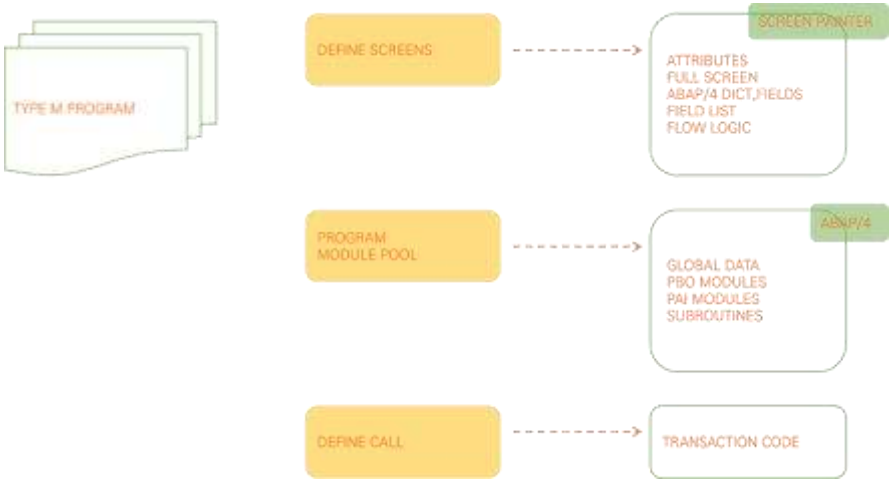
0707 | 0708 | 0709 | 0710 | ABAP

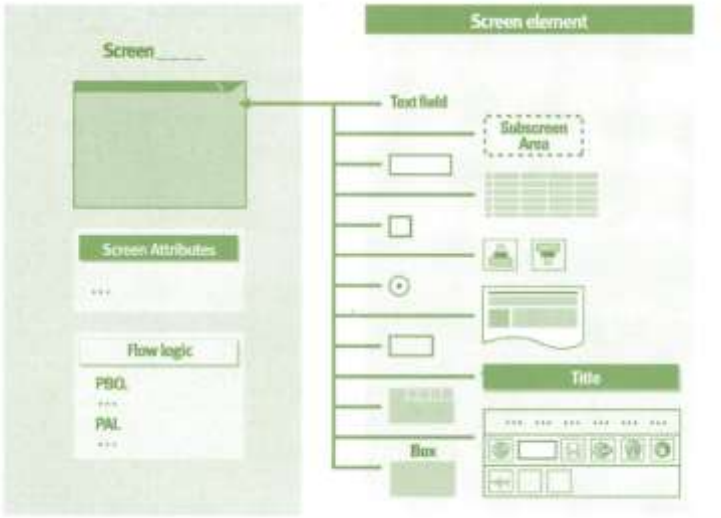
TYPE-M PROGRAM | 스크린 정의 | 스크립트 추가 | 트랜잭션 코드 생성



MODULE POOL PROGRAM | REPORT PROGRAM 과의 차이

REPORT PROGRAM은 프로그램이 자동으로 생성해주는 1000번 SELECTION SCREEN을 사용하며 MODUEL POOL PROGRAM | 온라인 프로그램은 개발자가 직접 생성한 일반 스크린을 사용한다. REPORT PROGRAM은 DATABASE TABLE에서 조회한 데이터를 화면에 송출하는 것이 주 목적이며 MODULE POOL PROGRAM은 데이터를 조회 수정 삭제 생성하는 등 데이터 관리를 위한 것을 주 목적으로 한다. 데이터 관리는 BUSINESS FLOW에 의해 파생되는 데이터 처리도 가능하다는 것을 의미한다. 그러나 실무에서는 TYPE 1 PROGRAM | EXECUTABLE PROGRAMS DML SELECT-OPTION과 같은 기능을 활용하기 위해 이를 크게 구분하지 않는다.

SCREEN | SAPGUI 에 조회되는 모든 화면



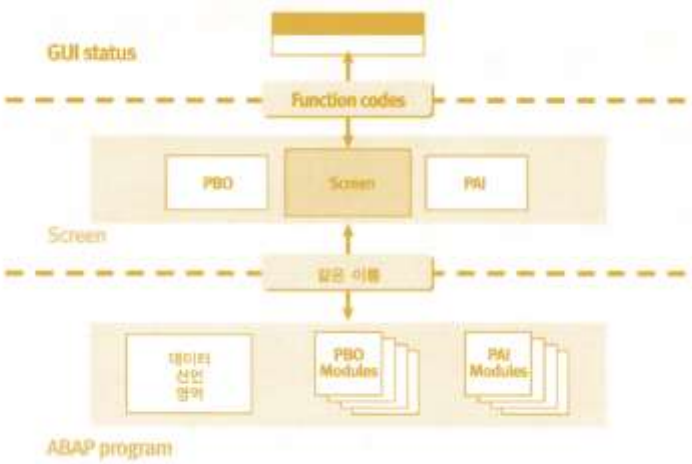
SCREEN 은 사용자와 상호 작용을 통해 데이터를 생성하고 조회하는 작업 영역으로 정의된다. TYPE-M 프로그램 에서만 사용하는 것이 아닌 TYPE-1, TYPE-F에서도 사용되며 INPUT OUTPUT 필드와 FLOW LOGIC 으로 구성된다.

FLOW LOGIC |

PBO	PROCESS BEFORE OUTPUT
PAI	PROCESS AFTER INPUT

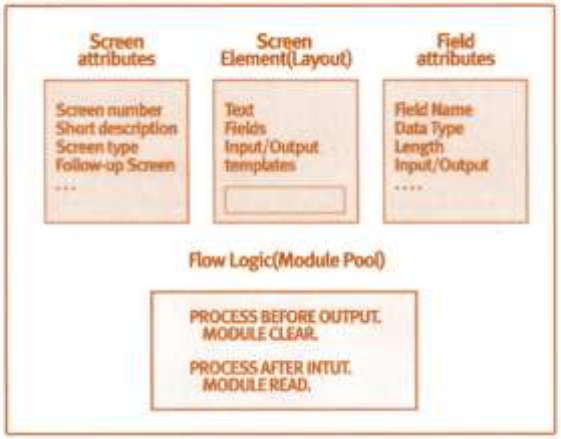
SCREEN FLOW LOGIC은 PBO와 PAI 이벤트로 나누어지며 PBO는 스크린이 화면에 보여지기 전 실행되며, PAI는 스크린 상에서 USER ACTION이 발생된 후 실행되는 이벤트이다.

SCREEN 위치 | SAPGUI 에 조회되는 모든 화면



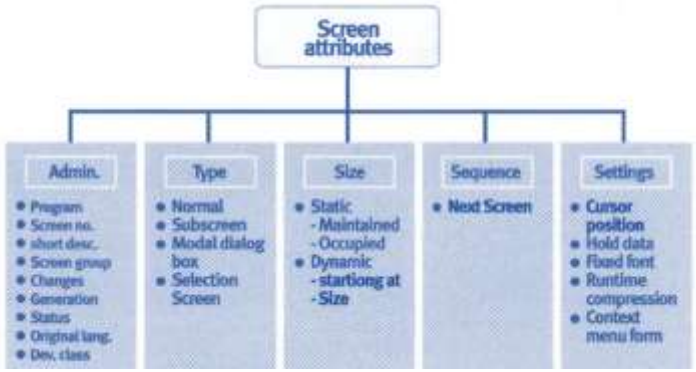
위의 그림은 GUI STATUS와 ABAP PROGRAM 사이에서 SCREEN의 위치를 보여준다. SCREEN은 동적인 프로그램으로 자신의 데이터 오브젝트인 스크린 필드를 가질 수 있다. 스크린 필드는 스크린 상에서 INPUT OUT 필드와 연결되어 있다. 스크린이 조회되거나 사용자의 액션이 끝난 후 자동으로 같은 이름을 가진 스크린 필드들과 ABAP 프로그램의 데이터 오브젝트들 사이에서는 복사가 일어난다. 이 과정은 PAI 이벤트 호출 시 프로그램에서 내부적으로 수행된다. 스크린에는 GUI STATUS 가 존재하며, 이는 MENU BAR, STANDARD TOOLBAR, APPLICATION TOOLBAR을 포함하고 있다. GUI STATUS는 MENU PAINTER를 사용하여 생성하며 스크린에 ABAP 프로그램 내 SET PF-STATUS 구문을 이용해 동적으로 할당할 수 있다.

SCREEN 구성 | 속성 요소 필드 흐름 로직

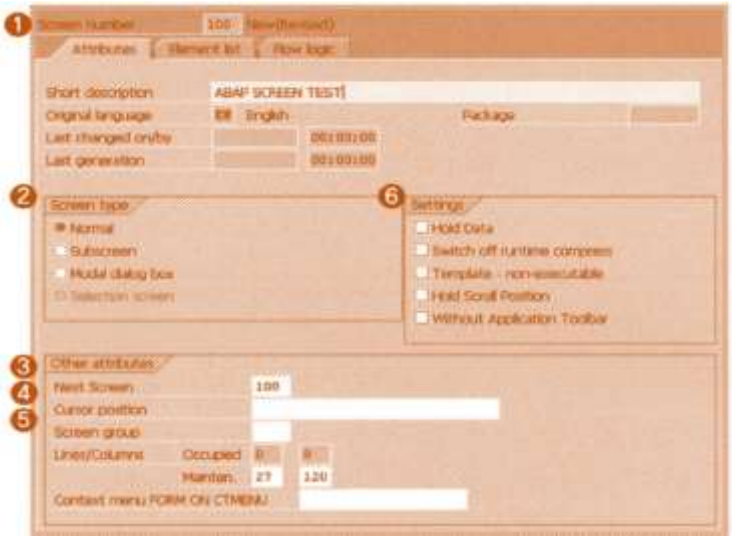


스크린 속성	스크린 번호, 타입, 이름, 내역, 창 크기, 다음 화면을 정의하고 SAP 시스템에 스크린 오브젝트를 연결
스크린 요소	사용자가 데이터를 조회하고 입력하는 GUI 화면 디자인 텍스트 필드, INPUT OUTPUT 필드, 체크박스, 라디오 버튼 과 같은 스크린 구성 요소 정의
스크린 필드	메인 스크린 필드의 데이터 타입과 길이 정의
스크린 흐름 로직	사용자 액션에 반응하는 PAI PBO 와 관련되어 절차적으로 수행되어야 할 부분 정의

SCREEN 속성 | 스크린 필드들로 구성되며 스크린 페인터에서 설정



스크린 필드는 스크린이 메모리로 로딩되어 활성화되었을 때 스크린 페인터에서 정의한 필드명을 그대로 사용한다. 이는 조회 또는 사용자의 액션 후 자동으로 같은 이름을 가진 스크린 필드들과 ABAP PROGRAM의 데이터 오브젝트들 간의 데이터 전달됨으로 스크린 상에서 INPUT OUTPUT 필드들과 연결을 의미한다.



스크린은 1000번에서 1010 사이는 표준 SELECTION SCREEN 과 ABAP DICTIONARY MAINTANANCE 스크린으로 예약되어 있다. SAP CUSTOMER은 9000번 이상을 사용하도록 권장하나 실제 업무에서는 100번 단위로 구분하여 사용한다.

