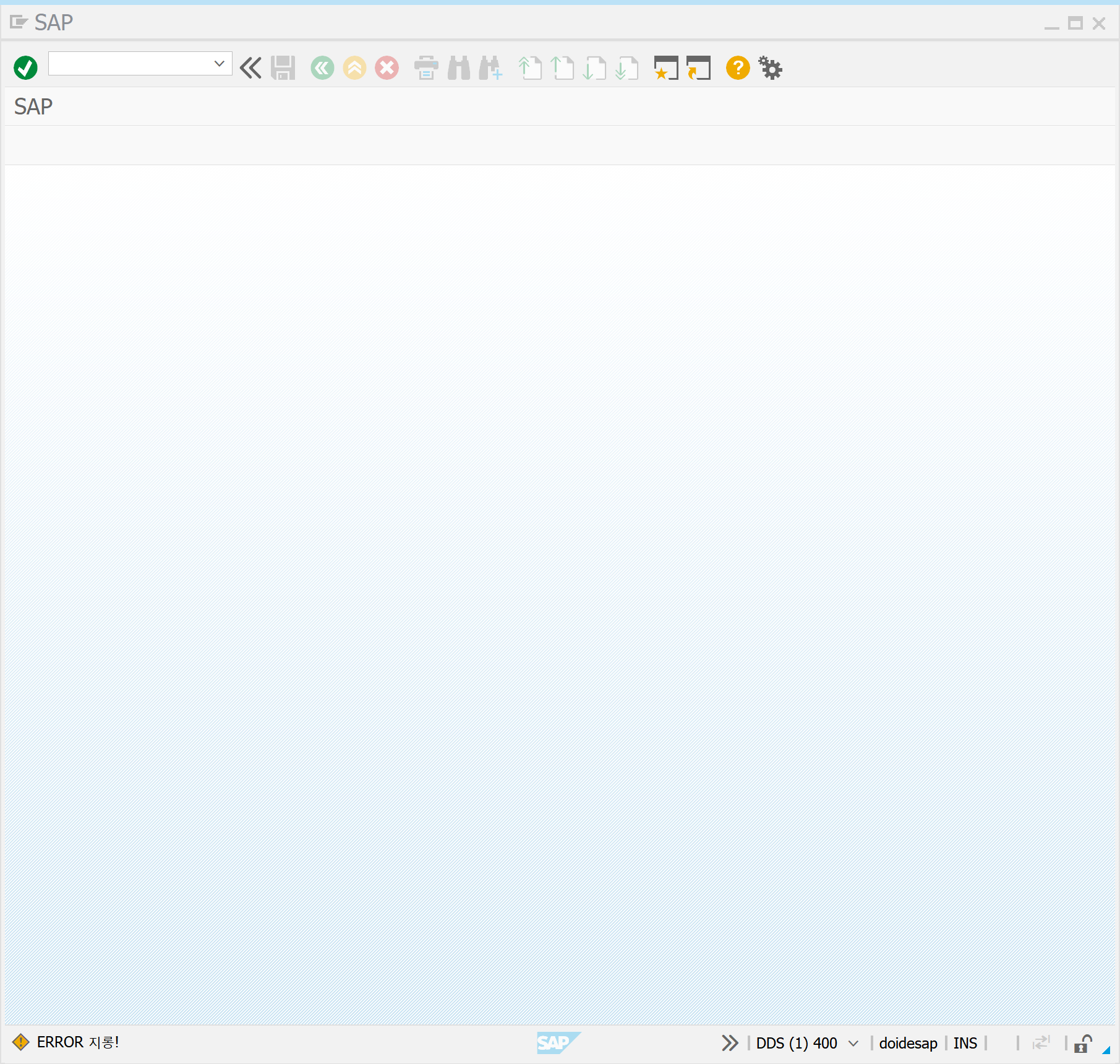












순환 반복 구문 |

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.

