MASTER EN SCIENCES INFORMATIQUES

Protocole d'appareil virtuel

Rotens, Etienne; Tournay, Patrick

Award date:
1985

Awarding institution:
Universite de Namur

Année académique 1984 - 1985.

P R O T O C O L E

D' A P P A R E I L V I R T U E L

Promoteur BRUNIN J.

ROTENS Etienne

TOURNAY Patrick

Nous adressons notre gratitude à Monsieur BRUNIN pour les conseils judicieux qu'il nous a donnés durant ce travail.

Nous remercions toutes les personnes, parents et amis, qui nous ont aidés, encouragés de quelque façon que ce soit au cours de nos études et de l'élaboration de ce travail.

TABLE DES MATIERES.

T A B L E D E S M A T I E R E S.

PREMIERE PARTIE.

DEUXIEME PARTIE

PAGES

A V A N T - P R O P O S

A V A N T - P R O P O S.

A une époque où les réseaux de transmission de données sont en pleine expansion, où l'on parle de portabilité, compatibilité des systèmes informatiques, qu'en est-il réellement ?

Si on sait qu'actuellement pour un même problème, on est obligé de créer de nombreuses applications supplémentaires, dû au fait que les terminaux possèdent des caractèristiques d'entrée/sortie différentes, on peut constater que le chemin de la compatibilité est encore long.

Cependant, certains travaux sont en cours afin d'accroître les interactions possibles entre les systèmes, et parmi ceux-ci le protocole d'appareil virtuel (PAV).

Lorsqu'on aborde un sujet aussi vaste, il est nécessaire, au départ, de choisir l'orientation à donner à l'étude.

Celle que nous avons choisie a été motivée par cette question :

Comment est-il possible de concrétiser pratiquement un noyau du protocole d'appareil virtuel facilement extensible aux problèmes à venir ?

Pour répondre à cette question, nous avons divisé notre travail en deux parties.

La première partie, plus théorique, pose d'abord, les problèmes qui ont amené la nécessité de créer le modèle ISO.

Elle présente, ensuite et de manière générale, les solutions possibles pour résoudre ces problèmes : le protocole d'appareil virtuel en étant une. Enfin, elle explicite les bases du protocle en développant les éventuels choix qui sont offerts à leur réalisation.

La seconde partie constitue ce que nous avons appelé notre solution.

Pour chacun des fondements du protocole présenté dans la première partie et parmi les diverses manières de les réaliser qui nous étaient données, nous avons posé des choix, les avons justifiés, en présentant leurs avantages et inconvénients ainsi que leurs limites et, enfin, les avons exploités jusqu'à les amener à un niveau concret proche de la réalisation.

P R E M I E R E P A R T I E.

C H A P I T R E I.

L E M O D E L E D E R E F E R E N C E D E L'I S O.

CHAPITRE I : LE MODELE DE REFERENCE DE L'ISO.

1.1. INTRODUCTION.

Les réseaux de téléinformatique sont nés à la fin des années 50 quand on a commencé à déporter quelques terminaux très simples comme des télétypes asynchrones (TTY). Les ordinateurs sont ensuite devenus de plus en plus puissants, le nombre et les possibilités des terminaux se sont accrus et différentes procédures de télécommunication (asynchrone TTY, synchrones BSC...) ont commencé à proliférer.

Cette évolution s'est encore accélérée suite aux coûts rapidement décroissants de l'électronique, à l'imagination des fabricants de terminaux et aux besoins d'interconnecter de nombreux ordinateurs en un vaste réseau à l'échelle d'un pays ou même de la planète (réservation des places d'avion).

Il devint bientôt urgent de structurer : cette réflexion est à l'origine des produits des constructeurs tel que SNA chez IBM, Decnet chez Digital, BNA chez Burroughs, DSA chez Honeywell Bull,...

Cette structuration des produits des constructeurs profite certes à l'utilisateur mais le lie toujours plus à son fournisseur principal d'où un besoin de plus en plus pressant de normes internationales téléinformatiques.

