Questions d’examen

Qu'est-ce qu'un mandant ?

Le mandant permet de matérialiser une séparation entre deux ERP sur le même système SAP. En d’autres termes, une séparation du système SAP en deux systèmes dont les données sont différentes. Effectivement, lorsqu’on se connecte au système SAP, il faut préciser le mandant avec lequel on se connecte.

Cette séparation est implémentée simplement par un champ supplémentaire dans les tables qui ne peuvent pas être communes. Par exemple, la table de clients possède un champ MANDT de type CLNT (Client = mandant != custommer) et donc, lorsqu’on visualise les données de la table, on ne verra que les données dont on est « propriétaire » (c'est-à-dire, dont le mandant est le même que le mandant connecté).

Autre vision des choses : le mandant est un nombre de sur trois chiffres utilisé pour sélectionner les tuples d’une table dont le mandant fait partie de la clef. Le rôle du mandant est donc de séparer les choses… on ne peut pas accéder aux données d’un autre mandant.

Les éléments multi-mandants sont des éléments qui sont communs à tous les mandants. Typiquement : le repository contenant tous les programmes, ainsi que le data dictionnary contenant les structures de tables… De plus, certaines tables telles que la table des mandants (logique…), la table des langues… sont également des éléments multi-mandants.

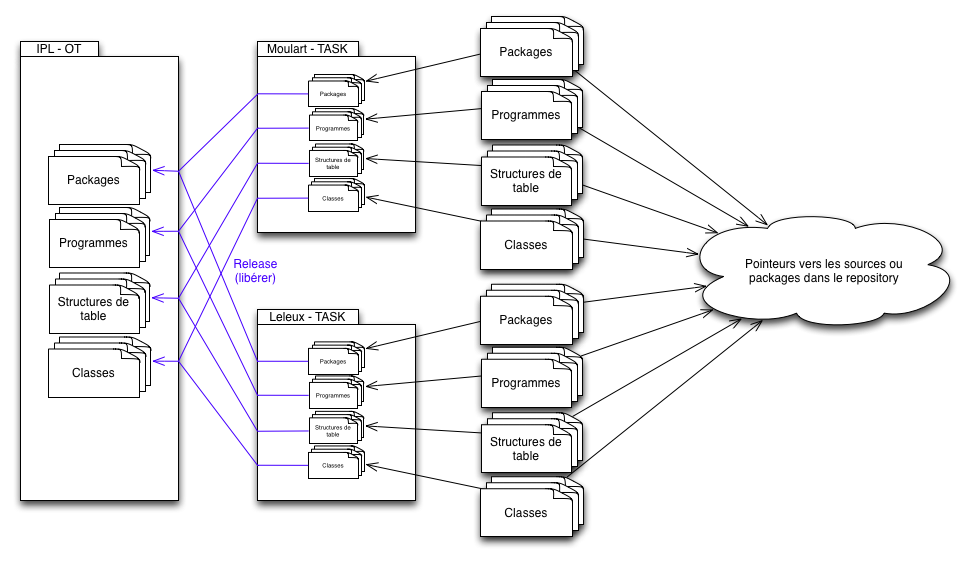
Les éléments mono-mandants sont des éléments qui sont propres à chaque mandant. Typiquement : les utilisateurs, les données de l’ERP telles que les données client, fournisseurs… les codes TVA, les transactions…

Il y a des mandants « de base » tels que le mandant 000 qui sert de mandant de référence contenant toutes les données de base qui peuvent être récupérées si besoin est, et le mandant 066 qui sert à SAP pour faire du monitoring préventif.

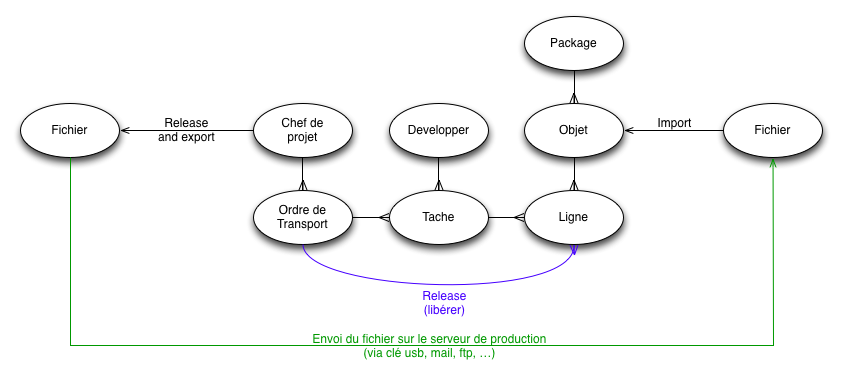
En résumé, voici les trois avantages qu’apporte l’utilisation de mandants :

1. Il permet de protéger les données d’un utilisateur, si bien qu’un autre ne puisse ni lire, ni modifier ces données (isolement des données)
2. Plusieurs utilisateurs peuvent travailler en parallèle sans s’interférer les uns avec les autres (isolation fonctionnelle)
3. La configuration d’un mandant peut être modifiée à tout moment (indépendance des mandants)

Expliquez les notions de tâche et d'ordre de transport, ainsi que les liens entre ces éléments, le développeur et les objets développés (Bachman)



Chaque développeur se voit attribuer une ou des "tâches" une sorte de dossier dans lequel il met des "post-it" qui sont des pointeurs vers les objets du repository qu'ils créent/utilisent/modifient. Une fois sa tache terminée, ses programmes écrits, … le développeur peut libérer sa tâche et attacher ainsi ses "post-it" à l'ordre de transport auquel était attaché la tâche depuis sa création. Une fois que tous les développeurs ont libéré toutes leurs tâches, le chef de projet peut libérer et exporter l'ordre de transport, ce qui a pour effet de récupérer les sources effectives pointées par les "post-it" et de les placer dans un fichier, qu'il transfère alors sur le serveur de production (via clé usb, mail, …) pour qu'il y soit importé, avec tous les programmes/packages/fonction/structures de table… nécessaires.



Quelles sont les principales étapes pour transporter un programme (ou tout autre objet du repository) d'un système de développement vers un système de production ?