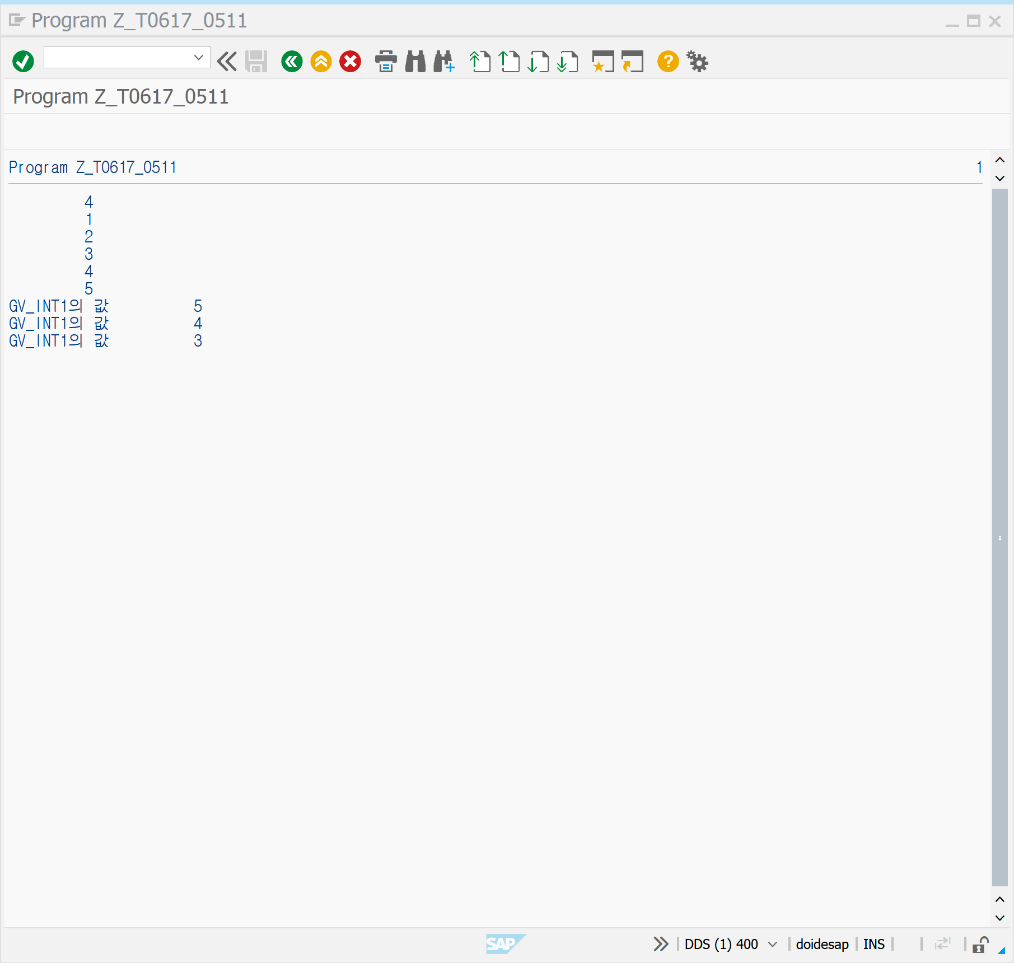
WHILE GV\_FLAG = ‘X’. ~~~ ENDWHILE.

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 알아보자

*\*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.