C'est pourquoi l'Organisation Internationale de Normalisation ISO (International Standard Organisation) a créé un sous-comité chargé de l'interconnexion des systèmes ouverts (OSI = Open System Interconnection). Ce sous-comité s'est consacré à la définition d'un modèle d'architecture de réseau informatique au sein de laquelle pourraient prendre place les protocoles normalisés qui permettraient la constitution de réseaux multiconstructeurs.

Rappelons brièvement les caractéristiques principales et le rôle de chacune des sept couches qui forment ce modèle.

1.2. LES 7 COUCHES.

1.2.1. Couche physique.

La couche physique fournit les procédures normalisées (X21, V24...) et les fonctions mécaniques, électriques et fonctionnelles nécessaires pour établir, maintenir et libérer des connexions physiques entre équipements terminaux de traitement de données (ETTD), équipements de terminaison de circuit de données (ETCD) et/ou centres de commutation de données (CCD).

Elle assure la transmission de flots binaires par une connexion permanente ou dynamique, en duplex ou en semi-duplex.

Elle assure une compatibilité des interfaces qui remplissent des fonctions telles que le codage, la modulation et l'amplification du signal.

1.2.2. Couche liaison.

La couche liaison fournit les procédures (HDLC, SDLC...) et les moyens fonctionnels nécessaires pour établir, maintenir et libérer des connexions de liaison de données entre ETTD, ETCD et CCD.

Les services fournis par cette couche sont :

- le contrôle de flux,
- le séquencement des blocs de données,
- la détection et contrôle des erreurs de transmission,
- la retransmission dans les cas d'anomalies.

1.2.3. Couche réseau.

La couche réseau fournit les procédures et les moyens fonctionnels nécessaires à l'échange des informations données par la couche supérieure. C'est un service de bout en bout qui est responsable de l'acheminement des paquets de données qui peuvent traverser plusieurs noeuds intermédiaires.

Les services fournis par cette couche sont :

- Le service d'adressage sur le réseau,
- le contrôle de flux des paquets,
- la fonction de routage des paquets de données,
- les fonction de détection et de recouvrement des erreurs.

1.2.4. Couche transport.

La couche transport fournit à la couche supérieure le transfert transparent des messages des utilisateurs. Elle s'occupe de tous les détails de l'exécution d'un transfert de données et assure l'arrivée correcte des messages (groupe de paquets) des utilisateurs connectés aux réseaux, à leurs destinataires.

Elle optimise l'emploi des ressources de transmission disponibles, afin d'assurer le plus économiquement possible le niveau de performance requis par chaque utilisateur du service de transport.

Les services fournis par cette couche sont :

- le service d'adressage au niveau transport,
- le contrôle de flux des messages,
- la détection d'erreurs et les reprises,
- la purge après des graves problèmes de transmission.

1.2.5. Couche session.

La couche session a pour objectif de négocier les interactions entre les utilisateurs pour synchroniser les opérations effectuées sur les données.

Les services fournis par cette couche sont :

- les mécanismes et les algorithmes nécessaires pour garantir la cohérence des contextes.

Elle est composée d'autant de modules d'échanges qu'il y a de type d'application (acquisition de données, interactivité,...)

1.2.6. Couche présentation.

La couche présentation permet d'interpréter la signification des données échangées entre les utilisateurs. Elle assure une compréhension syntaxique entre les utilisateurs, en gérant les formats des données à échanger et en effectuant les transformations nécessaires sur les données pour les rendre compréhensibles entre matériels hétérogènes.

Les services fournis par cette couche sont :

- la négociation des paramètres de formatage du support,
- la présentation des données échangées suivant un format compréhensible par les applications.

1.2.7. Couche application.

La couche application permet la compréhension et l'exécution de commandes liées aux processus d'application.

Les principales fonctions concernent les demandes de connexions entre plusieurs processus d'application distants.

Les services fournis par cette couche sont :

- l'adressage, l'activation et la désactivation des processus,
- la surveillance effectuant le contrôle d'erreurs et la reprise sur blocage.