Cfr. Ci-dessus : création de l'ordre de transport, création des tâches et rattachement de celles-ci à l'OT, programmation, libération des tâches, libération et export de l'OT, import du fichier généré sur le système de production.

Comment ce mécanisme est-il mis en place sans SAP au niveau d'une équipe de développement?

Le chef de projet, via la transaction SE01, va pouvoir visualiser l'état d'avancée et les différentes tâches libérées ou non pour enfin pouvoir exporter et libérer l'ordre de transport. La visualisation se fait sous forme d'arbre hiérarchique, comme l'explorateur de Windows.

Enumérez les différents outils de l'ABAP workbench en précisant le rôle de chacun

Voir Page 24

* ABAP Editor for writing and editing reports, module pools, includes and subroutine pools (SE38)
* ABAP Dictionary for processing database table definitions and retrieving global types (SE11)
* Menu Painter for designing the user interface (menu bar, standard toolbar, application toolbar, function key assignment) (SE41)
* Screen Painter for designing screens and flow logic (SE51)
* Function Builder for function modules (SE37)
* Class Builder for ABAP Objects classes and interfaces (SE24)
* The Object Navigator (transaction SE80) provides a single integrated interface into these various tools.

Décrivez les principaux objets gérés au niveau du data dictionary en mettant en évidence les différentes relations qui existent entre eux. Indiquez les objets mono-mandants et multi-mandants.

The ABAP Dictionary contains all metadata about the data in the SAP system. It is closely linked with the ABAP Workbench in that any reference to data (e.g. a table, view, data type, etc.) will be obtained from the dictionary. Developers use the ABAP Dictionary transactions (directly or through the SE80 Object Navigator inside the ABAP Workbench) to display and maintain this metadata.

When a dictionary object is changed, a program that references the changed object will automatically reference the new version the next time the program runs. Because ABAP is interpreted, it is not necessary to recompile programs that reference changed dictionary objects.

A brief description of the most important types of dictionary objects follows:

* ***Tables*** are ***data containers*** *that exist in the underlying relational database*. In the majority of cases there is a 1-to-1 relationship between the definition of a table in the ABAP Dictionary and the definition of that same table in the database (same name, same columns). These tables are known as "**transparent**". There are two types of **non-transparent tables**: "pooled" tables exist as independent entities in the ABAP Dictionary but they are grouped together in large physical tables ("pools") at the database level. Pooled tables are often small tables holding for example configuration data. "Clustered" tables are physically grouped in "clusters" based on their primary keys; for instance, assume that a clustered table *H* contains "header" data about sales invoices, whereas another clustered table *D* holds the invoice line items. Each row of H would then be physically grouped with the related rows from D inside a "cluster table" in the database. This type of clustering, which is designed to improve performance, also exists as native functionality in some, though not all, relational database systems.
* ***Structures*** are ***complex data types consisting of multiple fields*** (comparable to *struct* in C/C++).
* ***Data elements*** provide the ***semantic content for a table or structure field***. For example, dozens of tables and structures might contain a field giving the price (of a finished product, raw material, resource, ...). All these fields could have the same data element "PRICE".
* ***Domains*** define *the* ***structural characteristics of a data element***. For example, the data element PRICE could have an assigned domain that defines the price as a numeric field with two decimals. Domains can also carry semantic content in providing a list of possible values. For example, a domain "BOOLEAN" could define a field of type "character" with length 1 and case-insensitive, but would also restrict the possible values to "T" (true) or "F" (false).
* ***Search helps*** (successors to the now obsolete "matchcodes") *provide* ***advanced search strategies*** *when a user wants to see the possible values for a data field*. The ABAP runtime provides implicit assistance (by listing all values for the field, e.g. all existing customer numbers) but search helps can be used to refine this functionality, e.g. by providing customer searches by geographical location, credit rating, etc.ary ou

Note: F1 -> technical info, permet de trouver le field name et le data element d’un objet.(page 51)(ou alors double clic pour parcourir l’ABAP Dictionnary)



Les objets de base du dictionnaire des données sont les tables, les domaines, les data éléments, les champs et leurs relations entre tables.

Un **domaine** est un objet central pour décrire les attributs d'un objet commercial. Un domaine décrit un ensemble de valeurs pour un champ. Cet ensemble de valeur est inscrit comme attribut formel spécifique, par exemple : longueur, maximum et minimum,…. Ce sont donc les informations techniques.  
=>Un domaine peut-être défini sans référence aux autres objets du dictionnaire.

Un **data element** est un domaine sémantique. Il donne une description précise de la fonction du domaine dans un contexte commercial spécifique pour les champs qui en dépendent. C'est une information logique, une description fonctionnelle.  
Un data élément présuppose l'existence d'un domaine.  
C'est au niveau du data élément qu'on documente les champs.

Un champ est dépendant d'une table et présuppose l'existence d'un data élément.

🛈Quand on parle de *domaine à deux niveaux*, on parle du *domaine et* du *data élément*.

Qu'est-ce qu'une "classe de développement" et quel est son intérêt ?

Classe de développement **= Package**.

Définition :  
Un package est unité de rangement spécifiant si les objets qu’il contient sont transportables ou non. Un package est lui même un objet du repository.  
Il regroupe un ensemble d’objets interagissant les uns avec les autres dans le but d’accomplir un objectif du programmeur(*éventuellement une tâche* *si* package *transportable* par exemple)

Une classe développement est **le plus haut noeud d’une hiérarchie d’objets** ( écrans, fonctions, tables, …). Grâce à « Object View », nous avons la liste de tous les objets présents dans une classe de développement. Cette liste est

maintenue à jour automatiquement par le système. Nous pouvons à partir de la liste créer, afficher et modifier tous les objets.

Remarque sur les packages :  
La structuration de ses sources en packages permet de hiérarchiser ses sources (l'ensemble des objets SAP) mais aussi de gérer plus facilement les transports en les réalisant par package.

C'est une pratique courante dans le monde Java et .net mais encore pas assez développée sur SAP. L'objectif est d'avoir un modèle hiérarchisé de l'ensemble de ses sources maisons (techniques et spécifiques métier). Par exemple il est possible de créer un package ZSOCIETE (avec le nom de la société) et dedans des packages FRAMEWORK, FI, SD, MM, etc... et ainsi de suite.