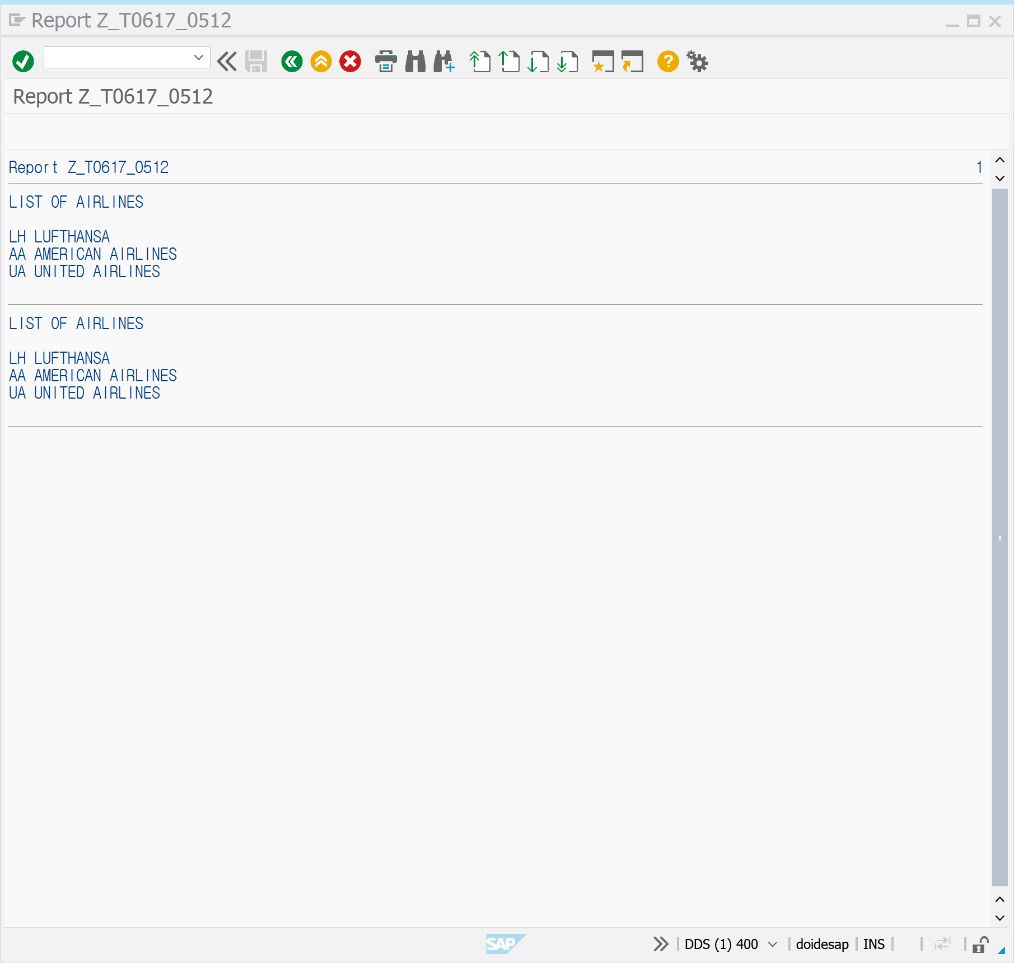


SUBROUTINE |

SUBROUTINE은 FORM으로 시작하여 END FORM으로 종료되는 구문을 의미하며, 스크립트의 모듈화, 재사용, 구조화를 주목적으로 한다. ABAP 프로그램에서는 PERFORM 구문을 이용한 SUBROUTINE으로 유사한 기능을 제공한다.이외에 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.