Enfin, voici une liste de quelques types d'application :

- le télétraitement
- le conversationnel
- l'acquisition de données
- le transfert de fichiers, de documents...
- la télémesure
- la commande de processus industriel.

C H A P I T R E II.

L A C O U C H E P R E S E N T A T I O N.

CHAPITRE II : LA COUCHE PRESENTATION.

Dans toutes les fonctions qui viennent d'être énoncées, il reste à développer une fonction importante de la couche présentation à savoir le traitement de l'incompatibilité du formatage et du codage des données entre une application et un terminal.

Nous allons approfondir les problèmes soulevés par cette fonction et présenter des solutions envisageables.

2.1. DEFINITION DU PROBLEME.

Nous savons que les messages échangés entre un système de traitement et un terminal comportent généralement, en plus de l'information utile destinée à être interprétée par l'utilisateur, des informations permettant de commander à distance certaines fonctions du terminal.

Par exemple, le caractère CR de l'alphabet international n° 5 (IA5), transmis à la fin d'un message, provoque sur un télétype le retour de la tête d'impression et l'avancement du papier. Sur un terminal à écran, les informations de commande accompagnant le message servent à indiquer la position du message dans l'écran, l'utilisation de dispositifs de mise en évidence de certaines zones de texte (surbrillance, clignotement,...) la protection de zones en saisie de données, etc...

Le programmeur d'application doit généralement insérer lui-même dans le texte émis ces informations de commande du terminal. Il existe une très grande variété de fonctions sur les terminaux ainsi que de caractéristiques physiques (par exemple le nombre maximum de caractères par ligne, le nombre maximum de lignes par écran...). De plus, la façon d'indiquer une fonction à réaliser par une information de commande, peut varier considérablement d'un terminal à l'autre.

Le programme d'application est alors très dépendant du terminal utilisé et la connexion de plusieurs types de terminaux à un même programme se fait au prix d'une extrême complication. Ceci pose un grave problème aux utilisateurs qui veulent renouveler leur parc de terminaux ou travailler avec des parcs mixtes, car le coût de modifications et de mise au point des programmes existants devient prohibitif. Cette dépendance entre programme et terminal rend également difficile l'élargissement de systèmes téléinformatiques à un groupe important de terminaux et de centres de traitements qui est rendu possible grâce aux réseaux de commutation de données privées ou publics.

2.2. LES SOLUTIONS ENVISAGEABLES.

Afin de réduire cette dépendance entre un programme d'application et un terminal, nous envisageons deux alternatives.

2.2.1. La banalisation des terminaux.

La banalisation des terminaux revient à ce que tous les constructeurs basent la conception de leurs produits sur les mêmes critères de codage de fonctions, de format d'écran,...: tous les terminaux ont les mêmes caractéristiques et sont directement compatibles.

Cette alternative s'oppose à deux concepts qui la rendent sans effet jusqu'à présent :

* d'une part la constante évolution des terminaux nécessite la révision continuelle des classifications des fonctions par modification ,suppression et création de paramètres

*d'autre part, beaucoup de constructeurs n'ont ni la motivation, ni l'intérêt à ce que ces discordances soient atténuées.

2.2.2. L'adaptation des messages.

L'adaptation des messages peut être mise en oeuvre de deux manières différentes, soit par une adaptation spécifique à chaque terminal, soit par la création d'un langage standard associé à des logiciels d'interface.

2.2.2.1. Adaptation spécifique.

L'adaptation spécifique est une méthode qui doit être évaluée en fonction du système informatique considéré. Dans le cas d'un système central unique auquel sont raccordés des terminaux locaux, une adaptation des messages pour chaque type de terminal est possible. Par contre, dans le cas d'un réseau d'ordinateur, le problème de la diversité se pose avec beaucoup plus d'acuité et il n'est plus concevable de réaliser ces adaptations.

Montrons sur l'exemple d'un réseau maillé l'impact de cette méthode sur le nombre de logiciels nécessaires.

Soit un réseau composé de S serveurs, chacun de ceux-ci disposant de Pi programmes d'application. (Fig. 2.2.2.1.)

Soit T terminaux de type différent susceptible de pouvoir accéder à tous les programmes des S serveurs.