Quelle est la différence entre "classe de développement" et "ordre de transport"

Un ordre de transport est défini par le chef le projet, les packages sont définis par les développeur.

Une ordre de transport contient donc plusieurs packages (du moins les pointeurs vers ceux-ci).

Ce n’est pas parce qu’on libère un ordre de transport que tout le package sera mis en production. Uniquement les objets du package qui ont été créés ou modifiés seront mis en production.

Package != request change(ordre de transport). -> voir page 42

Quelle est la différence entre une table interne et une table du Data Dictionary?

Références utiles :  
 ⇒ voir page 75(notion de table type).

Exemple de table interne :   
ZZ\_02\_INVOICE\_DISPLAY et p79,80,81,...  
  
----------------------------------------

Première chose à noter: les screens ne fonctionnent qu’avec des éléments de l’ABAP dictionnary.

supposition

Donc si on utilise des PARAMETERS ou des tables internes, il faudrait faire des move corresponding de la table interne vers les structures de l’abap dictionnary. -> chiant

----------------------------------------

**What are internal tables?**

Internal Tables are *local tables within a program containing a series of lines having same data type*. ABAP Open SQL allows single field, range of fields, entire database table or view into an Internal table.  
  
In technical terms Internal table is a *dynamic sequential dataset in which all records have the same data structure and a key.*  
**Time to live ?**Only during the runtime of the program.   
  
**What for ?  
-** Used for fetching large volume of data from the database, storing in ABAP working memory line-by-line and processing within a program. **-** Used to perform table calculations on subsets of database tables and for re-organising the contents of database tables according to the user’s needs.

**How does it work ?**Although Internal tables are declared with the other data objects, at runtime they *behave as dynamic objects* (i.e.) no need to specify the size of the object but only the length of a row in internal table is fixed. The number of rows is determined dynamically at runtime with the fixed structure.

**How to use it ?**If we declare structure with TYPE then it is only a *definition*.  
If we declare structure with DATA then it creates objet and contains *one record*.  
If we declare structure with TABLE OF then it creates *internal table of initial size*.

Faites un parallèle entre ces deux technique en matière de programmation ABAP.

Il est plus commode d’employer des tables prédéfinies dans l’ABAP Dictionary bien que les tables internes soient utilisées pour les raisons exposées ci-dessus càd pour des raisons de performance d’une part lorsque l’on désire traiter des ensembles conséquents de données et d’autre part pour créer des sortes de tables virtuelles que l’on pourra manipuler facilement afin de fournir les structures de données qui correspondent le mieux aux besoins de l’utilisateur.  
Sert également de table temporaire en raison de son scope.

A quoi servent les search-help?

Lorsqu’un écran de sélection est présenté à l’utilisateur, celui-ci peut appuyer sur la touche F4 ou cliquer sur l’icône à droite du champ pour voir apparaitre un écran qui lui offre la possibilité de sélectionner une valeur pour le champ. Il s’agit de la fonction « search-help ». S’il y a trop de possibilités pour le champ, on peut ajouter des contraintes pour en restreindre le nombre. Il y a également des informations supplémentaires sur les différents champs qui sont affichés.

Les search-help sont définis dans l’ABAP Dictionary. La transaction se11 (ABAP Dictionary) permet d’en créer et de les tester sans devoir les attacher à un programme existant.

Quels sont les différents types de dialogues utilisateurs prévus en ABAP et quels en sont leur usages ?

Il y a deux types de dialogues utilisateurs :

* **Écrans de sélection** : établir des contraintes pour sélectionner une plage de données.
* **Écrans (dynpros)**
* **Listes** : permet de faire du reporting éventuellement interactif grâce à l’évènement AT LINE-SELECTION.
* **Messages** : permet d’afficher des messages à l’utilisateur. Dans la barre de statut, ou dans une boîte de dialogue. Ils peuvent avoir deux fonctions : informer l’utilisateur (erreur ou non), terminer l’application.

Quelles sont les différences en terme de programmation entre "Ecrans de sélection" et "ecrans" (dynpros)

Les écrans de sélection sont créés grâce à deux instructions : SELECT-OPTIONS et PARAMETERS.

Le but d'un écran de sélection, comme son nom l'indique, est de permettre de définir, via une interface, des critères de sélection pour les requêtes qui seront effectuées durant le traitement de sélection des données.

Un écran de sélection est principalement utilisé dans les programmes qui doivent afficher un rapport en fin de traitement.

Avec un écran de sélection, on peut assez difficilement faire le layout, les positions des zones sont prédéfinies, il n'y a pas de "screen painter" pour nous aider.

Les écrans (dynpros) sont créés grâce au screen painter et sont appelés grâce à la méthode CALL SCREEN ou SET SCREEN.

Quant aux écrans, ils se déclarent et se construisent depuis l'éditeur d'écran accessible via SE38 (ABAP Editor), SE80 (Object Navigator) ou directement via SE51 (Screen Painter). Les utilisations d'un écran sont diverses et variées, ça peut être une utilisation ponctuelle en l'utilisant en tant que fenêtre modale par exemple.

Comme un écran doit être appelé via un CALL SCREEN, il ne peut être utilisé pour définir des critères.

Quels sont les différents éléments à préciser lors de la définition d'un module de fonction ?