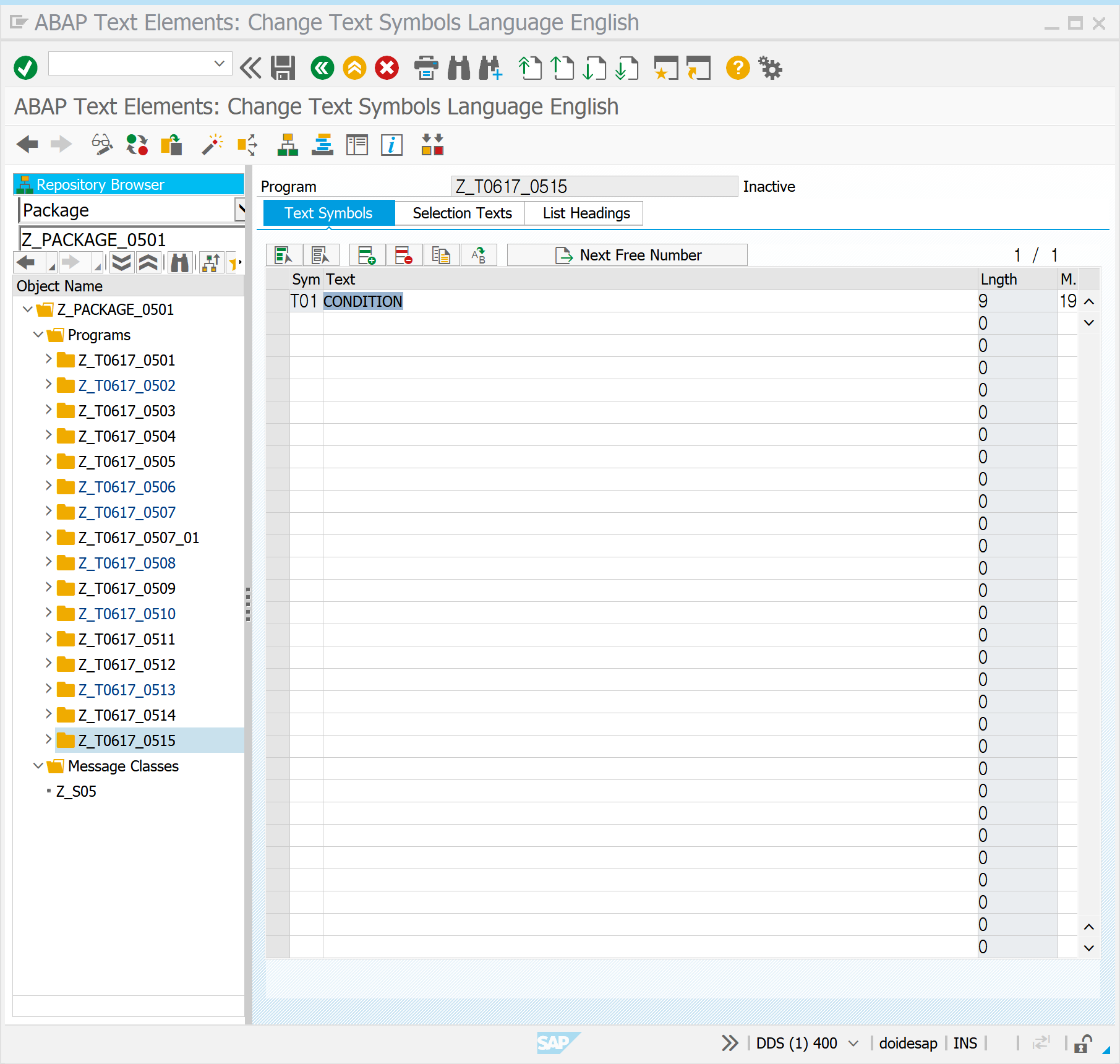
WITH FRAME | 프레임을 추가한다.