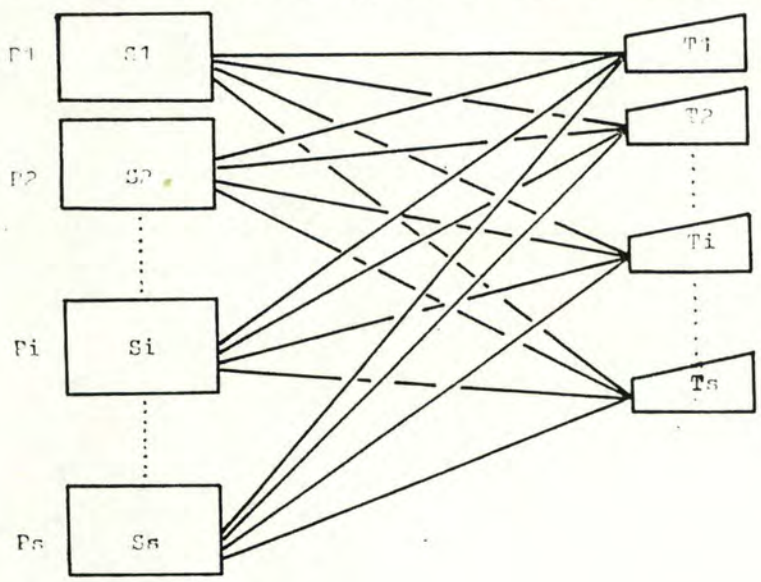


fig. 2.2.2.1. : Adaptation spécifique.

Etant entendu que tous les programmes sont distincts ainsi que les terminaux, le nombre total de logiciels accessibles pour chacun des terminaux est

$$\sum_{i=1}^{S} Pi$$

Assurer la compatibilité totale pour les T terminaux demande une version de chaque programmepour chaque terminal c'est-à-dire :

$$\sum_{i=1}^{S} Pi\ T$$

Considérons 200 programmes (Pi = 200) répartis sur 5 serveurs (S=5) et 30 terminaux (T=30), il résulte que le nombre total de logiciels nécessaires est égal à

$\sum_{i=1}^{5} 40.30 = 6000$ logiciels.

2.2.2.2 Adaptation par langage standard.

Plutôt que de réaliser une adaptation spécifique organisée à l'aide d'un langage influencé par les caractéristiques du terminal réel, nous assurons l'adaptation générale grâce à un langage standard d'interaction entre un terminal et une application. (fig. 2.2.2.2.)