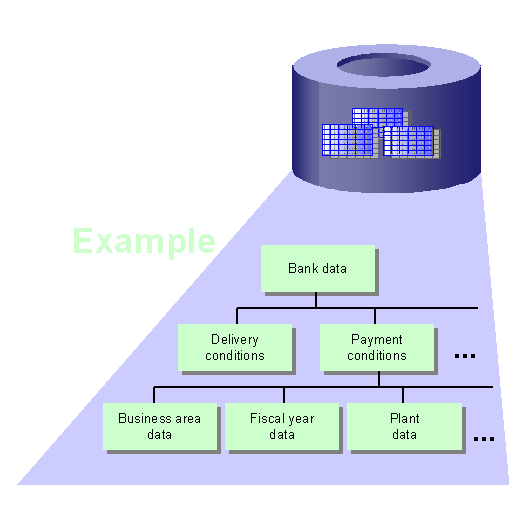
Un module de fonction est une fonction qui peut être appelée par n’importe quel programme ABAP. Le module de fonction peut être contenu dans un « groupe de fonctions ». Ce groupe peut contenir des déclarations globales de données et des sous-routines qui sont disponibles pour tous les modules de fonctions du groupe. (p. 172)

* **Les paramètres d’administration**: des informations telles que la classe de développement, le responsable du module, une description du module, le groupe de fonctions, Si la fonction est remote callable[[1]](#footnote-1) ou pas (REMOTE FUNCTION CALL)…
* **Les paramètres import** : paramètres reçus par le module de fonction
* **Les paramètres export** : paramètres renvoyés par le module de fonction vers la fonction appelante
* **Les paramètres changing**: paramètres reçus par le module de fonction, éventuellement modifiés, et ensuite renvoyés au programme appelant
* **Les tables** : les tables internes qui peuvent être importées ou exportées. Les tables peuvent être modifiées et renvoyées vers le programme appelant.
* **Les exceptions** qui peuvent être lancées par le module de fonction (RAISE <excp>)
* **La documentation**: des informations sur l’interface et les exceptions, un exemple d’utilisation, des notes…

Qu'est-ce qu'une "logical database"? A quoi servent-elles ?

A logical database provides a particular view of database tables in the R/3 System. It is always worth using logical databases if the structure of the data that you want to read corresponds to a view available through a logical database.

The data structure in a logical database is hierarchical. Many tables in the R/3 System are linked to each other using foreign key relationships. Some of these dependencies form tree-like hierarchical structures. Logical databases read data from database tables that are part of these structures.



Quelles sont les différentes techniques de modularisation proposées par SAP?

* Events : START-OF-SELECTON, AT SELECTION-SCREEN, END-OF-SELECTION…
* Routines : fonctions qui sont placées en fin de programme et qui peuvent être appelées au sein d’une application. Elles sont déclarées par FORM … ENDFORM.
* Modules de fonction
* Méthodes (ABAP Objects)

Quand utilisera-t-on l'une plutôt que l'autre ?

Les évènements sont utilisés au sein d’une application quand il s’agit d’exécuter du code à certains moments du déroulement de l’application. (START-OF-SELECTION est l’évènement par défaut si on ne met rien)

Les routines sont utilisées quand il faut externaliser du code au sein d’une même application. SAP incite à abandonner l’usage des routines au profit des modules de fonction et des méthodes.

Les modules de fonction permettent de partager des méthodes pour plusieurs applications. Elles permettent également de partager du code pour un groupe de méthodes partagées (groupes de fonctions). De plus, les modules de fonctions peuvent être appelées à distance par d’autres systèmes SAP (si le module le permet : RFC). La programmation de SAP Business Workflow utilise les transactions et fonctions existantes pour réaliser des business process utiles pour l’entreprise et qui ne sont pas prévues par défaut sur le système. Il s’agit donc de mettre bout à bout des transactions et modules de fonctions existantes (on utilise pour cela Workflow Builder).

Les méthodes sont utilisées dans le cadre de la programmation orientée objet avec ABAP Objects.

Quels sont les avantages et inconvénients de "l'open SQL" en programmation ABAP ?

A key component of the ABAP runtime system is the Database Interface, which turns database-independent ABAP statements ("Open SQL") into statements understood by the underlying DBMS ("Native SQL"). The database interface handles all the communication with the relational database on behalf of ABAP programs; it also contains extra features such as buffering of tables and frequently accessed data in the local memory of the application server.

Open SQL consists of a set of ABAP statements that perform operations on the central database in the R/3 System. The results of the operations and any error messages are independent of the database system in use. Open SQL thus provides a uniform syntax and semantics for all of the database systems supported by SAP. ABAP programs that only use Open SQL statements will work in any R/3 System, regardless of the database system in use. Open SQL statements can only work with database tables that have been created in the ABAP Dictionary.

Avantages

* **Simplicity** : Reduced statement and source code complexity
* **Portability**: if the companies decide to change the database, we don’t need to change our ABAP programs.
* **Buffering data**: While running Open SQL code all the database data will be buffered in Application Server. So in the same session next time the queries don’t need to fetch the data from database server. Instead of that data will provide from the buffering data. It will speed up the database access time.
* **Automatic Client handling**: The client filed will be automatically populated by the database interface.
* **Syntax check** is performed unlike the Native SQL (no syntax check between EXEC and ENDEXEC)

Inconvénients

* Tables must be declared in the ABAP Dictionary to be accessed
* You cannot use some of the special features supported by the database-specific SQL (e.g. passing hints to Oracle optimizer)

Comment SAP assure-t-il la portabilité de ses applications ?

SAP fournit des interfaces dans tous les domaines (p. 22) :

* **Operating system interface** : convertit tout appel système SAP en appel système propre à l’OS concerné
* **Presentation interface** : fait en sorte que la présentation et l’interaction de l’utilisateur avec R/3 soient équivalents sans tenir compte du type de GUI utilisé. Pour chaque session utilisateur, un processus (SAPGUI) est créé, et utilise les possibilités fournies par le logiciel de maintenance de fenêtres (Window Management Software). De cette manière, ce processus gère une grande partie de l’interface graphique de R/3.
* **Database interface** : supporte plusieurs systèmes de base de données relationnelles. Convertit des requêtes SQL SAP en requêtes SQL propres au DBMS utilisé (Optimisation et vérification de la syntaxe) grâce à l’Open SQL.
* **Communication interface**: fournit une interface pour des communications externes telles que : SMS, FAX, mail, http…

Le but de ces interfaces est d’obtenir une portabilité optimale grâce à l’indépendance point de vue présentation graphique (presentation interface), système de base de données (database interface) et système d’exploitation (operating system interface).

SAP basis ?

Quels sont les différents éléments constitutifs d'un programme ? Quels sont leurs rôles ?