SCREEN 속성 | 스크린 필드들로 구성되며 스크린 페인터에서 설정

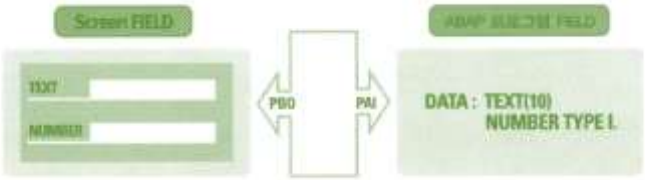
SCREEN NUMBER	프로그램 내부에서 스크린을 구별하는 4자리 숫자. 표준 스크린을 가지고 있을 경우 같은 NAME SPACE를 가진 1000 스크린을 사용할 수 X , 연속적 사용	
SCREEN TYPE	NORMAL	스크린이 전체 GUI 점유, 일반적 사용
	SUBSCREEN	SUBSCREEN 영역 안에서 사용
	MODAL DIALOG BOX	팝업창의 형태로 GUI 창 일부 사용 팝업을 종료 X 다른 스크린 선택 X
NEXT SCREEN	호출한 스크린의 PAI 실행 후 다음에 실행되는 스크린 번호를 지정할 수 O , 만약 호출 프로그램이 없으면 트랜잭션은 종료되며 동적으로 프로그램 내부에서 SET 스크린 구문을 이용해 일시적으로 변경 가능 NEXT SCREEN = 0 또는 SPACE 설정 시 프로그램이 종료하면 이전의 프로그램으로 복귀	
CURSOR POSITION	스크린이 DISPLAY 될 때 커서의 위치 지정 기본 값은 스크린 요소의 첫 번째 필드로 동적 설정 O	
SCREEN GROUP	스크린 실행 동안 시스템 변수 SY-DYNGR에 저장 여러 개의 스크린을 화면 그룹 하나로 지정하여 속성 변경 시 유용하게 사용되며 TFAWT 내에서 정의됨	
HOLD DATA	사용자 프로파일 내 저장된 DATA를 스크린의 기본 값으로 설정하기 위해 사용, 세션 종료 시까지 유효 이는 ABAP MEMORY 영역 데이터 사용 X 의미하며 스크린 속성에 HOLD DATA 설정 시 SYSTEM	USER PROFILE	HOLD DATA 기능 이용	

SCREEN 구성 요소 | DATA를 보여주며 사용자와 대화 창구 기능

스크린의 구성 요소들은 데이터를 보여줄 뿐만 아니라 사용자들과 대화할 수 있는 창구 기능 | USER DIALOG 을 한다. 화면 레이아웃은 TOOLBAR 기능을 이용하여 자유롭고 편리하게 디자인할 수 있는 GUI 환경을 제공한다. 모든 스크린 구성 요소들은 기본 속성으로 자동 설정되며 기본 세팅 이외의 사항은 스크린 페인터를 통해 변경 할 수 있다. 스크린 페인터는 스크린 요소들의 레이아웃 정렬과 같은 일을 수행하며 스크린 페인터에서 속성을 정의하더라도 실행시점에 ABAP 프로그램으로 정의된 속성으로 변경된다.

TEXT FIELDS AND FRAMES	조회 목적으로 사용되는 요소로서 사용자가 변경할 수 X 주로 INPUT 필드의 내역으로 사용
INPUT OUPUT FIELD	ABAP 프로그램에서 데이터를 보여주거나 사용자에게 입력 받으려고 사용되는 요소로 스크린 필드와 연결
PUSHBUTTONS	스크린 FLOW LOGIC의 PAI EVENT 발생 PUSH BUTTON에 지정된 FUNCTION CODE 전달
CHECKBOXES	CHECK BOX 형태로 사용자가 선택 및 해제할 수 있는 특별한 형태의 INPUT/OUTPUT 필드로 'X' 또는 ' ' 값
RADIO BUTTONS	RADIO BUTTON의 형태 요소로 GROUPING 된 RADIO BUTTON 중 하나 선택
SUBSCREEN	스크린 내에서 다른 SUBSCREEN 삽입위한 AREA
TABSTRIPS	여러 개의 SUBSCREEN 중 사용자가 원하는 SUBSCREEN을 선택하기 위한 TAB ELEMENT
CUSTOM CONTROLS	GUI CONTROL을 보여주기 위해 사용되는 AREA ALV CLASS에 사용되는 것이 대표적 CONTROL FRAMEWORK 의 수단으로 ABAP 프로그램과 연결된 SAP GUI SOFTWARE COMPONENT 영역에 독립적인 GUI CONTROL
TAB CONTROLS	TABLE 형태로 INPUT OUTPUT 필드를 받는 요소

SCREEN 필드 | 스크린의 작업 영역 메모리에 존재하는 필드



스크린의 구성 요소인 필드는 실행 시점에 어떠한 값을 가지게 될지 모르는 정적인 상태로, 사용자가 입력한 값을 받아 메모리에 저장하고 스크린 필드에 값을 보이게 하는 역할을 한다. 즉, 스크린 필드라는 개념이 존재해 실행 시점의 입력 값들에 대해 프로그래밍 할 수 있다. 이것을 DYNPROS | DYNAMIC PROGRAM 이라 한다. 스크린 필드는 스크린 작업 영역 | 메모리에 존재하는 필드로서 PAI 이벤트가 발생하기 전 시점에 ABAP 프로그램에 있는 같은 이름을 가진 필드로 값이 복사된다. 또한, PBO 이벤트가 종료되는 시점에서 다시 ABAP 프로그램에 있는 같은 이름을 가진 필드로부터 값을 복사해 온다. 값 전달을 위해 스크린 필드는 유일한 변수명을 가지며, ABAP 프로그램 내 같은 필드명으로 선언할 필요가 있다. 이러한 환경에서 PBO/PAI를 통해 스크린 필드와 ABAP 프로그램 간에 데이터가 전달된다. 스크린 필드 영역에서 필드의 속성은 ABAP 프로그램 영역 또는 ABAP DICTIONARY에서 선언한 데이터 타입과 길이를 참고하며 스크린 페인터에서 생성한 스크린 필드들의 이름은 스크린 필드와 같은 이름으로 연결되어 있다. 화면에서 사용자가 버튼을 클릭하는 등의 이벤트를 수행하게 되면 OK_CODE에 값이 저장된다. 하나의 MODULE POOL PROGRAM에는 여러 개의 스크린이 존재할 수 있으며 각각의 스크린에 OK_CODE를 지정해야 한다.

SCREEN FLOW LOGIC | 스크린이 절차적으로 실행되어야 할 부분

스크린 FLOW LOGIC은 ABAP EDITOR와 유사한 FLOW LOGIC 에디터에서 기술할 수 있다. 스크린 FLOW LOGIC에서 사용하는 문법은 ABAP와 유사하지만 다른 언어로 스크린 LANGUAGE 라고 부르기도 한다. 가장 큰 차이점은 명시적일 데이터 선언 부분이 존재하지 않는 것인데, 이는 존재하지 않는 것이 아니라 스크린 요소를 구성하며 정의하는 것을 의미한다. 그러나 PROCESSING BLOK을 가지고 있다는 점에서는 ABAP과 유사하다. PROCESS BEFORE OUTPUT, PROCESS AFTER INPUT은 필수 사항이다.

스크린 요소 | PROCESSING BLOCK |

각 프로세싱 블록은 MODULE 이라는 기능으로 이루어져 있으며, 이러한 측면에서 TYPE-M 프로그램을 MODULE POOL PROGRAM 이라고 한다.

PROCESS BEFORE OUTPUT	스크린의 PAI 이벤트가 실행되고 현재 스크린이 조회되기 전에 자동으로 실행. PBO 이벤트가 실행되고 나서 스크린이 조회되며, 화면 처음 실행 시 PBO만 수행하며 화면에서 사용자가 이벤트를 발생시키면 PAI가 수행되며 PBO가 실행. PBO는 일반적으로 화면의 초기값을 지정하는 데 자주 사용.
PROCESS AFTER INPUT	사용자가 버튼을 클릭하는 것과 같은 액션을 수행했을 때 발생하는 이벤트 블록으로 PAI 이벤트가 실행된 후 다음 스크린의 PBO 이벤트를 호출.
PROCESS ON HELP-REQUEST	사용자가 F1 키를 눌렀을 때 발생하는 이벤트 블록.
PROCESS ON VALUE-REQUEST	사용자가 F4 키를 눌렀을 때 발생하는 이벤트 블록

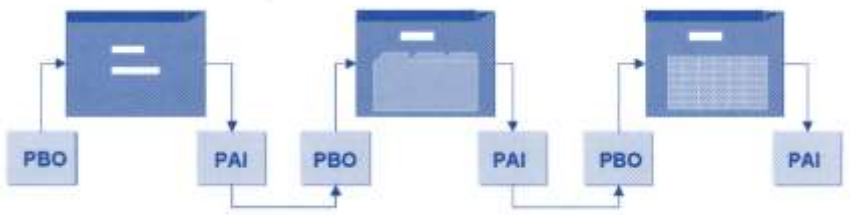
FLOW LOGIC KEYWORD |

각 프로세싱 블록은 MODULE 이라는 기능으로 이루어져 있으며, 이러한 측면에서 TYPE-M 프로그램을 MODULE POOL PROGRAM 이라고 한다.

CALL	SUBSCREEN을 호출
MODULE	프로세싱 MODULE 정의, DIALOG MODULE 호출
FIELD	스크린 필드에서 ABAP 필드로 DATA를 복사하는 구문. 스크린 필드들은 PAI 이벤트 수행 전 ABAP 프로그램에서 CONTROL 할 수 없다. 이는 아직 스크린 필드의 데이터가 ABAP 프로그램의 데이터로 복사되지 않았기 때문이다. FIELD 구문을 선언하게 되면 PAI를 수행하지 않아도 ABAP 프로그램으로 데이터를 체크할 수 있으며 특정 필드의 값이 변경되어 체크 로직을 추가할 때 많이 사용된다. MODULE,SELECT 구문과 함께 사용되는 것이 일반적 이며 다음과 같이 사용된다. FIELD WBTALE-FIELD MODULE CHECK_FIELD ON REQUEST.
ON	FIELD 구문과 함께 사용
VALUES	FIELD KEYWORD와 같이 사용
CHAIN	CHAIN 처리를 시작하며 여러 개의 필드를 그룹으로 처리
ENDCHAIN	PROCESSING CHAIN을 종료
CALL	SUBSCREEN을 호출
LOOP	LOOP 프로세싱을 시작
ENDLOOP	LOOP 프로세싱을 종료
MODIFY	스크린 TABLE을 변경
ON	FIELD 구문과 함께 사용
PROCESS	PROCESS 이벤트를 정의 PROCESS BEFORE OUTPUT...
SELECT	다음 구문과 같이 SELECT 구문 사용, '*' 제약사항 O

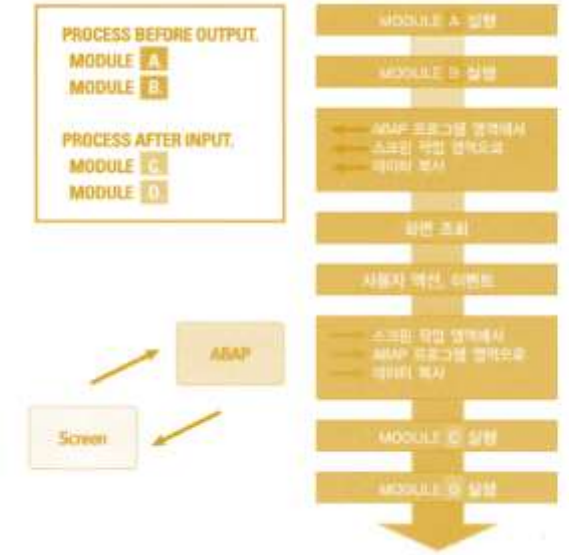
	SELECT * FROM TABLE NAME WHERE TABLE-KEYFIELD = INPUTFIELD AND... INTO FIELDNAME WHEREVER NOT FOUND | FOUND SEND ERRORMESSAGE | WARNING MESSAGENUMBER WITH FIELDNAME ...	
VALUES	INPUT 값을 정의 FIELD FIELD_NAME VALUES | LIST OF VALUES	
	VALUE	입력 가능한 SINGLE VALUE
	NOT VALUE	입력 불가능한 SINGLE VALUE
	BETWEEN VALUE AND VALUE	입력할 수 있는 VALUES 범위
	NOT BETWEEN VALUE AND VALUE	입력할 수 없는 VALUES 범위

스크린 간 호출 시 PBO / PAI 순서 |



위는 하나의 스크린에서 다른 스크린을 호출할 때 PBO와 PAI가 호출되는 순서를 설명한다.