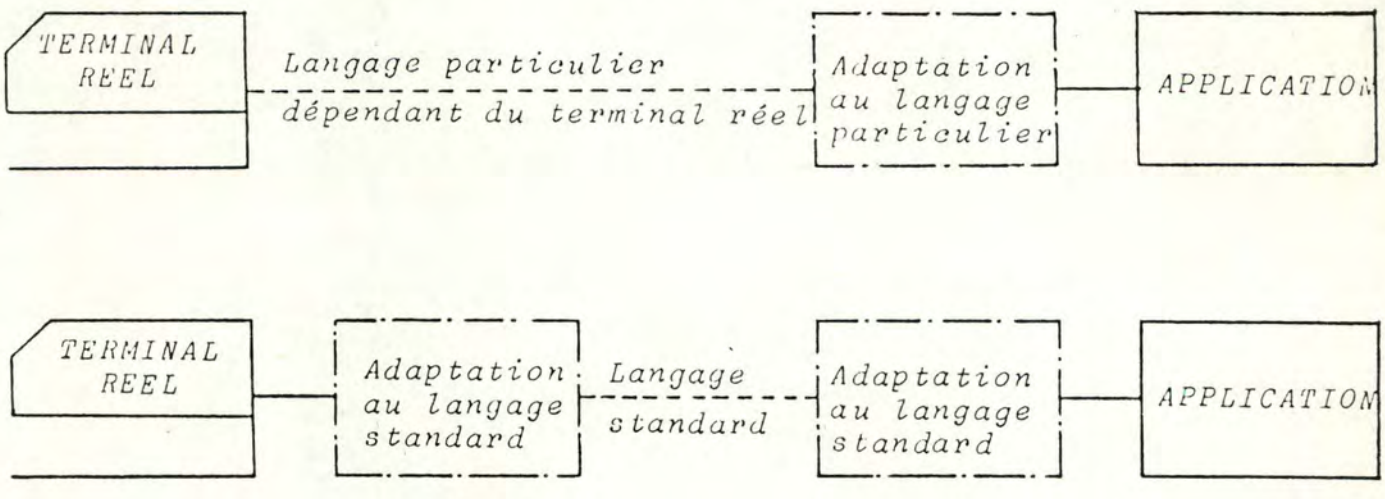


fig. 2.2.2.2. : Adaptation au langage standard

Ce langage est défini à l'échelle du réseau. Il existe deux adaptations locales :

* du langage spécifique du terminal au langage standard et inversement
* du langage spécifique de l'application au langage standard et inversement.

Montrons, parallèlement à ce qui a été fait pour l'adaptation spécifique l'impact de cette méthode sur le nombre de logiciels.

Le nombre <u>total</u> de logiciels accessibles par chaque terminal étant le même, la compatibilité totale est assurée par les seules adjonctions de logiciels d'interfaces. Nous avons :

$$\sum_{i=1}^{S} Pi + T + S$$

où T et S sont les logiciels d'interfaces respectivement des terminaux et des serveurs.

Evaluons sur base des mêmes données que dans l'exemple précédent et il vient que le nombre total de logiciels nécessaires est égal à :

$$\sum_{i=1}^{5} 40 + 30 + 5 = 235.$$

Parce qu'elle est la seule qui puisse être employée sur une grande échelle et qui n'impose rien au constructeur, cette dernière solution peut être considérée, du moins théoriquement, comme la plus valable.

Ainsi, n'en fallait-il pas plus pour qu'elle trouve son prolongement dans un protocole : <u>le protocole d'appareil virtuel</u>.

C H A P I T R E III.

L E P R O T O C O L E D'A P P A R E I L V I R T U E L.

CHAPITRE III : LE PROTOCOLE D'APPAREIL VIRTUEL.

Ce chapitre s'attache à décrire les 5 notions fondamentales sur lesquelles repose le protocole d'appareil virtuel (P.A.V.).

- le langage standard : critères de choix et conséquences,
- la notion de terminal virtuel,
- le dialogue entre partenaires : description et conséquences,
- la nécessité d'une négociation entre partenaires et l'utilité d'une émulation,
- l'organisation des échanges de la station VTP.

3.1. LE LANGAGE STANDARD.

3.1.1. Choix d'un langage normalisé.

Le langage standard est une des bases du protocole d'appareil virtuel. Evidemment, c'est dans la mesure où il est normalisé qu'il prend son intérêt véritable.
Il est donc indispensable de le définir par un ensemble de commandes normalisées ayant chacune une définition rigoureuse de la fonction qu'elle représente et un codage précis.

Considérons ce qui existe actuellement :

* certaines fonctions élémentaires telles que :

  BS : Back Space
  LF : Line Feed
  CR : Carriage Return

  sont reprises dans les commandes normalisées internationales ASCII n° 5, IA2 et dans le code auxiliaire EBCDIC.

* d'autres fonctions de gestion de terminal plus élaborées telles que la mise en évidence de zones, l'adressage dans une page, la définition de zones protégées, etc... sont, par contre, totalement absentes des codes normalisés internationaux.

Nous pouvons noter, cependant, qu'à défaut de norme internationale, l'American National Standard Institute Inc (ANSI) définit un ensemble de fonction de contrôle pour faciliter les échanges de données entre des appareils d'entrée/sortie.
Celles-ci comprennent des fonctions d'édition (curseur), de formatage, de spécification et de contrôle des zones protégées, de sélection de mode, etc...
Leur structure de codage est similaire aux structures des séquences "ESCAPE" qui grâce aux paramètres numériques et sélectifs ont l'avantage de toujours être aptes à accepter de nouvelles fonctions (voir par exemple annexe II).