* Sélection screens
* Event blocks
* Local types and field definitions
* Modules
* Screens (dynpro)
  + PBO
  + PAI

Expliquez le mécanisme des "dynpros"

Dynpro est l'abréviation de DYNamisches PROgramm (programme dynamique). Une dynpro se compose d´une définition d'écran et de modules.

L'écran définit la position et l'agencement des éléments d´interface graphique de base tels que des cases à cocher, zones de texte, boutons radio, cadres ...). De plus une Dynpro peut contenir d´autres Dynpros à l´aide de conteneurs.

Les modules contiennent les processus de commandes de l'écran. Il existe principalement deux types de modules :

* PBO (process before output) : processus avant l´affichage de l’écran (SET PF\_STATUS, SET TITLEBAR…)
* PAI (process after input) : processus après l’affichage de l’écran et après la saisie de l’utilisateur.

Ces modules peuvent être écrits dans un nouvel include ou dans la source du programme.

Qu'est-ce qu'un "module-pool"

Reports follow a relatively simple programming model whereby a user optionally enters a set of parameters (e.g. a selection over a subset of data) and the program then uses the input parameters to produce a report in the form of an interactive list. The term "report" can be somewhat misleading in that reports can also be designed to modify data; the reason why these programs are called reports is the "list-oriented" nature of the output they produce.

Module pools define more complex patterns of user interaction using a collection of screens. The term “screen” refers to the actual, physical image that the user sees. Each screen also has a “flow logic”, which refers to the ABAP code implicitly invoked by the screens. Each screen has its own flow logic, which is divided into a "PBO" (Process Before Output) and "PAI" (Process After Input) section. In SAP documentation the term “dynpro” (dynamic program) refers to the combination of the screen and its flow logic. Module pool programs are excuted using TCodes (Transaction Codes).

Qu'est-ce qu'une transaction ? Quels en sont les principaux composants ?

A transaction in SAP terminology is the execution of a program. The normal way of executing ABAP code in the SAP system is by entering a transaction code (for instance, VA01 is the transaction code for "Create Sales Order"). Transactions can be called via system-defined or user-specific, role-based menus. They can also be started by entering the transaction code directly into a command field, which is present in every SAP screen. Transactions can also be invoked programmatically by means of the ABAP statements CALL TRANSACTION and LEAVE TO TRANSACTION.

The term "transaction" must not be misunderstood here; in the context just described, a transaction simply means calling and executing an ABAP program. In application programming, "transaction" often refers to an indivisible operation on data, which is either committed as a whole or undone (rolled back) as a whole. This concept exists in SAP and is called as a LUW (Logical Unit of Work). In the course of one transaction (program execution), there can be different LUWs. Transaction for ABAP Workbench could be invoked using transaction code SE80 to work on all ABAP related activities

**Quels en sont les principaux composants ?**

The non-executable program types are:

* INCLUDE modules
* Subroutine pools
* Function groups
* Object classes
* Interfaces
* Type pools

An INCLUDE module gets included at generation time into the calling unit; it is often used to subdivide very large programs. Subroutine pools contain ABAP subroutines (blocks of code enclosed by FORM/ENDFORM statements and invoked with PERFORM). Function groups are libraries of self-contained function modules (enclosed by FUNCTION/ENDFUNCTION and invoked with CALL FUNCTION). Object classes and interfaces are similar to Java classes and interfaces; the first define a set of methods and attributes, the second contain "empty" method definitions, for which any class implementing the interface must provide explicit code. Type pools define collections of data types and constants

Lorsque l'on crée une transaction dans SAP, on associe les éléments suivants :

- un code à 4 caractères

- un programme ABAP

- un numéro d'écran

- un libellé

Donc une transaction n'est qu'une sorte de raccourci permettant de lancer un programme, avec un premier écran de démarage sans passer pour cela par une transaction standard SAP (genre SE38 (Abap Editor)) pour exécuter ce programme.

Ce code transaction est pris en compte lorsqu'il est passé comme paramètre dans l'OK-Code (soit manuellement par l'utilisateur soit par la programmation).

En ce qui concerne les Dynpros, il existe plusieurs moyens pour passer d'un écran à l'autre.   
 Premièrement, lorsque l'on crée un écran dans le screen-painter, dans les attributs de l'écran, on définit le numéro d'écran suivant par défaut. De plus "l'unité de traitement" en terme de temps ou d'événement est l'écran lui-même, c'est pourquoi les initialisations, vérifications et traitements, se font écran par écran.   
On ne peut donc pas faire de traitement à la sortie d'un champs, ou à l'entrée d'un champs, etc... les traitements se font avant l'affichage de l'écran (PBO) et après remplissage de l'écran par l'utilisateur ou par le programme (PAI). C'est donc après avoir vérifié la validité des informations introduites que l'on passe à l'écran suivant... on spécifiera cet écran dans le PAI (via l'instruction CALL SCREEN XXX ) si l'on ne désire pas brancher automatiquement sur l'écran par défaut.

Enfin, dès que l'on actionne un bouton, que l'on utilise un menu ou que l'on appuie sur ENTER, ON PASSE A

L'ECRAN SUIVANT!!!

chaque push button -> doivent avoir un FCODE(identifiant pour le bouton) qui correspond à un OK\_CODE, quand la personne click dessus il suffit dès lors de tester la valeur de l’ok\_code avec un switch\_case comme suit:

case ok\_code.

when 'NEXT'. set SCREEN 200.”Next = Fcode du pushbutton établi grace au screen painter.

attention : ne pas oublier de rajouter Ok\_code like sy-ucomm dans le data du programme ainsi que la reference dans l’element list du screen sur lequel on travail.

+set pf-status à main pour gerer les bouton de la standard tool bar(back, exit, …)

exemple de PBO -> clear l’ok code !

exemple de PAI -> exit if asked(apres que le mec click sur qqch on check si il voulait pas quitter :))

Expliquez les différentes étapes d'une exécution de transaction simple.

Entre OK code et écran d’application

voir p. 144,145,146,...

call screen 100?