SCREEN FLOW LOGIC 순서 |



스크린 LOGIC과 PBO, PAI 모듈이 실행되는 순서를 보여준다. 사용자가 T-CODE를 입력하여 화면을 조회하면 먼저 PBO 모듈 A, B가 실행된다. 이때는 ABAP 작업 영역 데이터가 스크린 작업 영역으로 옮겨가 화면이 조회되며, 조회된 화면에서 사용자가 이벤트를 발생시키면 PAI의 C, D 모듈이 실행된 수 다시 PBO를 수행하여 화면이 조회되는 순서로 이루어져 있다.

USER ACTION | INPUT FIELD DATA 입력

사용자는 스크린 상에서 INPUT 필드에 값을 넣거나 변경할 수 있다. INPUT FIELD에 입력하는 액션은 PAI 이벤트를 발생시키지 않지만. FUNVTION CODE를 가진 INPUT 필드 | CHECK BOX, RADIO BUTTON. DROP DOWN BOX는 이벤트를 발생시킬 수 있다.

USER ACTION | PAI EVENT 실행

SAP GUI 상에서 스크린과 사용자의 상호 작용을 결정하고 APPLICATION SERVER의 실행 환경에서 PAI 이벤트를 호출하는 방법에는 여러 가지가 있다.

PUSHBUTTON을 선택
FUNCTION CODE가 할당된 CHECK BOX 또는 RADIO BUTTON 선택
MENU	STANDARD TOOLBAR	APPLICATION TOOLBAR 에 있는 FUNCTION 선택
KEYBOARD 에 있는 FUNCTION KEY 선택
DROP DOWN LIST 에 있는 엔트리 선택

만약, ELEMENT LIST 에서 OK_CODE 필드가 이름을 가지며 ABAP PROGRAM에서 같은 이름을 가진 변수가 존재한다면 사용자가 FUNCTION을 선택하면 그에 할당된 FUNCTION CODE가 OK_CODE 변수에 복사된다. 만약 OK_CODE가 설정되지 않은 상태에서 PAI 이벤트가 실행되면 FUNCTION CODE의 값을 채워줄 스크린 필드가 존재하지 않기 때문에 프로그램에서 PAI에 해당하는 작업을 수행하기 어렵다. 즉, PAI 이벤트가 발생하면 그 순간에 시스템 변수 SY-UCOMM 값이 스크린 필드 OK_CODE에 복사된다. 모듈 풀 프로그램에서 OK_CODE를 SY-UCOMM 대신 사용하는 이유는 ABAP 프로그램에서 자체적으로 선언된 모든 변수를 관리하기 위함과, 시스템 필드는 시스템에서 사용되는 변수로 프로그램에서 변경하는 것은 삼가야 하기 때문이다. NEXT SCREEN이 호출되었을 때 OK_CODE를 초기화하지 않으면 이전의 FUNCTION CODE를 저장하고 있기 때문에 다른 스크린이 호출되었을 때 PBO에서 원치 않은 액션이 수행될 수 있다. 또는 사용자가 화면에서 ENTER을 입력하면 SY-UCOMM에는 아무런 값이 전달되지 않으며 이전의 FUNCRTION CODE를 가진다. 이런 문제를 방지하기 위해 스크린에서 PAI를 수행한 후 OK_CODE를 초기화 하는 작업이 필요하다. 즉. 스크린마다 OK_CODE 변수를 지정해야 하는 것이다.

USER ACTION | PROCESSING INPUT/OUTPUT 필드

INPUT / OUTPUT 필드는 키보드나 VALUE LIST 상에서 값을 입력하는 FIELD, RADIO BUTTON, CHECKBOX 등을 의미한다. 모든 스크린 필드들은 스크린 필드와 연결된 이름을 가져야 한다. 여기서 중요한 것은 데이터 타입이라 할 수 있으며 스크린 필드의 데이터 타입이 입력 값의 포맷을 결정한다. 따라서 숫자 필드에 문자형 값을 넣을 수 없으며 이는, 스크린이 사용자가 유효하지 않은 값을 입력하고자 할 때 스크린 필드에서 이를 인식하기 때문이다. RADIO BUTTON 과 CHECKBOX는 항상 CHAR 1자리의 데이터 타입을 가지고 있으며 선택되면 'X' 해제된 상태라면 빈 값("")을 가져야 한다.

USER ACTION | F1 | FIELD HELP

사용자가 화면에서 F1 키를 누르거나 HELP 아이콘을 클릭하면, 현재 커서가 존재하는 필드의 텍스트 도움말이 조회된다. ABAP DICTIONARY 필드의 DATA ELEMENT DOCULENTATION이 생성되어 있으면 스크린에 도움말 문서가 자동으로 첨부된다. DATA ELEMENT DOCULENTATION [A]은 프로그램과 스크린에 한정된 도움말을 생성하며 DATA ELEMENT DOCULENTATION [D] 는 ABAP DICTIONARY 레벨에서 생성한다. DICTIONARY 레벨에서 도움말을 변경하면 해당 필드를 참고하는 모든 스크린 필드들의 도움말이 변경되므로 주의해서 사용해야 한다. 스크린의 POH 이벤트에 PROCESS ON HELP-REQUEST를 추가하며 도움말을 생성한 필드를 호출하는 구문인 FIELD 〈F〉 [MODULE 〈MOD〉] WITH 〈NUM〉. 을 추가한다. MODULE 〈MOD〉 구문은 선택 조건으로 도움말을 수행하기 전 추가하고 싶은 로직을 구현할 수 있다.

USER ACTION | F4 | INPUT HELP

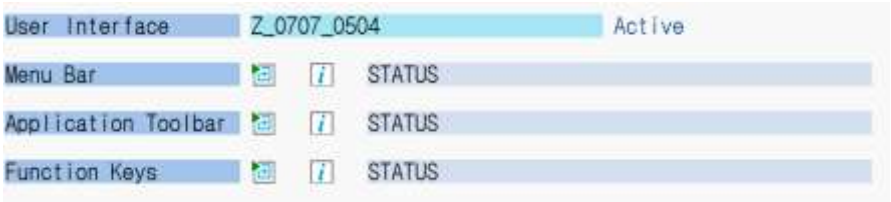
SEARCH HELP에서 학습한 내용으로 3가지 방법을 통해 추가할 수 있다.

ABAP DICTIONARY 를 이용한 INPUT HELP	SEARCH HELP를 생성하여 TABLE FIELD 에 할당하고, 스크린 필드는 TABLE FIELD의 속성을 상속받아 INPUT HELP로 사용할 수 있다.
스크린을 이용한 INPUT HELP	스크린 페인터에서 개별 필드에 직접 SEARCH HELP를 할당하거나, 스크린의 PAI 이벤트에서 입력 값을 제한할 수 있다.
DIALOG MODULE 에서의 INPUT HELP	사용자가 SCREEN 필드에서 F4 키를 누를 때, POV | PROCESS ON VALUE-REQUEST 이벤트에서 DIALOG 모듈을 호출함으로써 INPUT HELP를 화면에 보여줄 수 있다.

TYPE -M PROGRAM | 생성과 화면 디자인

모듈 프로그램 이름은 SAPM으로 시작 + CBO 구분자 Z + 모듈 구분명 + 프로그램 순번으로 지정하는 것이 일반적이다. 프로그램 TYPE은 M MODULE POOL 을 선택하며, 마우스 오른쪽 버튼을 클릭하여 CREATE | DYNPRO 를 통해 SCREEN을 생성한다. 스크린을 디자인하기 위해서는 스크린 페인터를 실행한다. 스크린 페인터는 스크린을 정의하고 DIALOG STEP을 처리하는 ABAP WORKBENCH TOOL의 한 종류로 GRAPHICAL 에디터와 ALPHANUMERIC 에디터 모드를 포함한다. UTILITIES | SETTING | GRAPICAL LAYOUT EDITOR 를 해제하면 ALPHANUMERIC 에디터를 사용할 수 있다. ALPHANUMERIC 에디터는 GUI 환경이 사용에 불편하기 때문에 DRAG & DROP 과 같은 편리한 기능을 사용할 수 있는 GRAPHICAL 에디터를 사용한다.

TYPE -M PROGRAM | GUI STATUS 생성



GUI STATUS는 그림과 같이 3가지 종류가 존재한다. 모든 스크린은 MENU BAR, STANDARD TOOLBAR, APPLICATION TOOLBAR를 가지는 GUI STATUS를 추가하여야 한다.

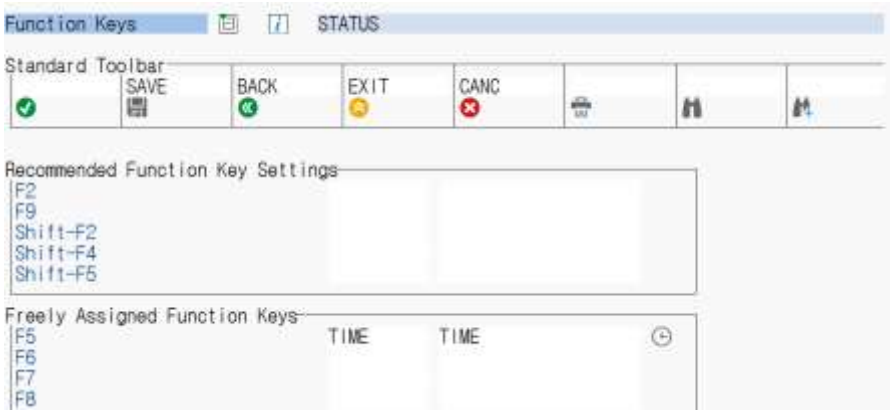
GUI STATUS | FUNCTION TYPE

E	트랜잭션 종료 시 설정하는 FUNCTION TYPE으로 AT EXIT-COMMAND 모듈이 수행된다
S	시스템이 사용하는 FUNCTION TYPE
T	트랜잭션을 호출하는 FUNCTION TYPE으로 FUNCTION CODE는 SAP 에 존재하는 트랜잭션 입력
SPACE	표준 APPLICATION FUNCTION TYPE
P	LOCAL GUI FUNCTION TYPE으로 LOCAL GUI LOGIC을 사용하는 TABSTRIP 에서만 사용
H	PROCESS ON HELP REQUEST 이벤트가 호출된다

GUI STATUS | FUNCTION KEY

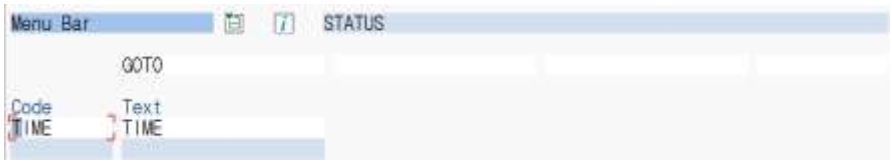
FUNCTION KEY는 스크린을 생성하면 기본적으로 제공되는 STANDARD TOOLBAR와 개발자가 추가하게 되는 FUNCTION KEY 가 존재한다.