3.1.2. Conséquence de l'adaptation à un langage standard.

La traduction entre commandes normalisées et commandes interprétables par le terminal réel a deux conséquences :

- La nécessité d'utiliser une interface.
- La création de la notion de terminal virtuel.

3.1.2.1. L'interface.

L'adaptation au langage standard nécessite l'adjonction d'une interface de conversion qui peut se situer à différents niveaux :

. dans l'ordinateur ou son frontal,

. dans le terminal de façon intégrée ou complémentaire de diverses manières :

Soit par :

* Modification du logiciel interne du terminal.

  Si le terminal est géré par un software, cas de la majorité des appareils actuellement, il est possible de modifier le logiciel pour y insérer les modules nécessaires à l'adaptation au langage standard.

Avantages :

+ L'intégrité hardware du terminal est conservée.
+ Cette méthode ne demande pas d'ajout de matériel de type interface, etc...
+ La modification ne diminue pas les performances du dialogue terminal-ordinateur.
+ Une étude d'adaptation est valable pour tous les terminaux de même modèle.
+ L'investissement à consentir réside dans l'étude des modules et leur implantation dans le logiciel existant.

Inconvénients :

+ Il est nécessaire de connaître le logiciel existant sur le modèle de terminal considéré, pour pouvoir insérer les nouvelles fonctions sans perturber le fonctionnement normal de l'appareil.
+ Il est possible que l' on soit confronté à un problème de place mémoire suite à la limitation de cette dernière.
+ Une étude complète est indispensable par modèle de terminal.
+ Aucun module de l'adaptateur ne peut être étudié de manière universelle puisqu'il sera fonction du processeur interne du terminal.

* Par modification hardware.

Avantages :

+ Cette solution peut être envisagée quel que soit le genre de terminal ; qu'il soit géré par software ou non.
+ La modification laisse toute latéralité au concepteur et rend cette méthode souple.

Inconvénients :

+ La méthode est complexe et coûteuse en ce sens que, si l'étude d'adaptation est applicable pour tous les terminaux de même modèle, la modification hardware doit se faire sur chacun d'eux.
+ Revient à refaire un nouveau terminal.

* Par ajout d'une interface indépendante.

Cette méthode consiste à adjoindre au terminal une interface indépendante constituée par une carte µp+ mémoire.

Avantages :

+ L'intégrité hardware du terminal est maintenue.
+ Le logiciel de gestion du terminal, s'il en existe un, ne doit pas être connu. Seule est nécessaire la connaissance des codes de contrôle du terminal.
+ Cette méthode ne demande qu'une seule étude d'adaptation quel quesoit le modèle de terminal. La seule modification dépendant du modèle de terminal réside dans la mise à jour du vecteur qui reprend les codes de contrôle propre à son modèle.
  On crée ainsi un logiciel de base identique pour tout terminal.
+ Le problème de la place mémoire ne se pose pas. Eventuellement on peut ajouter dans le logiciel des modules d'émulation supplémentaires.

Inconvénients :

+ Il existe un niveau de dialogue et de transmission supplémentaire, entre terminal et ordinateur, et donc une diminution des performances.

3.1.2.2. Notion de terminal virtuel.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-

Le terminal composé du terminal réel et de l'adaptation au langage standard est considéré comme un terminal abstrait que nous appellerons dorénavant terminal virtuel. (fig. 3.1.2.2.)