Expliquez le fonctionnement de l'instruction CHECK

Check permet de vérifier la validité d’une condition lorsqu’on parcourt un certain nombre de tuples. Si la condition n’est pas vérifiée, on passe au tuple suivant. Il peut être utilisé dans un SELECT ou dans toute autre boucle (LOOP, DO TIMES…). Elle peut également être utilisée dans un « event block », dans ce cas, le check arrêtera l’exécution des instructions du bloc.

Expliquez le fonctionnement de l'instruction SELECT-OPTIONS

Déclare une table interne (de sélection) qui est liée à deux « inputs » d’un écran de sélection. Ces deux inputs représentent un range de valeurs, une sélection unique, ou une liste de sélections uniques. La table contient plusieurs colonnes :

* SIGN : I(Include), E(Exclude).
* OPTION :
  + si HIGH est vide, c’est à dire qu’il représente une sélection unique, on peut utiliser les options :
    - EQ (Equal)
    - NE (Not Equal)
    - GT (Greater Than)
    - GE (Greater Than or Equal)
    - LT (Less Than)
    - LE (Less Than or Equal)
    - CP (Contains Pattern)
    - NP (Not Contains Pattern)
  + si HIGH n’est pas vide, c'est-à-dire qu’il spécifie un range de valeurs, on peut utiliser les options :
    - BT (Between)
    - NB (Not Between)
* LOW : si HIGH est vide, LOW représente une sélection unique. Si HIGH n’est pas vide, LOW représente la borde inférieure d’un range de valeurs.
* HIGH : contient la borne supérieure d’un range de valeurs

Chaque tuple de cette table de sélection représente donc un range de valeurs ou une sélection unique.

Lorsque l’utilisateur va entrer un range de valeurs, la table interne va être remplie avec les données de l’utilisateur. Il suffit ensuite de faire des tests sur cette table. Par exemple :

DATA WA TYPE SPFLI.

SELECT-OPTIONS S\_CARRID FOR WA-CARRID.

IF 'LH' IN S\_CARRID.

WRITE 'LH was selected'.

ENDIF.

Ou encore :

DATA WA TYPE SPFLI.

SELECT-OPTIONS S\_CARRID FOR WA-CARRID.

SELECT \* FROM SPFLI WHERE SPFLI-CARRID IN S\_CARRID

WRITE SPFLI-CARRID.

SKIP.

ENDSELECT.

L’instruction SELECT-OPTIONS s’utilise de la manière suivante :

SELECT-OPTIONS <internTableName> FOR <field>.

Elle peut prendre plusieurs paramètres :

* NO-DISPLAY
* OBLIGATORY
* NO-INTERVALS
* VISIBLE LENGTH <int>
* …

Expliquez le fonctionnement de l'instruction PARAMETERS.

Déclare une variable qui est liée à un « input » d’un écran de sélection.

L’instruction PARAMETERS s’utilise de la manière suivante :

PARAMETERS <nomDeLaVariable>.

Elle peut prendre plusieurs paramètres :

* NO-DISPLAY
* OBLIGATORY
* AS CHECKBOX [DEFAULT <valeur>]
* VISIBLE LENGTH <int>
* DEFAULT <valeur>
* …

Remarque : le domaine d’une checkbox est les deux caractères ‘’ et ‘X’.

Expliquez les différentes possibilités en matière de passage de paramètres à une routine.

Voici un exemple d’une routine :

DATA: valeur1 TYPE i VALUE 10, valeur2 TYPE i VALUE 12.

DATA: monResultat TYPE i.

\* appel à la sous-routine

PERFORM addition

USING valeur1 valeur2

CHANGING monResultat.

\* la sous-routine

FORM addition

USING

var1 TYPE i

var2 TYPE i

CHANGING

Result TYPE i.

result = var1 + var2.

ENDFORM.

USING permet de faire un passage par valeur, on envoie des valeurs qui sont récupérées dans la routine et qui y sont utilisées mais pas envoyées.

CHANGING permet de faire un passage par référence. En envoie la référence vers une variable que la routine peut modifier.

Quels sont les différents types numériques prédéfinis ?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Type** | **Signification** | **Longueur initiale** | **Longueur valide** | **Valeur initiale** | **Utilisation** |
| **Numeric** | I | Integer | 4 | 4 | 0 | compteurs, nombre d’objets, index… |
| F | Floating point number | 8 | 8 | 0 | nombres très grands ou les erreurs d’arrondis ne sont pas importantes |
| P | Packed number | 8 | 1 – 16 | 0 | distances, poids, montants… |
| **Character** | C | Text field | 1 | 1 – 65535 | ‘ … ’ | chaines de caractères alphanumériques |
| D | Date field | 8 | 8 | ‘00000000’ | dates (YYYYMMDD) |
| N | Numeric text field | 1 | 1 – 65535 | ‘0 … 0’ | caractères numériques |
| T | Time field | 6 | 6 | ‘000000’ | heures (HMMSS) |

Expliquez le fonctionnement de l'instruction COLLECT.

L’instruction COLLECT permet d’ajouter des tuples dans une table interne sans faire de doublons, mais en additionnant les champs numériques des doublons. Lorsqu’on utilise l’instruction pour ajouter un tuple dans la table interne, le système va d’abord vérifier qu’il n’existe pas un tuple dans la table qui correspond à la clef primaire. S’il n’y a pas de tuple, il l’ajoute dans la table. Sinon, il additionne les champs numériques dans les champs correspondants du tuple dans la table interne. Exemple :

DATA: BEGIN OF line,

col1(3) TYPE C,

col2(2) TYPE N,

col3 TYPE I,

END OF line.

DATA itab LIKE SORTED TABLE OF line WITH NON-UNIQUE KEY col1 col2.

line-col1 = ‘india’. Line-col2 = ’20′. line-col3 = 6.

COLLECT line INTO itab.

line-col1 = ‘usa’. Line-col2 = ’10′. line-col3 = 5.

COLLECT line INTO itab.

line-col1 = ‘india’. Line-col2 = ’20′. line-col3 = 7.

COLLECT line INTO itab.

LOOP AT itab INTO line.

WRITE: / line-col1, line-col2, line-col3.

ENDLOOP.

Ceci va afficher : india 20 13 et usa 10 5

If you specify wa INTO, the entry to be processed is taken from the explicitly specified work area wa. If not, it comes from the header line of the internal table itab.