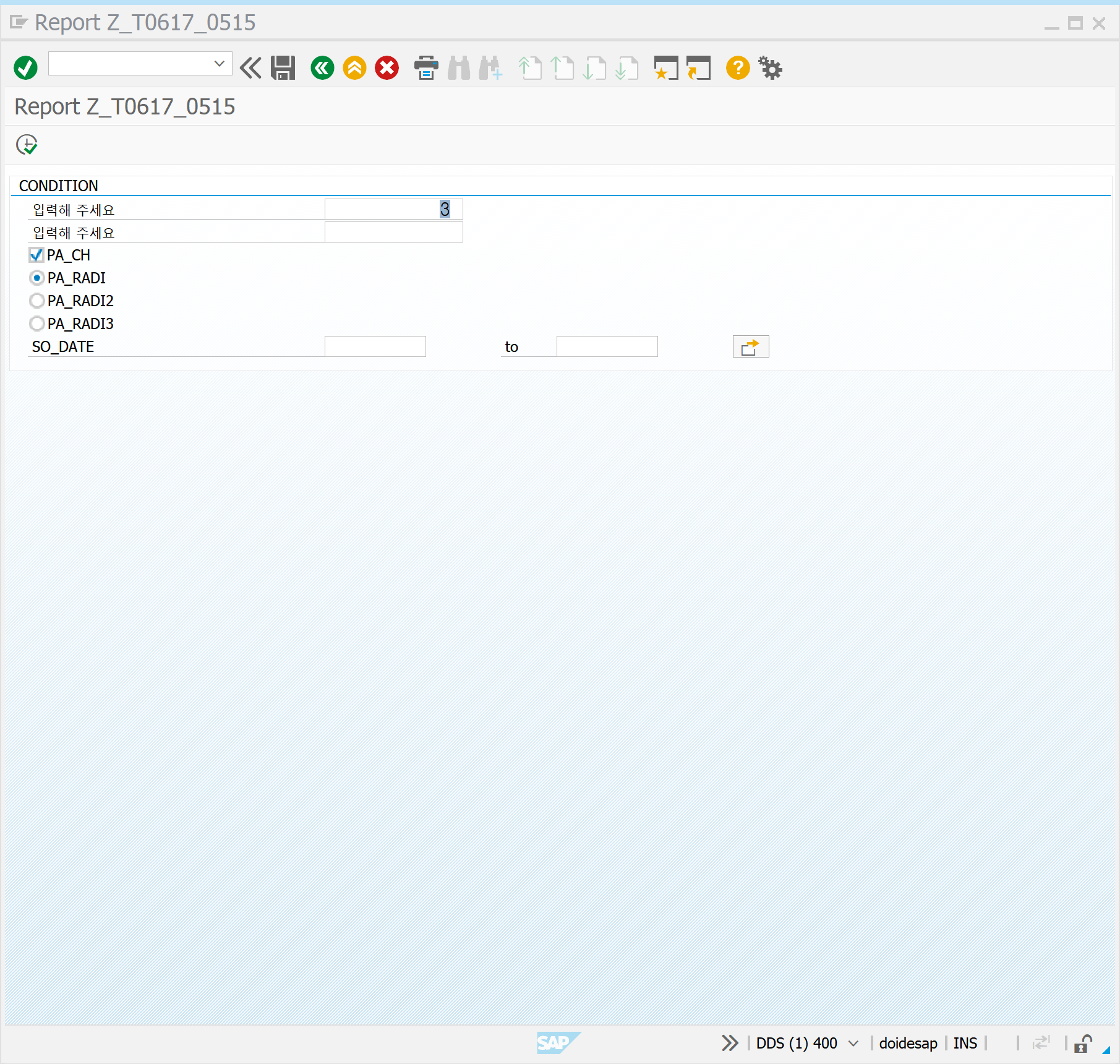
TITLE T1 | 프레임의 TITLE을 추가한다.

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.*  
*\*RADIOBUTTON GROUP은 GROUP 명으로 묶어 생성한다.*  
*\*만약 RADIO BUTTON 에 CHECK를 하고 싶다면 DEFAULT 'X'를 사용한다.*  
*\*RADIO GROUP에서 반드시 하나의 값은 CHECK를 해야 한다.*  
  
SELECTION-SCREEN BEGIN OF BLOCK B1 WITH FRAME TITLE TEXT-T01.  
PARAMETERS: PA\_INT1 TYPE I OBLIGATORY DEFAULT 3,  
                     PA\_INT2 TYPE I,  
                     PA\_CH AS CHECKBOX DEFAULT 'X',  
                     PA\_RADI RADIOBUTTON GROUP RAD1 DEFAULT 'X',  
                     PA\_RADI2 RADIOBUTTON GROUP RAD1,  
                     PA\_RADI3 RADIOBUTTON GROUP RAD1.  
  
SELECT-OPTIONS: SO\_DATE FOR SY-DATUM.  
SELECTION-SCREEN END OF BLOCK B1.