GUI STATUS | FUNCTION KEY



STANDARD TOOLBAR는 SAP에 존재하는 모든 스크린에 기본적으로 제공되는 TOOLBAR로 FUNCTION KEY는 F1, SHIFT + F1 과 같이 키보드의 기능키를 이용해 빠르게 접근할 수 있도록 한다. FUNCTION KEY는 RESERVED FUNCTION KEY, RECOMMENDED FUNCTION KEY, FREELY ASSIGNED FUNCTION KEY로 구별된다. RESERVED FUNCTION KEY는 F1 | HELP, F3 | BACK, F4 | POSSIBLE ENTRY 와 같이 이미 예약되어 있어 개발자가 정의하여 사용할 수 없다.

GUI STATUS | MENU BAR



MENUBAR은 표준 프로그램에서 기본적으로 제공하나, 추가로 구성하고 싶은 메뉴를 설정할 수 있다.

PBO 모듈 | 화면의 속성을 초기화하거나, 필드의 초기값 설정 시 사용

화면을 초기화 하거나 스크린 요소의 속성을 변경하는 작업을 주로 수행한다, INCLUDE 프로그램 생성 시 개발환경에 맞게 이름을 변경하는 것이 좋다. 보통 프로그램 명 + 기능 + 순번으로 지정하며 기능은 다음과 같다.

T	변수, 인터널 테이블 등을 그룹화한 INCLUDE 프로그램
S	파라미터 등을 그룹화 한 INCLUDE 프로그램
C	클래스 오브젝트를 한 그룹으로 모은 INCLUDE 프로그램
F	PERFORM 구문을 그룹화 한 INCLUDE 프로그램
O	PBO 모듈 INCLUDE 프로그램
I	PAI 모듈 INCLUDE 프로그램

PBO 모듈 | 모듈 구현 및 출력 전용 속성 변경

스크린 레이아웃 편집기의 필드 속성을 이용해 출력 전용으로 변경할 수 있으며 일반 필드 속성은 다음과 같다.

NAME	필드 이름 * # / _ - $ 문자 가능
TEXT	필드 텍스트
DROPDOWN	INPUT OUTPUT 필드만 설정 가능 LIST BOX 설정 | WITH KEY 옵션
WITH ICON	OUTPUT 필드만 설정 가능
ICON NAME	TEXT 필드 PUSH 버튼, RADIO 버튼, CHECK BOX 만 ICON 설정 가능
SCROLLABLE	DEF.LENGTH 가 VIS.LENGTH 보다 클 때 스크롤 사용
LINE / COLUM	필드의 라인/컬럼 위치를 설정
DEF.LENGTH	필드 속성의 길이를 설정
VIS.LENGTH	스크린에 보이는 필드 길이로 SCROLLABLE 선택
HEIGHT	높이 설정. TEXT 및 ENTRY ELEMENT는 항상 1의 값
GROUPS	MODIFICATION GROUP, 여러 개의 필드 한 번에 변경

FCTCODE	FUNCTION CODE 설정. PUSH 버튼과 DROPDOWN LIST 가 설정된 INPUT/OUTPUT 필드만 설정 가능
FCTYPE	FUNCTION TYPE 설정
ON_CTMENU	CONTEXT 메뉴 설정

ATTRIBUTES 의 DICT 속성 |

FORMAT	INPUT 필드의 데이터 타입을 설정
FROM DICT	ABAP DICTIONARY의 속성과 동일 사용 설정 | 복사
MODIFIED	레이아웃 에디터에 자동 반영 X 설정 시 ABAP DICTIONARY 속성이 변경되면 스크린 필드에 자동으로 반영
CONV. EXIT	변환 루틴 사용 | 두가지 변환 루틴 지원 CONVERSION_EXIT_ 〈NAME〉 _INPUT CONVERSION_EXIT_ 〈NAME〉 _OUTPUT
SEARCH HELP	탐색 도움말 사용
REF. FIELD	참조 필드 | 두가지 용도로 사용 1 TABLE STRIP CONTROL 에서 사용 | TABL TITALE 과 SUBSCREEN 연결 2 CURRENCIES AND QUANTITIES 필드의 참조 단위 필드로 사용
PARAMETER ID	파라미터 ID 설정
SET / GET PARAMETER	파라미터 설정 시 사용 SET | 사용자가 입력한 값을 PARAMETER ID에 저장 GET | PARAMETER ID 값을 화면에 조회되게 함
FOREIGN KEY CHECK	ABAP DICTIONARY 에서 정의된 외부 키 체크
UPPER / LOWER CASE	대소문자 구분을 설정하지 않으면 사용자가 입력한 모든 문자는 대문자 | UPPER CASE로 인식

ATTRIBUTES 의 PROGRAM 속성 |

INPUT 필드	INPUT 필드를 입력 가능 속성으로 설정 REQUIRED 로 설정되면 필수 입력 필드로 설정되며 ? 또는 ☑ 로 표시
OUTPUT 필드	INPUT 필드 조회 속성 변경
POSS. ENTRIES	POSSIBLE ENTRY 속성 설정 INPUT OUTPUT만 설정 0 보이지 않음 1 필드 선택 시 보임 2 항상 보임
RIGHT-JUSTIFIED	숫자 필드 오른쪽 정렬
LEADING ZEROS	NUMS 타입에만 사용, 자리 수만큼 0으로 채워서 조회
*ENTRY	사용자에게 첫 자리에 애스터리스크 *를 입력할 수 있도록 함 첫째 자리의 애스터리스트 * 문자는 다음 모듈을 호출 FIELD... MODULE... ON *-INPUT
WITHOUT TEMPLATE	. ! ? _ 와 같은 어떠한 문자도 입력할 수 있도록 함 사용자가 입력한 문자는 SAP 시스템에서 해석 절차를 거치지 않는 단순 문자열로 인식

ATTRIBUTES 의 DISPLAY 속성 |

FIXED FONT	INPUT OUTPUT 필드와 텍스트 필드의 글자 폰트 고정 FIXED FONT 사용시 OUTPUT ONLY 필드로 설정
BRIGHT	스크린 요소의 글자를 강조
INVISIBLE	스크린에서 보이지 않게 설정
2D DISPLAY	2D로 보이게 함 INPUT OUTPUT 필드에 ICON 설정 시 자동으로 설정됨
AS LABEL ON LEFT	텍스트 필드 왼쪽 정렬 TEXT 필드, INPUT/OUTPUT 필드에 설정 가능 INPUT 필드는 설정할 수 X

AS LABEL ON RIGHT	텍스트 필드를 오른쪽 정렬 TEXT 필드, INPUT/OUTPUT 필드에 설정 가능 INPUT 필드는 설정할 수 X
RESPONDS TO D-CLICK	더블 클릭에 반응	HOTSPO 설정 TEXT FIELD , INPUT/OUTPUT 필드에만 설정 가능

스크린 속성 |

NAME	필드 이름
GROUP1	MODIFICATION 그룹 1 이름
GROUP2	MODIFICATION 그룹 2 이름
GROUP3	MODIFICATION 그룹 3 이름
GROUP4	MODIFICATION 그룹 4 이름
REQUIRED	필수 필드 설정
INPUT	입력 가능 필드 설정
OUTPUT	출력 전용 설정
INTENSIFIED	강조
INVISIBLE	보이지 않게 함
LENGTH	길이 설정
ACTIVE	활성화	0으로 설정 시 다음 속성과 같음 INPUT = 0. OUTPUT =0 , INVISIBLE = 1
DISPLAY_3D	TREE-DIMENSIONAL BOX
VALUE_HELP	INPUT HELP BUTTON DISPLAY
REQUEST	입력 대기	READY FOR INPUT

INCLUDE 프로그램 | R/3 REPOSITORY에 저장되는 오브젝트

다음과 같은 2가지 기능을 목적으로 개발된다.

LIBRARY 모듈화	INCLUDE 프로그램은 같은 소스 코드를 다른 프로그램에서 사용할 수 있도록 함
ORDER 순서	가독성	INCLUDE 프로그램은 복잡한 프로그램을 순서대로 정렬하여 뛰어난 가독성을 제공

INCLUDE 프로그램에는 다음과 같은 특성이 있다.

INCLUDE 는 하나의 프로그램이나 독립적으로 실행될 수 없음
INCLUDE 프로그램은 다른 프로그램 내에 내장	BUILT IN 되어야 함
INCLUDE 프로그램은 또 다른 INCLUDE를 포함할 수 있음
INCLUDE 프로그램은 자기 자신을 호출할 수 없음
INCLUDE 프로그램은 파라미터를 가지지 않음

PAI 모듈 구현 |

사용자가 입력 필드에 값을 입력하고 ENTER를 입력하거나 스크린의 푸시 버튼을 클릭했을 때와 같은 USER DIALOG에 반응하는 이벤트.

TRANSACTION | T-CODE 생성

TRANSACTION CODE는 프로그램을 실행시키는 명령어로서 우리가 개발하게 되는 프로그램을 최종 사용자가 실행할 수 있도록 해준다. T-CODE의 종류에는 크게 5가지가 있으며 SE93에서 생성 변경 조회할 수 있다. TSTC/TSTCT 테이블에 트랜잭션 정보가 저장되며 TRANSACTION CODE는 최대 20자까지 이름을 지정할 수 있다.

DIALOG TRANSACTION	TYPE-M 모듈 풀 프로그램에서 사용되는 트랜잭션 코드로서 스크린과 연결되어 프로그램을 실행
REPORT TRANSACTION	TYPE-1 리포트 프로그램을 시작하는 트랜잭션 코드
OBJECT-ORIENTED TRANSCTION	ABAP 오브젝트인 클래스의 메서드를 트랜잭션 코드 생성
VARIANT TRANSACTION	트랜잭션 변형 | TRANSACTION VARIANT 를 사용해 프로그램의 필드를 조정 가능 표준 프로그램 필드 조회모드로 변경 등 작업 가능
PARAMETER TRANSACTION	트랜잭션이 수행된 스크린 필드에 초기 값을 지정하며 스크린 필드 이름과 값을 파라미터에 입력한 후 트랜잭션 실행 시 스크린 필드에 초기 값이 설정됨

TRANSACTION |

트랜잭션의 이름은 Z + 모듈 + 기능 + 순번으로 지정한다. 트랜잭션을 호출 시 스크린 필드명과 값을 파라미터로 설정하여 전달할 수 있다. PARAMETER TRANSACTION은 화면 필드 값을 초기화 하는 것 이외에 유지보수 뷰를 트랜잭션으로 호출 시 유용하게 사용된다.

INPUT CHECK |

사용자가 화면의 INPUT FIELD 에 값을 입력하면, 필드가 허용하는 올바른 타입의 데이터인지 체크하는 과정을 거친다. 이를 검증 | VALIDATION이라 부른다. 스크린에는 3가지 유형의 INPUT CHECK 로직이 존재한다.