After COLLECT, the system field **SY-TABIX** contains the *index of the – existing or new – table entry* with default key fields which match those of the entry to be processed.

Expliquez le fonctionnement de l'instruction MESSAGE.

L’instruction MESSAGE permet d’afficher des messages à l’utilisateur. Ces messages sont regroupés en classes de message qui sont stockés dans la table T100. La classe de message utilisée pour une application doit être précisée à côté de l’instruction REPORT. Un message d’une classe de messages à un identifiant et un texte. Ce texte peut contenir jusqu’à 4 ‘&’ qui représentent des paramètres qui peuvent être passés lors de l’appel au message. De plus, lorsqu’on appelle un message, il faut préciser un type. Ce type conditionne la manière dont il sera affiché à l’utilisateur. Voici les différents types :

|  |  |  |
| --- | --- | --- |
| A | Abend (Termination) | Le message apparait dans une boite de dialogue et le programme se termine et le contrôle revient au précédent. |
| E | Error | En fonction du contexte de l’application, un message d’erreur apparait ou l’application se termine. |
| I | Information | Le message apparait dans une boite de dialogue. Si l’utilisateur confirme le message, le programme continue directement après l’instruction MESSAGE. |
| S | Status | Le programme n’est pas interrompu, un message est affiché dans la barre d’état. |
| W | Warning | En fonction du contexte de l’application, un message d’erreur apparait ou l’application se termine. |
| X | Exit | Aucun message n’est affiché et le programme s’arrête. Ceci permet de forcer l’arrêt d’une application. |

Il y a plusieurs syntaxes pour utiliser l’instruction MESSAGE (sachant que t = type, nnn = numero, id = classe de messages) :

* MESSAGE ID <classeDeMessages> TYPE <type> NUMBER <numero>
* MESSAGE tnnn
* MESSAGE tnnn(id)

Exemple :

REPORT ZEXERCICE MESSAGE-ID ZEXERCICE1MESSAGES.

MESSAGE ID ‘ZEXERCICE1MESSAGES’ TYPE ‘s’ NUMBER ‘000’.

MESSAGE s000(ZEXERCICE1MESSAGES).

MESSAGE s000.

Quand on a mis un & dans le texte du message, il faut ajouter WITH ‘valeur de mon &’ derrière l’appel au message.

You cannot perform input checks in PAI modules of programs until you have transported the contents of the input fields to the ABAP program. You can then use logical expressions to check the values that the user entered. You should then allow the user to correct any wrong entries before calling further modules. You can do this by sending warning (type W) or error (type E) messages from PAI modules that are called in conjunction with the ABAP statements FIELD and CHAIN.

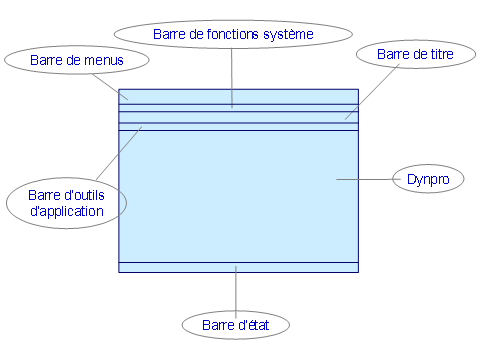
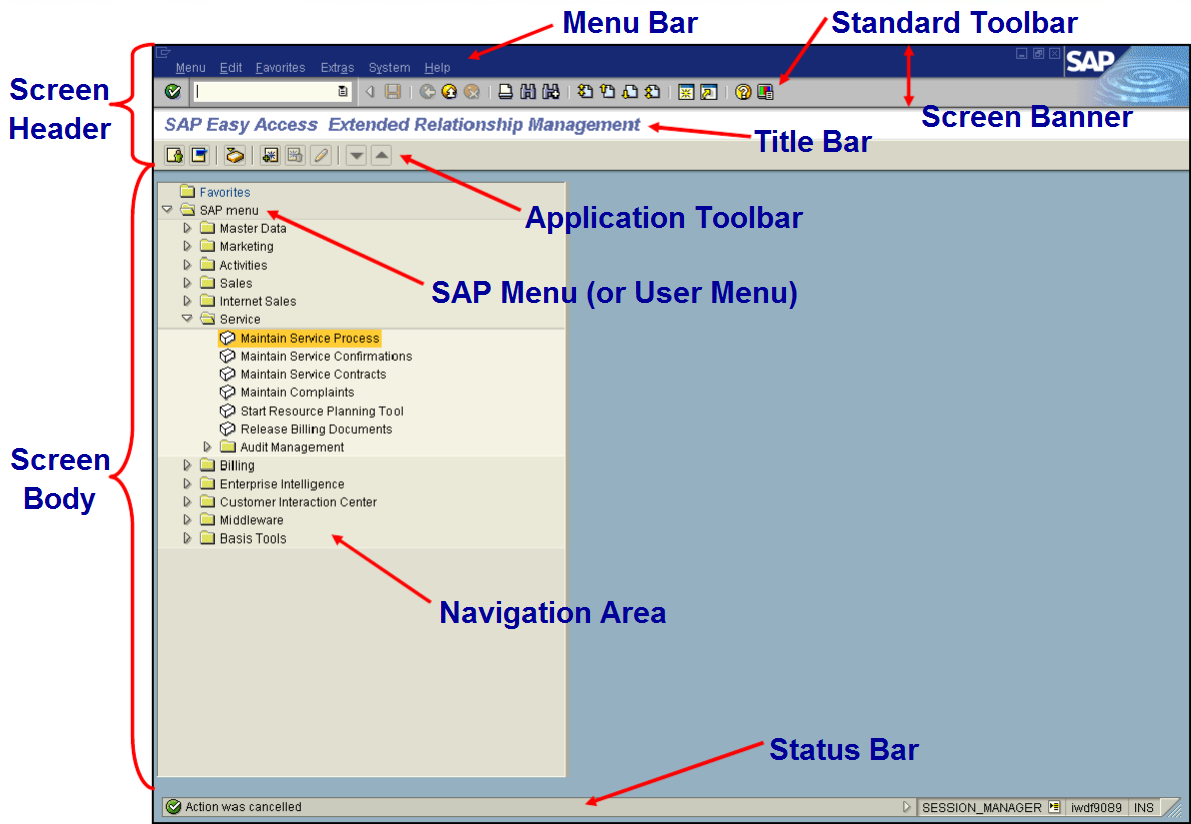
If you send a warning or error message from a module <mod> that you called using a FIELD statement as follows:

FIELD <f> MODULE <mod>

The corresponding input field on the current screen (and only this field) is made ready for input again, allowing the user to enter a new value. If the field is only checked once, the PAI processing continues directly after the FIELD statement, and the preceding modules are not called again.