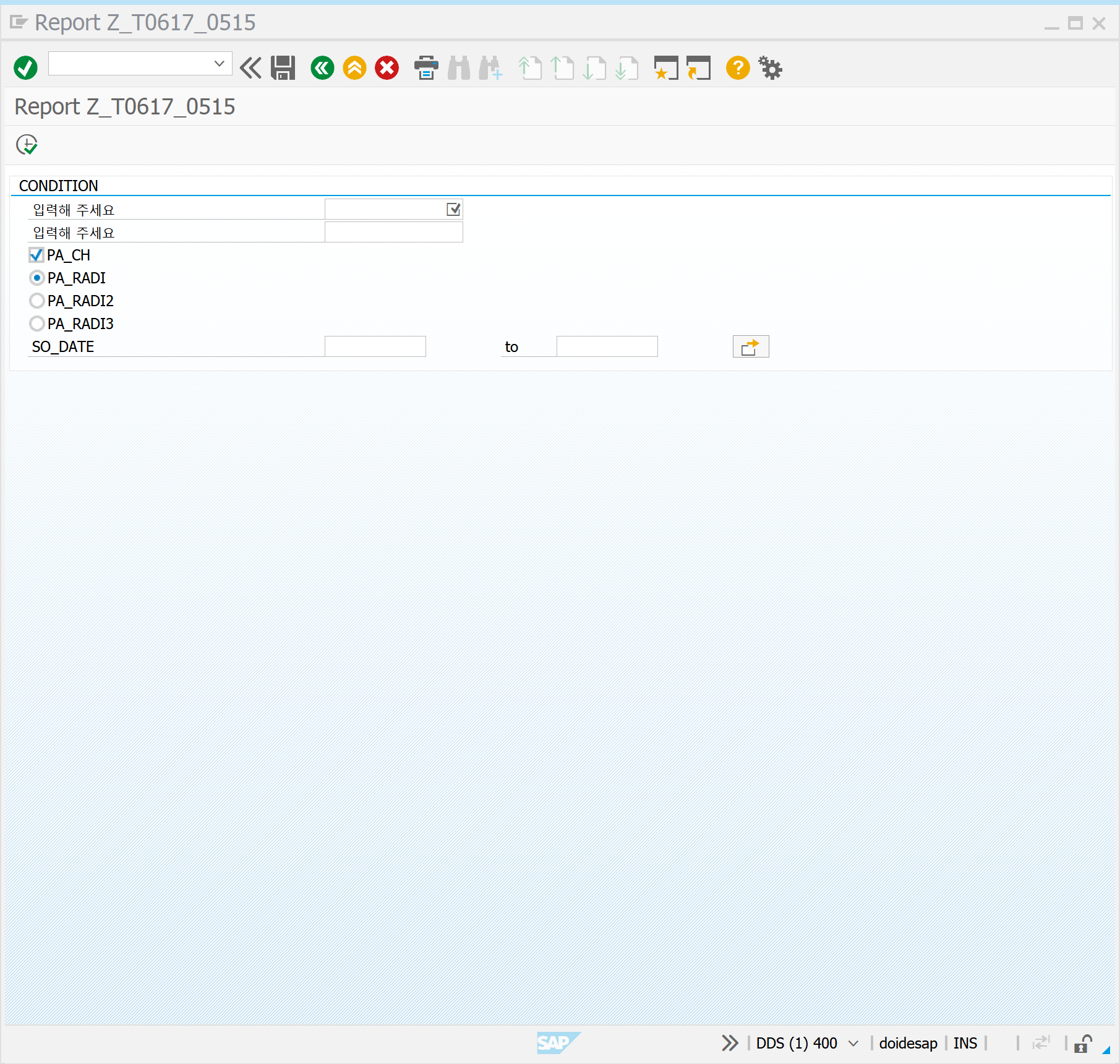
SELECT-OPTION | MULTI SELECTION

RADIO BUTTON GROUP

TTITLE TEXT

TEXT ELEMENT

CHECK BOX



OBLIGATORY