AUTOMATIC INPUT CHECK
INPUT CHECKS IN THE FLOW LOGIC
INPUT CHECKS IN DIALOG MODULE

INPUT CHECK | AUTOMATIC INPUT CHECK

AUTOMATIC INPUT CHECK는 스크린 필드의 데이터가 ABAP 프로그램으로 복사되거나 DIALOG 모듈이 호출되기 이전에 PAI 이벤트에서 자동으로 수행된다. 필수 입력 필드, 데이터 포맷 체크 그리고 ABAP DICTIONARY 레벨의 체크가 수행된다. AUTOMATIC INPUT CHECK 가 수행되기 이전에, FUNCTION CODE 가 'E' 타입으로 선언된 MODULE 〈XXX〉 AT EXIT-COMMAND를 호출할 수 있다. 만약 필수 필드가 설정된 스크린에서 사용자가 EXIT 버튼을 클릭하여 화면을 빠져나가려고 할 경우 FUNCTION CODE가 'E'로 설정되어 있지 않으면, AUTOMATIC INPUT CHECK가 가장 먼저 수행되어 에러가 발생하고 사용자는 화면을 빠져나갈 수 없게 된다. 즉. AT EXIT-COMMAND는 필수 필드를 입력하지 않고 스크린을 빠져나가기 위한 목적으로 주로 사용된다.

INPUT CHECK | INPUT CHECK IN THE FLOW LOGIC

DIALOG MODULE이 호출되기 이전에 FLOW LOGIC에서 INPUT CHECK를 수행하며 NOT 'VALUE' | BETWEEN 'VALUE' AND 'VALUE' | NOT BETWEEN 'VALUE' AND 'VALUE' 와 같은 옵션들이 존재한다. 또한 데이터베이스 테이블 레벨에서 체크가 가능하다.

INPUT CHECK | INPUT CHECK IN DIALOG MODULE

PAI 모듈에서 INPUT CHECK를 수행한다. 다음 구문으로 INPUT CHECK를 수행하며 실무에서 가장 많이 사용하는 방법이다. FIELD 〈F〉 MODULE 〈MOD〉. 사용자가 입력한 값을 제한하거나 테이블에 존재하는 값인지 체크할 수 있으며 모듈로 존재하기 때문에 많은 기능을 추가하여 사용할 수 있다. FIELD 〈F〉 MODULE 〈MOD〉 ON REQUEST. 필드 F의 값이 초기값과 다를 때만 모듈 MOD 가 수행되며 DATA TYPE마다 INITIAL VALUE가 다르다. CHAR 타입은

SPACE가 초기 값이며 NUMERIC 필드는 0이 초기 값으로 필드의 F 값이 변경 될 때 마다 모듈 MOD가 수행된다.

SCREEN 호출 |

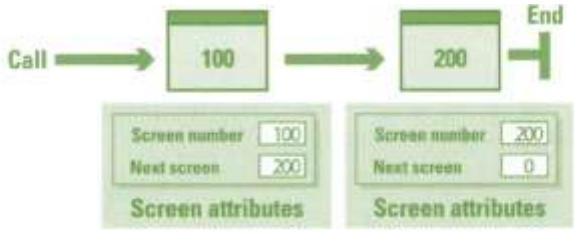
모듈 풀 프로그램 내에서 스크린을 호출하는 방법에는 두 가지가 있다.

```
SET SCREEN NN.
CALL SCREEN NN.
```

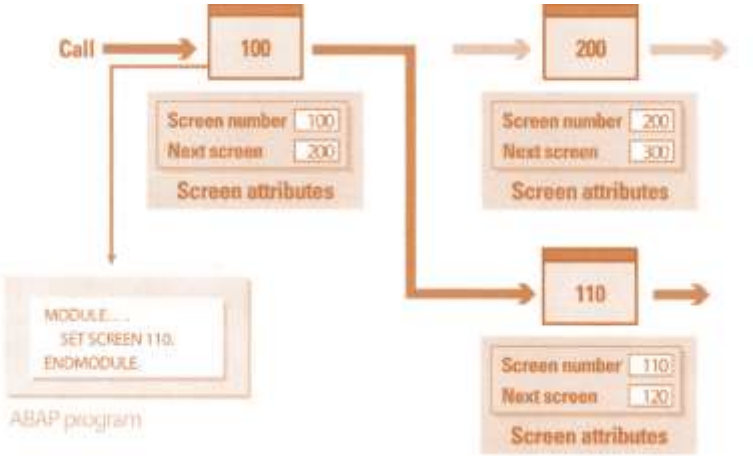
SCREEN 호출 | SET SCREEN NN.

SET SCREEN 구문을 이용하여 스크린을 호출하면, 현재 수행 중인 PAI 모듈을 다 수행할 후 스크린을 호출하며 현재 스크린을 종료한다.

그림과 같이 NEXT SCREEN이 200번으로 설정되어 있다면 ABAP 프로그램 내에서 LEAVE SCREEN 구문을 만나게 되면 200번 스크린으로 이동한다.



그러나 아래와 같이 ABAP 프로그램 내에서 SET SCREEN 110. 구문을 호출하게 되면 200번 스크린이 아닌 110번 스크린으로 이동한다. 이를 DYNAMIC SCREEN SEQUENCING이라 한다.



SCREEN 호출 | CALL SCREEN NN.

CALL SCREEN 구문을 이용해 스크린을 호출하면 CALL SCREEN 구문을 만나는 즉시 호출한 스크린으로 이동한다. 그리고 현재 스크린은 종료되지 않고 비활성화된 상태로 존재한다. SET SCREEN은 추가 옵션이 존재하지 않으나, CALL SCREEN은 다음과 같은 구문을 추가하여 사용할 수 있다.

```
CALL SCREEN NN
        STARTING AT X1 Y1 ENDING AT X2 Y2
```

SCREEN 종료 |

ABAP 프로그램에서는 다음 두 가지 구문을 사용하여 스크린을 빠져나온다.

```
LEAVE SCREEN.
LEAVE TO SCREEN NEXT SCREEN.
```

SCREEN 종료 | LEAVE SCREEN

LEAVE SCREEN 구문은 현재 화면을 종료하고 | END 이어서 수반되는 다음 화면을 호출한다. NEXT SCREEN 은 STATIC NEXT 스크린 또는 DYNAMIC NEXT 스크린을 수 있으며 이때 DYNAMIC NEXT 스크린 | 동적 다음 화면 이라는 것은 앞에서 설명했듯이 LEAVE 스크린 구문 이전에 SET 스크린 구문으로 STATIC NEXT 스크린을 겹쳐 쓴 경우를 의미한다.

SCREEN 종료 | LEAVE TO SCREEN NEXT SCREEN.

LEAVE TO SCREEN 구문은 현재 화면을 빠져나오고 | EXIT , DYNAMIC NEXT 스크린을 호출한다. 즉 LEAVE TO SCREEN 은 다음 두 구문을 합쳐 놓은 효과를 가진다. SET SCREEN NN. LEAVE SCREEN. 그러나 LEAVE TO SCREEN. 구문은 현재의 화면을 종료 | END 하지 않고, 단순히 다른 화면을 호출하기만 한다. 모든 스크린을 종료하려면 LEAVE TO NEXT SCREEN 0. 구문을 사용한다.

TABLE CONTROL |

화면에 많은 데이터가 조회되어야 할 경우 테이블 컨트롤을 사용한다. 테이블 컨트롤은 칼럼 헤더와 칼럼 정보를 가지는 SIMPLE 컨트롤의 한 종류로서, 엑셀 시트와 같이 TABBULAR 형태로 데이터를 보여준다. ABAP은 스크린 내에서 테이블을 보여주거나, 조정할 때 두 가지 형태의 메커니즘을 제공하는데 이는, TABLE CONTROL 과 STEP LOOP로, TABLE CONTROL 은 하나의 정의된 ROW 들로 구성된 테이블이며, STEP LOOP 는 하나 이상의 ROW를 확장하는 개념으로 T-CODE: MB1A 와 같은 트랜잭션에서 INPUT 필드들을 그룹으로 묶어서 LOOP를 수행하면서 처리할 수 있도록 한다. 테이블 컨트롤이 STEP LOOP 보다 장점이 많아 주로 사용되며 현재 실무에서는 ALV가 등장하면서 테이블 컨트롤보다 ALV가 많이 사용되고 있다.

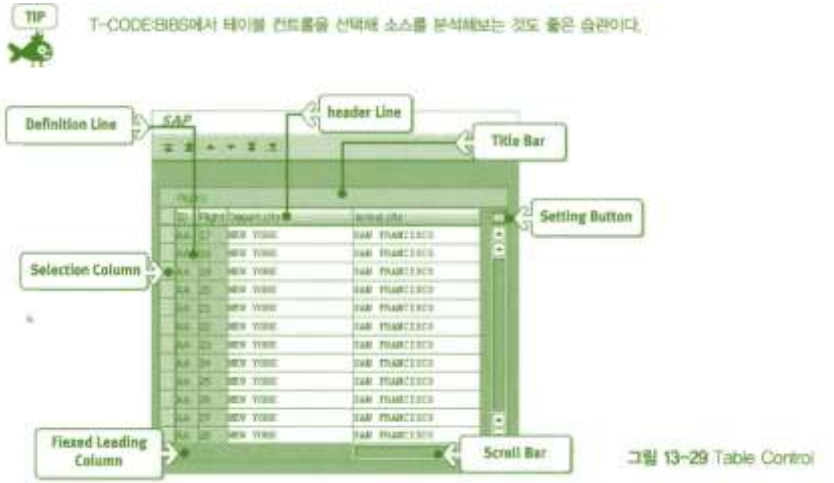


그림 13-29 Table Control

```
테이블 컨트롤의 LINE은 KEYWORLDS INPUT/OUTPUT RADIO BUTTON CHECKBOX RADIOBUTTON GROUP PUCHBUTTON을 포함할 수 있다.
LINE은 255 칼럼까지 지정할 수 있다.
```

TABLE CONTROL | 생성

모듈 풀 프로그램에서 테이블 컨트롤을 추가하려면 다음과 같은 순서로 진행한다.

테이블 컨트롤 영역을 정의한다.
테이블 컨트롤 구성 요소를 선언한다. 테이블 컨트롤의 TITLE을 추가한다	선택 사항
MODULE POOL 프로그램 내에서 테이블 컨트롤을 선언한다.

TABLE CONTROL | 속성

W/COLHEAD	테이블 컨트롤의 헤더 표시 여부
CONFIGBL	SETTING BUTTON 활성화
W/TITLE	TITLE 바 추가
RESIZING	테이블 컨트롤의 크기 변경 가능
SEPARATORS	수평/수직 구분자
LINE SEL.	라인 선택 시 멀티 선택 여부
COLUMN SEL.	칼럼 선택 시 멀티 선택 여부
W/SELCOLUMN	선택 버튼 추가 예제에서는 DEMO_CONN-MARK 라는 이름을 지정함
FIXED COLUMNS	고정칼럼 개수

TABLE CONTROL | 선언

TABLE CONTROL을 사용하려면 다음 문장을 프로그램 선언부 | TOP 에 기술한다. CONTROLS CTR TYPE TABLEVIEW USING SCREEN SCR. CONTROL 구문은 COMPLEX DATA OBJECT인 컨트롤을 생성한다.