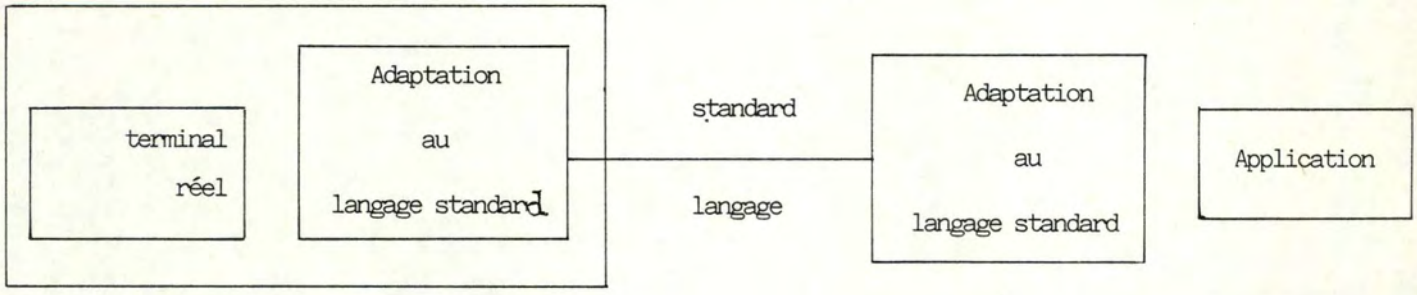


fig. 3.1.2.2. : Terminal Virtuel.

3.2. LES CLASSES DE TERMINAUX VIRTUELS.

L'examen de l'ensemble des terminaux présents sur le marché fait apparaître un éventail très large qui va du terminal simple (TTY) au terminal très évolué (Vidéo Data Entry).

La disparité de cet éventail est telle qu'il paraît difficilement concevable de définir un terminal virtuel unique qui convienne dans tous les cas. En effet, si d'une part on définit un terminal virtuel très évolué, on rend coûteux et difficile l'adaptation des terminaux simples à cause de l'ajout important de software et éventuellement de hardware, si d'autre part on définit un terminal virtuel très simple, on ne permet plus aux terminaux évolués d'utiliser toutes leurs possibilités et facilités.
Aussi, pour concilier les avantages d'une adaptation économique du terminal réel et d'une bonne utilisation des possibilités de ce dernier, il est souhaitable d'introduire plusieurs terminaux virtuels de complexité croissante. (Les plus simples étant définis comme sous-ensemble des plus complexes.).

La classification des terminaux se fait sur base de deux critères :

1. Le mode de fonctionnement du terminal.

   Ce critère est certainement le plus efficace pour séparer les terminaux les plus simples des plus évolués.

   Il existe trois modes de fonctionnement :

   A - Le mode Scroll : caractérise l'utilisation d'une structure de données à une dimension (ligne) constituée d'une seule zone (pas de notion d'attribut).

   B - Le mode Page : caractérise l'utilisation d'une structure de données à deux dimensions (page). Le seul attribut utilisable est l'attribut de protection de zone.

C - Le mode Data Entry : caractérise l'utilisation d'une structure de données à deux dimensions (page). Il est doté d'un jeu d'attributs nettement plus étendu (protection numérique, alphabétique..., sélection par crayon lumineux...).

2. Le type de terminal.

Ce critère permet de séparer des terminaux de nature différente. Il existe trois types de terminaux :

A - Imprimante : le terminal ne possède pas de clavier alphanumérique pour introduire des données. L'affichage se fait sur un support papier.

B - Télétype : le terminal possède un clavier alphanumérique pour introduire des données. L'affichage se fait sur un support papier.

C - Vidéo : le terminal possède un clavier alphanumérique pour introduire des données. L'affichage se fait par tube cathodique ou similaire.

En combinant les éléments de ces deux critères, nous allons définir les classes de terminaux :

- Le mode page et le mode data entry ne regroupent que des terminaux de type vidéo puisqu'ils sont les seuls à pouvoir travailler dans une structure de données à deux dimensions, nous obtenons ainsi les deux classes :

  * Data Entry Video ⟶ Data Entry
  * Page Video ⟶ Page

- Le mode scroll regroupe les trois types de terminaux. Mais le fait que les possibilités d'un terminal type vidéo sont plus proches de celles d'un terminal mode page (structure à deux dimensions) que d'un télétype, nous a amené à séparer les vidéos des TTY et imprimantes.