Quelles sont les différentes parties d'un écran SAP lorsqu'il apparait à un utilisateur ? Quel est le rôle de chacune de ces parties et par quel outil et/ou instruction ABAP les gère-t-on ?

## p164,165,166,....

Les barres d’outils qui sont au-dessus du dynpro sont gérées par le menu-painter :

* SET TITLEBAR <title> : permet de définir le titre de tous les écrans suivants d’une succession d’écrans jusqu’à ce qu’un nouveau titre soit donné.
* SET PF-STATUS <nomDeLObjetStatus>

Le dynpro est géré par le screen-painter.

La barre d’état peut afficher des informations grâce à l’instruction MESSAGE.

Le screen-painter permet de créer des objets status. L’objet status permet de modifier les informations suivantes :

* **menu bar** : Les menus vous permettent de trouver une transaction spécifique lorsque vous ne connaissez pas le code de transaction. Le menu est organisé en fonction de la tâche que vous exécutez dans le système SAP.
* **standard toolbar** : La barre de fonctions système est la barre qui se trouve au-dessous de la barre de menus. Elle est constituée d'un ensemble d'icônes correspondant aux fonctions GUI générales (sauvegarder, retour, terminer, imprimer, rechercher…) et d'une zone où vous pouvez saisir un code de transaction.
* **application toolbar** : Elle contient les icônes significatives pour cette application en particulier.
* **function keys** : raccourcis clavier pour les entrées de la barre des menus. Attention, tous les raccourcis doivent pointer vers une entrée dans le menu, mais toutes les entrées dans le menu n’ont pas nécessairement de raccourci clavier.

Par quelle(s) techniques peut-on facilement réaliser des application multilingues et multidevises ?

Ne jamais hardcoder des libellés dans le code source, traduits

Code langue dans la table SY-LANGU

/\* /2 permet de passer deux fois à la ligne, (001) permet de définir un élément qu'on peut ensuite traduire dans GOTO -> Translation \*/

**WRITE:** **/**2 **(**18**)** 'Material'**(**001**),** **(**40**)** 'Description'**(**002**).**

/\* Inputs de l'utilisateur, pour que les libellés soient traduits, GOTO -> Text Element -> Selection Texts, Cocher la case dictionnaire \*/

**PARAMETERS:** p$\_land1 **TYPE** t001w**-**land1 **DEFAULT** 'DE' OBLIGATORY**,**

pf\_unit **AS** CHECKBOX**.**

Ref : p.117, 118

Tous les messages qui peuvent apparaître dans les écrans de SAP peuvent être configurables.   
Le programmeur va créer un dictionnaire par langue et va placer ces messages dedans. Ainsi le programmeur ne fait plus qu’une référence vers le message grâce au “Text Element”.   
Ainsi les messages ne sont plus écrits en « dur » dans le code mais répertoriés dans des dictionnaires.

En pratique, il suffit d'écrire un nombre à trois chiffres de text-element après le texte pour l'identifier uniquement dans le programme. Ensuite on active le text élément en double cliquant dessus et on peut alors accéder au menu de traduction pour cet élément via goto->translation. Une autre solution est d'utiliser TEXT : dans le programme, les TEXT ELEMENTS sont définis par le mot clé TEXT combiné avec un nombre à 3 chiffres (cela à l'avantage que le contenu du texte est également inclus dans le code du programme).

On n’hardcode donc jamais rien (pas de libellé dans le code source) et lors de l'exécution, on va utiliser la langue de l’utilisateur SY-LANGU, par exemple pour les entêtes, les textes de sélection, … Le système identifie chaque TEXT ELEMENT par son nombre et la langue de login (SY-LANGU).

La traduction des TEXT ELEMENTS peut être totalement indépendante du développement du programme.

Concernant les devises, pour que chaque personne puisse configurer la manière d'afficher les devises, et pour que notre programme affiche les devises selon ces configurations, il faut toujours utiliser l'option CURRENCY du mot clé WRITE :

**Syntax**

**WRITE f CURRENCY c.**

This statement determines the number of decimal places in the output according to the currency **c**. If the contents of **c** exist in table TCURX as currency key CURRKEY, the system sets the number of decimal places according to the entry CURRDEC in TCURX. Otherwise, it uses the default setting of two decimal places. That means that TCURX only has to list the exceptions with a number of decimal places other than 2.

The output format for currencies does not depend on the decimal places of a number that may exist in the program. The system uses only the sequence of digits. This sequence of digits thus represents an amount specified in the smallest unit of the currency in use, for example Cents for US Dollar (USD) or Francs for Belgian Francs (BEF). For processing currency amounts in ABAP programs, SAP therefore recommends that you use data type **p** without decimal places.

**REPORT demo\_list\_write\_currency LINE-SIZE 40.**

**DATA: num1 TYPE p  DECIMALS 4 VALUE '12.3456',**

**num2 TYPE p  DECIMALS 0 VALUE '123456'.**

**SET COUNTRY 'US'.**

**WRITE: 'USD', num1 CURRENCY 'USD', num2 CURRENCY 'USD',**

**/ 'BEF', num1 CURRENCY 'BEF', num2 CURRENCY 'BEF',**

**/ 'KUD', num1 CURRENCY 'KUD', num2 CURRENCY 'KUD'.**

1. Les RFC, Remote Function Call, sont des fonctions appelables à distance via un protocole de type CPI/C (aussi connu sous le nom de RPC), les ancêtres des webservices. [↑](#footnote-ref-1)