CTR	테이블 컨트롤의 이름
SCR	테이블 컨트롤을 사용하는 스크린 번호

TABLE CONTROL 스크립트 구현 | LOOP

TABLE CONTROL을 사용하려면 PBO와 PAI 모듈에 LOOP 구문을 추가해야 한다. 이는 LOOP 구문을 수행하며, ABAP 프로그램 필드와 스크린 필드의 데이터가 연결되도록 한다. ROW BY ROW로 복사된다. PBO 구문의 LOOP 구문은 ABAP 프로그램 필드 데이터가 스크린 필드 | 테이블 컨트롤에 조회되게 하고, PAI 구문의 LOOP 구문은 스크린 필드 | 테이블 컨트롤 의 데이터가 ABAP 프로그램 데이터에 복사되게 한다. 에러를 피하려면 PBO/PAI에 내용이 없는 LOOP 구문이라도 추가해야 하며 프로그램 실행 시 데이터가 없는 테이블 컨트롤이 조회된 것을 확인하게 한다.

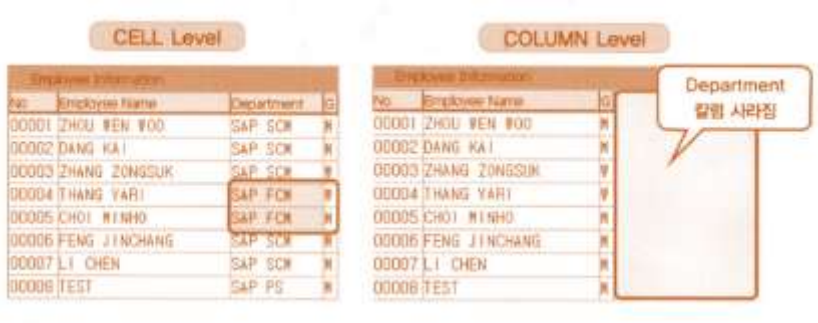
PBO |	PAI |
LOOP [WITH CONTROL Z_CON]. ENDLOOP.	LOOP [WITH CONTROL Z_CON] . ENDLOOP.

TABLE CONTROL 스크립트 구현 | LOOP AT

TABLE CONTROL에서 LOOP 구문을 수행하는 방법은 스크린을 이용하는 방법과 인터널 테이블 이용하는 방법 두 가지가 있다. PBO에서 LOOP AT 구문을 사용하게 되면, 라인 수 X 칼럼 수만큼 LOOP를 수행하게 되므로 테이블 컨트롤의 필드 속성을 동적으로 | DYNPROS 변경할 수 있다.

LOOP [WITH CONTROL Z_CON]. ENDLOOP.	LOOP AT 〈INTERNAL TABLE〉. ENDLOOP.
스크린 필드와 ABAP 프로그램 영역 상호 간에 데이터를 LINE BY LINE 형태로 복사	스크린 필드와 ABAP 프로그램 영역 상호 간에 데이터를 병행	IN PARALLEL 하여 수행

TABLE CONTROL 칼럼 속성 변경 |



테이블 컨트롤의 칼럼 속성을 변경하는 방법은 CELL, COLUMN 레벨로 크게 두 분류가 있다. CELL 레벨에서 필드의 속성을 변경하려면 LOOP AT 구문을 사용하여 병렬 | IN PARALLEL 처리를 수행하며, CELL을 변경해야 한다. 또한 COLUMN 단위의 속성을 변경하려면 CXTAB_CONTROL 구조체의 속성을 이용해야 한다.

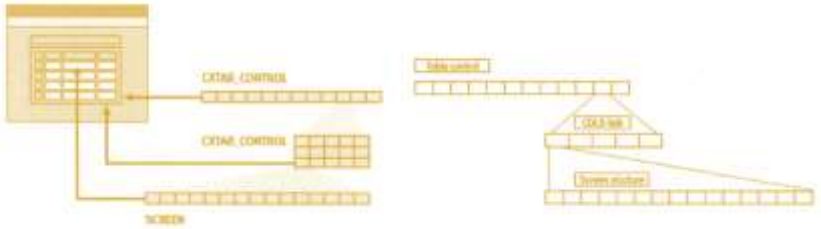
TABLE CONTROL 칼럼 속성 변경 | CELL 레벨 테이블 컨트롤 속성 변경

PBO 이벤트의 LOOP AT 구문 내에 테이블 컨트롤 속성을 변경하기 위한 모듈을 선언하며 LOOP AT ITAB. 구문을 사용하게 되면. 테이블 컨트롤의 칼럼 수 X LINE 수만큼 LOOP를 수행하게 된다.

TABLE CONTROL 칼럼 속성 변경 | COLUMN 레벨 테이블 컨트롤 속성 변경

CELL 레벨에서는 한 칼럼 전체를 보이지 않게 설정할 수 없다. 사용자가 원하는 칼럼 전체를 화면에서 보이지 않게 설정하려면 칼럼 레벨에서 테이블 컨트롤의 속성을 변경해야 한다. 아래의 테이블 컨트롤은 CXTAB_CONTROL 타입을 참고하며 CXTAB_COLUMN 타입 구조를 지니고, 스크린 구조를 취한다.

TABLE CONTROL 칼럼 속성 변경 | COLUMN 레벨 테이블 컨트롤 속성 변경



TABLEVIEW TYPE은 CXTAB_CONTROL 구조체를 참고한다. DEEP STRUCTURE 인 CXTAB_CONTROL은 테이블 컨트롤의 속성들을 포함하고 있다. CXTAB_COLUMN과 칼럼 속성을 포함하는 테이블로 CXTAB_CONTROL-COLS-SCREEN의 항목들은 SYSTEM 테이블인 스크린과 같은 타입의 FLAT STRUCTURE이다.

FIXED_COLS	I	고정 칼럼 수
LINES	I	테이블의 컨트롤 라인 수 수직 스크롤 설정을 위한 목적으로 사용
TOP_LINE	I	스크롤을 선택해 PAI 호출하면 NEXT PBO에서 TOP에 조회되는 라인 번호로 ABAP 프로그램에서 변경 가능
CURRENT_LINE	I	LOOP..ENDLOOP 내 현재 라인 번호 SY-STEPL+(TOP_LINE-1) 값으로 시스템에서 자동 세팅 되며 ABAP 프로그램에서 변경 가능
LEFT_COL	I	수평 스크롤 선택 시 왼쪽의 고정 칼럼 수 설정
LINE_SEL_MODE	I	라인 선택 모드 설정 0 라인 선택할 수 없음 1 SINGLE 라인 선택 가능 2 MULTIPLE 라인 선택 가능

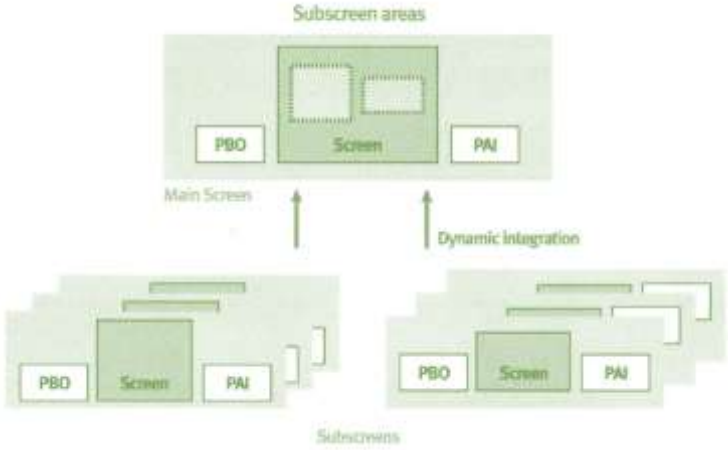
COL_SEL_MODE	I	칼럼 선택 모드 설정 0 칼럼 선택할 수 없음 1 SINGLE 칼럼 선택 가능 2 MULTIPLE 칼럼 선택 가능
LINE_SELECTOR	C(1)	속성이 'X'로 설정되어야 라인 선택 가능 LINE_SEL_MODE 효력을 가짐 ABAP 프로그램에서 변경할 수 있음
H_GRID	C(1)	수평 구분자
V_GRID	C(1)	수직 구분자
COLS	CXTAB_ COLUMN	다음 표에 설명
INVISIBLE	C(1)	보이지 않게 설정

시스템 변수 SY-LOOP 필드는 스크린의 테이블 컨트롤에 조회되는 전체 라인 수를 저장한다. SY-STPEL은 LOOP 구문에서 라인의 인덱스 내 값을 가지게 되며, SY-LOOPC 는 항상 조회된 전체 라인 수 4의 값을 가진다. 반면 시스템 변수 SY-STEPL은 스크린 테이블의 현재 라인의 인덱스를 저장한다. 시스템은 LOOP를 수행하면서 이 변수를 변경하며, SY-STEPL은 항상 1에서 현재 조회되고 있는 번호까지의 값만 가질 수 있으며 이는 스크린 필드와 인터널 테이블 값을 서로 주고받을 때 유용하게 작용한다.

CXTAB_CONTROL | 속성

SCREEN	SCREEN	스크린 구조의 속성
INDEX	I	칼럼 위치	조회 순서
SELECTED		선택된 칼럼
VISLENGTH	ICON_OLENG	칼럼이 보이는 넓이
INVISIBLE		칼럼이 보일 것인지를 설정

SUBSCREEN |



SUBSCREEN은 RUNTIME 환경에서 다른 스크린의 SUB AREA에 포함되도록 한다. 즉, 위의 그림과 같이 하나의 스크린이 또 다른 스크린을 자기 자신 안에 포함하는 것이다. SUBSCREEN은 주로 TABSTRIP에서 사용한다. SUBSCREEN은 각각의 FLOW LOGIC을 가질 수 있으며, 일반 스크린에서 사용하는 방법과 몇 가지를 제외하고는 유사하다. 예를 들어 SUBSCREEN은 메인 스크린의 OK_CODE를 상속받기 때문에 자신의 OK_CODE 필드를 가질 수 없으며 MODULE ... AT EXIT-COMMAND 모듈은 메인 스크린에서만 관리할 수 있다. SUBSCREEN 을 생성하려면 다음의 순서로 작업한다.

스크린 상에서 스크린 AREA를 정의
SUBSCREEN 정의
SUBSCREEN 영역 내 SUBSCREEN을 배치

SUBSCREEN에서 특이사항 및 주의할 점은 다음과 같다.

SUBSCREEN의 크기 조정 때문에 배치된 스크린 요소들이 잘리지 않도록 스크린 요소를 재배치해야 함
여러 개의 SUBSCREEN을 생성할 시 각각의 스크린 요소들이 이름을 유일하게 지정해야 함. 이는 스크린과 ABAP 프로그램과의 데이터 통신이 일어날 시 문제가 생성되기 때문
SUBSCREEN은 OK_CODE 필드를 가질 수 없음. SUBSCREEN 상에서 발생한 사용자 액션에 대한 FUNCTION CODE는 메인 스크린의 OK_CODE필드에 저장
FLOW LOGIC 중 MODULE AT EXIT-COMMAND를 가질 수 없음. TYPE E FUNCTION은 MAIN PROGRAM 에서만 사용될 수 있기 때문
SUBSCREEN은 자신의 GUI STATUS를 가질 수 없으며 다음과 같은 구문을 사용할 수 없음 | RUNTIME ERROR 발생
SET TITLEBAR SET PF-STATUS SET SCREEN LEAVE SCREEN LEAVE TO SCREEN

SUBSCREEN | 영역 정의

SUBSCREEN 영역은 스크린 페인터에서 정의하게 되며, 각각의 SUBSCREEN은 이름 위치 길이 높이를 지정해야 한다. SUBSCREEN 영역은 스크린의 다른 요소와 겹쳐져서는 안된다. 최종 사용자가 윈도우의 크기를 조절할 때 SUBSCREEN도 수직/수평으로 크기를 조절할 수 있도록 설정할 수 있으며, 사용자가 윈도우 크기를 조절할 때마다 PAI 이벤트가 호출된다.

INCLUDE SUBSCREEN |

SUBSCREEN을 메인 스크린에 추가하려면 PBO와 PAI 이벤트에 다음 구문을 추가해야 한다.

PBO

CALL SUBSCREEN 〈AREA〉 INCLUDING 〈PROGRAM〉 〈DYNNP〉.

PAI

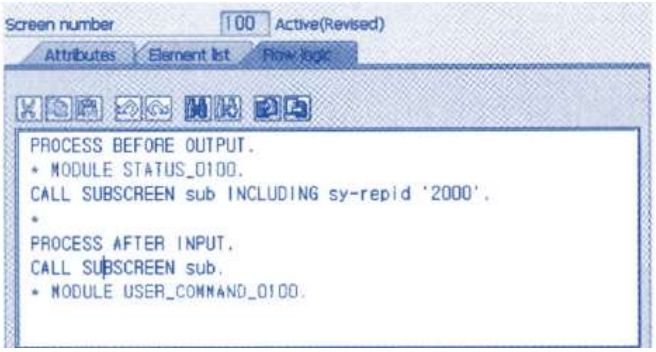
CALL SUBSCREEN 〈AREA〉.

PBO 이벤트에서 CALL SUBSCREEN 구문은 SUBSCREEN의 PBO FOW LOGIC을 호출한다. 메인 스크린의 CALL SUBSCREEN 구문 다음의 모듈은 SUBSCREEN의 PBO 로직이 수행된 이후에 호출된다. SUBSCREEN 영역에 추가되는 SUBSCREEN은 실행 시점에만 알 수 있으므로 DYNPRO | DYNAMIC PROGRAM 이라 부른다 . 이과 마찬가지로 PAI 이벤트의 CALL SUBSCREEN 구문은 SUBSCREEN의 PAI FLOW LOGIC을 호출하게 된다 SUBSCREEN 요소와 ABAP PROGRAM 필드가 같은 이름으로 존재하면 데이터 복사가 자동으로 일어나게 된다.

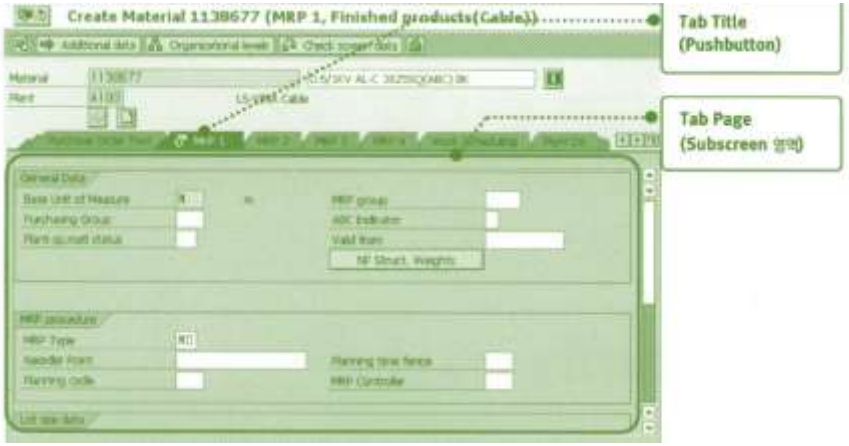
〈S_AREA〉	SUBSCREEN 영역 NAME
〈GV_REPID〉	프로그램에서 정의된 SUBSCREEN인지 지정 명시적으로 지정하지 않은 경우 메인 프로그램과 동일한 것으로 봄 | 선택사항 SUBSCREEN을 찾지 못할 때 RUNTIME ERROR 발생
〈GV_SNAME〉	스크린 번호

프로그램의 이름은 시스템 변수를 이용해 다음과 같이 사용할 수 있다.
GV_REPID = SY-REPID. 프로그램의 SUBSCREEN을 호출하고 싶으면 프로그램 내부의 SUBSCREEN이면 MODULE USER_COMMAND_0100 구현 부분에 〈SUBSCREEN1〉 버튼을 클릭한 후 GV_NAME 변수에 SUBSCREEN 이름을 할당하거나 외부 SUBSCREEN 을 호출할 경우 CALL SUBSCREEN 영역에 INCLUDING 프로그램 명 '스크린번호' 를 추가하면 된다.

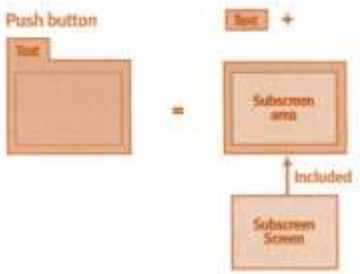
SUBSCREEN | SELECTION SCREEN을 SUBSCREEN으로 활용



TYPE M PROGRAM에 SELECTION-SCREEN을 추가하기 위해서는 MAIN SCREEN의 PBO와 PAI를 다음과 같이 구성한 후 트랜잭션 코드 생성 및 프로그램을 실행하면 TYPE-M 유형이나 TYPE-I 유형에서 사용하는 SELECTION SCREEN이 화면에 조회되는 것을 확인할 수 있다.



TABSTRIP 컨트롤은 두 개 이상의 스크린 오브젝트로 구성되어 있으며, 각각의 TAB 페이지는 하나의 TAB TITLE과 PAGE 영역으로 이루어진다. 다음의 그림은 TABSTRIP이 사용되는 전형적인 프로그램으로, 자재 코드를 조회하는 MM03 GHKAUSDLEK. MRP1 MRP2와 같이 헤더가 존재하는 부분들이 TABSTRIP으로 구성되어 있으며 각각의 TABSTRIP에는 SUBSCREEN이 추가되어 있다.



TABSTRIP은 기술적인 측면에서 SUBSCREEN과 PUSHBUTTON으로 구성되어 있다. 외형적인 측면에서는 TAB PAGE | SUBSCREEN 들의 세트로 구성되어 있으며 이러한 이유에서 TABSTRIP은 SUBSCREEN의 GUI STATUS를 사용할 수 없는 것과 같은 제한적인 특성을 상속받는다.

TABSTRIP | 종류

스크린 레이아웃에서 TABSTRIP과 TAB TITLE을 정의하려면. TABSTRIP AREA는 유일한 이름 위치 넓이 높이를 지정해야 한다. 사용자가 윈도우 크기를 조절할 시 TABSTRIP의 크기도 조절하여 설정할 수 있다. TABSTRIP AREA를 정의 | 생성 시 두 개의 TAB TITLE이 자동으로 생성되며, TAB TITLE은 기술적으로 PUSHBUTTON과 같은 속성을 지니고 있다. 이름이 존재하며, 텍스트를 지정해 동적인 할당을 할 수 있으며 FUNCTION CODE가 존재하며 아이콘을 보여줄 수 있다. TABSTRIP 생성은 다음과 같은 단계로 진행된다.

TABSTRIP | 종류

스크린에 TAB AREA 영역과 TAB TITLE을 정의
각 TAB TITLE에 SUBSCREEN 영역을 할당
스크린 FLOW LOGIC을 프로그래밍
ABAP PROCESSING LOGIC을 프로그래밍

또한 SAPGUI 또는 APPLICATION SERVER에서 PAGING을 할 것인지 결정해야한다. SAPGUI를 선택했다면 각각의 TAB PAGE는 자신의 SUBSCREEN을 가지고 있어야 하며, APPLICATION SERVER 선택 시 모든 TAB PAGE에서 공유되는 하나의 SUBSCREEN 영역만 지정하면 된다.

TABSTRIP | SAPGUI에서 PAGING

```
PROCESS BEFORE OUTPUT.
CALL SUBSCREEN:
  <area1> INCLUDING [<prog 1>] <dynp 1>,
  <area2> INCLUDING [<prog 2>] <dynp 2>,
  <area3> INCLUDING [<prog 3>] <dynp 3>,
          ...
PROCESS AFTER INPUT.
  CALL SUBSCREEN:      <area1>,
                       <area2>,
                       <area3>.
```

SAPGUI에서 PAGING은 문맥 의미 그대로, 사용자의 PC에 설치된 SAPGUI에서만 화면이 조회된다. 사용자가 TABTITLE을 선택할 때, APPLICATION SERVER의 FUNCTION CODE를 호출하지 않으며 PAI 이벤트를 호출하지 않음을 의미한다. 이는 스크린 필드의 어떠한 데이터도 ABAP 프로그램 영역으로 복사되지 않는다는 의미로 TABSTRIP 컨트롤의 성능이 개선되나, 사용자가 PAI 이벤트를 호출할 수 없어 INPUT CHECK를 SUBSCREEN마다 수행해야 하는 단점을 가진다. 따라서 LOCAL PAGING IN THE SAPGUI는 데이터를 조회하는 목적으로 사용하는 프로그램에 적합하다. SAPGUI에서 PAGING을 하려면 다음의 구문과 같이 TAB TITLE의 SUBSCREEN 영역에 개별 SUBSCREEN을 할당해야 한다. 만약 사용자가 저장 버튼을 클릭하는 것과 PAI를 호출하게 되면, TABSTRIP의 모든 스크린의 INPUT CHECK가 수행된다. 이의 의미는 개별 SUBSCREEN에 필수 필드가 존재하고 값이 입력되지 않으면 더는 작업을 진행할 수 없도록 한다. 이와 반대로 APPLICATION SERVER 에서 PAGING은 현재 활성화되어 있는 SUBSCREEN의 INPUT CHECK만 수행하게 된다. TAB TITLE의 FUNCTION CODE타입은 P LOCAL GUI FUNC 로 지정하며 사용자가 TAB TITLE을 선택할 때 APPLICATION SERVER에서 PAI를 호출하지 않는다. 또한 SAPGUI에서 SUBSCREEN이 자동으로 지정된다.

TABSTRIP | APPLICATION SERVER에서 PAGING

```
PROCESS BEFORE OUTPUT.
  ...
  CALL SUBSCREEN
       <area> INCLUDING [<prog>] <dynp>.
  ...
PROCESS AFTER INPUT.
  ...
  CALL SUBSCREEN <area>.
```

APPLICATION SEVER에서 PAGING을 사용하면, 사용자가 TAB PAGE를 변경할 때마다 PAI 이벤트가 호출된다. 하나의 SUBSCREEN 영역은 모든 TAB TITLE에서 공유되며 다음 구문에서 〈AREA〉 영역을 의미한다. 사용자가 FUNCTION CODE가 할당된 TAB TITLE을 클릭하면 〈DYNP〉에 SUBSCREEN을 변경하는 FLOW LOGIC을 추가해야 한다. 사용자가 TAB TITLE을 선택할 때마다 PAI 이벤트가 호출되므로, APPLICATION SERVER 측면에서는 비효율적이나 INPUT CHECK 가 현재의 TAB PAGE에서만 수행되는 장점이 있어, 데이터 수정이 필요한 ONLINE 프로그램에 적합하다.

TABSTRIP | 생성

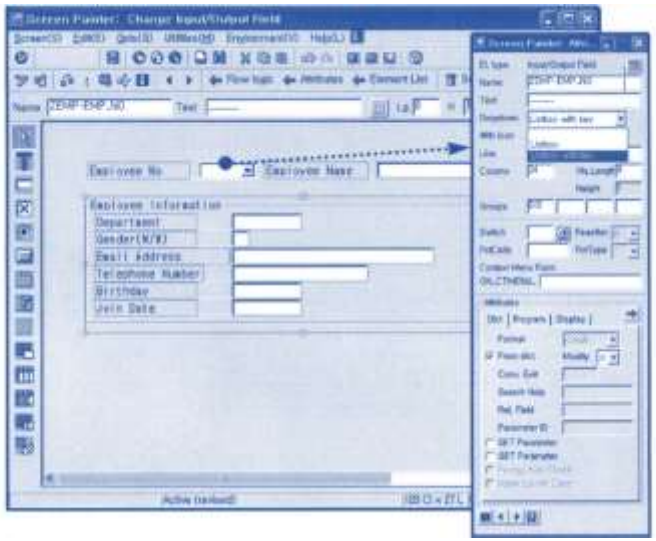
TABSTRIP 생성시 마법사를 이용하는 방법과 직접 설정하는 방법 두 가지가 있다. 마법사를 이용하는 경우 PAGING IN SAPGUI 또는 PAGING IN APPLICATION SERVER 을 선택하여 자동으로 스크립트와 화면을 생성한다. TABSTRIP 마법사가 자동으로 생성하는 스크립트는 복잡하지 않으므로 TABSTRIP 생성시에는 마법사를 이용하는 방법을 권유한다. TABSTRIP은 SUBSCREEN을 포함함으로 PBO/PAI에 SUBSCREEN 스크립트를 추가한다. TABSTRIP이 처음 화면에 조회될 때 TAB PAGE를 지정할 수 있다.

DROPDOWN LIST BOX |

사용자가 입력 값을 DIALOG BOX에서 선택할 수 있도록 하는 INPUT HELP와 유사하게 DROPDOWN LIST BOX는 그림과 같이 사용자가 입력할 수 있는 값을 리스트로 보여준다. LIST BOX 내에서 사용자는 이미 정해진 값 이외의 엔트리는 입력할 수 없다. 즉, 리스트에서 값을 선택하거나 리스트의 킷값을 입력해야 한다. 사용자가 리스트를 선택함과 동시에 PAI 이벤트가 호출된다. 스크린 페인터에서 FUNCTION CODE를 설정하여 LIST BOX에서선택한 이후의 로직을 추가할 수 있다. INPUT HELP 버튼과 LIST BOX를 동시에 설정할 수는 없다. LIST BOX는 KEY와 TEXT 필드로 구성된 라인들로 구성된다. 사용자가 리스트 중 한 라인을 선택하면 리스트의 키 값이 화면에 입력된다. 스크린 필드를 LIST BOX로 보이게 하려면 스크린 페인터에서 DROPDOWN 속성을 설정해야 한다. 스크린의 INPUT/OUTPUT 필드에 LIST BOX를 추가하는 방법은 다음과 같다.

ABAP DICTINARY를 이용한 LIST BOX 생성
POV 이벤트에서 INPUT HELP를 이용한 LIST BOX 생성
PBO 이벤트에서 함수를 이용한 LIST BOX 생성 | 추천하지 않음

DROPDOWN LIST BOX | ABAP DICTIONARY를 이용한 LIST BOX 생성



도메인에 설정된 고정 값 | FIXED VALUE 와 VALUE TABLE, 그리고 SEARCH HELP를 생성하여 TABLE FIELD에 할당하고, 스크린 필드는 TABLE FIELD의 속성을 상속받아 INPUT HELP로 구성한다. 그리고 스크린 페인터에서 DROPDOWN 속성을 설정하면 LIST BOX가 생성된다.

DROPDOWN LIST BOX | POV에서 INPUT HELP를 이용한 LIST BOX 생성

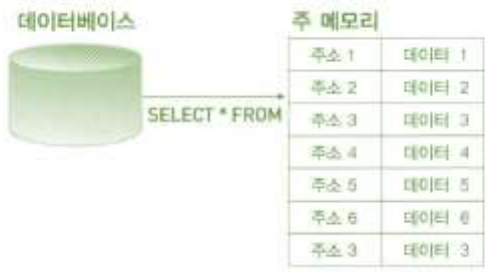
INPUT HELP를 구성하는 방법은 3가지가 있다. 이 중에서 스크린의 POV |

PROCESS ON VALUE-REQUEST 이벤트에서 DIALOG 모듈을 호출하여 INPUT HELP로 구성할 수 있다. 그리고 이와 같이 구성된 INPUT HELP는 스크린 필드에 DROPDOWN 속성이 설정되면, LIST BOX 형태로 조회된다. 이때 주의사항은 INPUT HELP가 2개 이하의 필드만으로 구성되어야 한다는 것이다. 이 두개의 필드는 각각 KEY와 TEXT를 의미하며, 하나일 경우에는 KEY 값만 설정된다. 트랜잭션을 실행하면 제일 먼저 PBO 이벤트가 실행되고, POV 이벤트가 수행되면서 F4_INT_TABLE_VALUE_REQUEST 함수를 호출하여 LIST BOX를 구성하게 된다.

DROPDOWN LIST BOX | PBO에서 함수를 이용한 LIST BOX 생성

PBO 이벤트에서 INPUT HELP를 이용하여 LIST BOX를 생성하는 것이 표준이지만, 실무에서는 PBO에서 LIST BOX를 생성하는 방법도 많이 사용된다.

SCREEN과 STACK |



ABAP 프로그램을 실행하여 데이터를 조회하게 되면, 메모리 영역으로 데이터를 LOAD하려 처리한다. 다음 그림과 같이 ABAP 프로그램 내의 데이터 소스 결과 값 등은 모두 주 메모리 영역에 저장되게 된다. 주 메모리는 우리가 하드웨어적으로 부르는 RAM 메모리 | MAIN MEMORY | 주 기억장치를 의미한다. RAM의 주요 역할은 보조 디스크에 존재하는 데이터를 RAM에 저장하여 CPU의 빠른 성능을 보완하는 것이다. 우리가 윈도우를 실행하게 되면 윈도우를 실행하는 데 필요한 프로그램들을 하드디스크에서 읽어와 주 기억장치에 LOAD하게 된다. 예를 들어, MS 엑셀 프로그램을 실행하면 프로그램을 실행하기 위한 정보들을 하드디스크에서 읽어와 주 메모리에 저장한다. 사용자가 MS엑셀에서 데이터를 입력하고, 저장하는 것은 주 메모리에서 작업하기 때문에 속도가 빠르며, CPU는 느린 하드디스크가 아닌 훨씬 빠르게 접근할 수 있는 주 메모리를 이용해 윈도우를 조작하게 된다. 여기에서 주의해야할 점은 SELECT 구문을 이용해 데이터를 읽어오는 것은 보조 디스크에서 읽기 때문에 속도 | MS 단위가 느리다는 것이다. 반면에 인터널 테이블은 주 기억장치에 저장되기 때문에 인터널 테이블을 READ 하는 속도 | MS 단위 는 SELECT에 비해 비교할 수 없을 정도로 매우 빠르다. 그래서 SELECT 구문은 가급적 JOIN 구문을 이용해 한 번에 인터널 테이블에 저장하는 것이 효율적이라는 것을 알 수 있다.

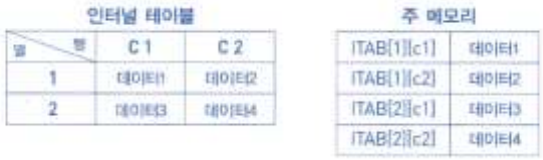
```
DATA : L_1 TYPE C.
C = 'A'.
```

ABAP 프로그램 내에서 다음 구문과 같이 변수를 선언하면, 주 메모리에는 다음 그림과 같이 주소 영역과 데이터 영역으로 구분되어 저장된다.

ABAP 프로그램에서 SELECT 구문으로 인터널 테이블에 데이터를 저장하면, 보조 기억 장치인 하드디스크에 설치된 데이터베이스의 데이터들이 주 기억 장치에 인터널 테이블 형태로 저장된다. 역시, 주소 영역과 데이터 영역으로 구분된다.

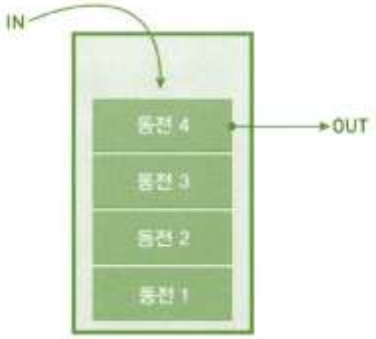
```
SELECT * FROM TABLE INTO ITAB.
```

위 구문에서 인터널 테이블 ITAB에 저장된 데이터는 주 메모리에 배열 순번으로 저장된다.



그러나 CALL SCREEN 100과 같이 모듈 풀 프로그램에서 스크린을 호출하는 명령어는 STACK이라는 메모리 공간에 순서를 저장하게 된다. STACK 메모리는 하드웨어적으로 존재하는 것이 아닌, MAIN MEMORY의 일정 영역을 지정하여 사용한다. 주소를 이용하여 호출하지 않기 때문에 당연히 데이터를 읽는 속도는 빠르게 된다.

택시 기사 아저씨들이 동전을 관리할 때 사용하는 것을 코인 스택이라 하는데 이것이 바로 스택의 개념을 쉽게 설명하는 예가 된다. 제일 먼저 들어간 동전은 맨 아래에 있고, 마지막 스택에 들어간 동전은 제일 먼저 뽑아 사용할 수 있다. 이는 컴퓨터 일반 이론에서 LIFO | LAST IN FIRST OUT을 의미한다.



이와 같은 구조로 ABAP 프로그램에서도 STACK을 이용하게 되는데, 대표적인 경우가 CALL SCREEN 구문을 사용할 때이다. CALL SCREEN 구문을 이용하여 스크린을 호출하면, CALL SCRREEN 구문을 만나는 즉시 호출한 스크린으로 이동한다. 그리고 현재 스크린은 종료되지 않고 비활성화된 상태로 존재한다. 아래의 그림에서 설명하는 것과 가이 ABAP 프로그램에서 CALL SCREEN 명령어로 3개의 스크린을 호출하게 되면, 스크린 순번대로 STACK 메모리에 저장된다.

사용자가 화면에서 BACK 버튼을 선택하면, LEAVE TO SCREEN 0 구문이 실행되며 STCK 메모리에서 하나씩 꺼내 온다. LEAVE TO SCREEN 0 명령어는 자신을 호출한 스크린으로 되돌아가는 기능을 수행한다. LEAVE TO SCREEN 100과 같이 스크린 번호를 명시적으로 기술하면 현재 스크린은 STACK 에 저장하지 않고, 바로 스크린 100으로 이동한다. 이를 아래의 그림에서 설명한다.

프로그램을 실행하면 최초로 조회되는 스크린 100이 있다고 가정하자. 현재 스크린이 100번이며, CALL SCREEN 200 구문을 수행한 후 LEAVE TO SCREEN 0 구문을 만나면 100번 화면으로 돌아가지만, LEAVE TO SCREEN 200 구문으로 실행된 후 LEAVE TO SCREEN 0 구문을 만나게 되면 프로그램을 종료하게 된다. 이와 같이 스크린이 여러 개 존재하는 모듈 풀 프로그램에서는 CALL 구문과 LEAVE TO SCREEN 구문을 명확히 정의하고 사용할 수 있어야 한다.