De plus, comme le TTY dispose d'un clavier et pas les imprimantes, nous obtenons les trois classes :

Mode scroll - TTY
Mode scroll - Imprimante
Mode scroll - Vidéo

Nous nous proposons donc d'introduire cinq terminaux virtuels définis sur les cinq classes précédentes :

Terminal virtuel mode scroll imprimante
Terminal virtuel mode scroll télétype
Terminal virtuel mode scroll vidéo
Terminal virtuel mode page
Terminal virtuel mode data entry

Pour appartenir à une classe donnée, un terminal quelconque doit posséder un certain nombre de fonctions.

On trouve dans le tableau 3.1. la liste des fonctions pour chacune des 5 classes.

CARACTERISTIQUES / FONCTIONS D 'ADDRESSAGES	CLASSES				
	Scroll mode Imprimante	Scroll mode Teletype	Scroll mode Video	Page	Data Entry
BS	X	X	X	X	X
CR	X	X	X	X	X
LF	X	X	X	X	X
TAB	X	X	X	X	X
↑			X	X	X
↓			X	X	X
→			X	X	X
←			X	X	X
Cursor Adress			X	X	X
Attributs					
Protégé			X	X	X
Jeu d'attributs					X
Transmission					
	X	X	X	X	X
Page				X	X
champs				X	X
Fonctions effacements					
			X	X	X
Clavier					
		X	X	X	X

Tableau 3.1. : Caractéristiques minimales des classes des terminaux.

3.3. LE DIALOGUE ENTRE DEUX PARTENAIRES.

Jusqu'à présent nous n'avons abordé que l'aspect d'une adaptation à un langage standard. Bien que celle-ci soit essentielle, elle reste purement locale et statique en ce sens qu'elle donne à chacun des partenaires les moyens de se comprendre mais pas de communiquer.

Dans ce paragraphe nous allons aborder l'aspect de la conversation entre entités distantes.

3.3.1. Les phases du dialogue.

Le dialogue entre un terminal et une application ou entre deux terminaux doit inclure les trois phases suivantes :

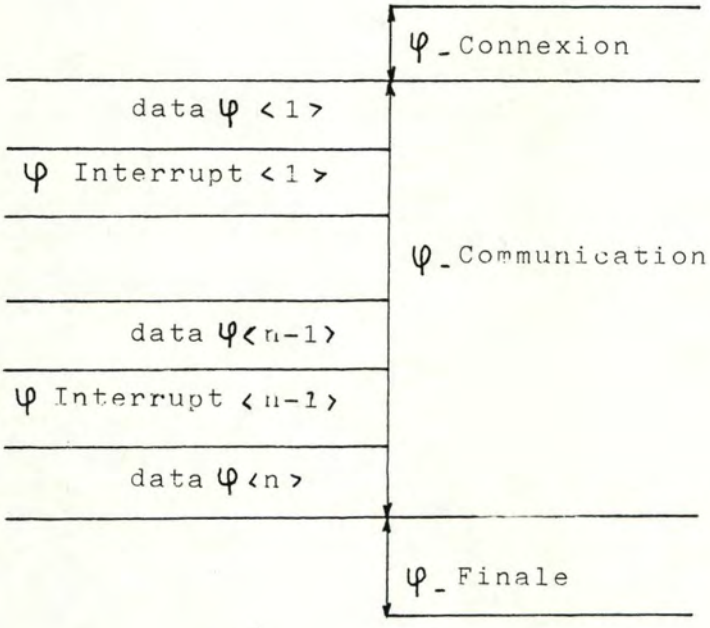
* La phase de connexion qui comprend d'une part la demande et l'établissement de la communication et d'autre part une négociation au cours de laquelle les deux partenaires essaient de se mettre d'accord sur l'ensemble des caractéristiques nécessaires à un bon dialogue. (Par exemple le choix du nombre de caractères/ligne, du nombre de lignes/écran,..). Ce point fera l'objet d'un paragraphe ultérieur.

* La phase de communication au cours de laquelle les deux partenaires ont la possibilité d'échanger des données. Cependant, le processus d'échange de données peut être parfois interrompu quand un des partenaires (cas de renversement d'échanges, temporisation dans les transferts, dialogues interactifs) demande la gestion d'une condition d'interruption. Celle-ci est traitée au cours d'une phase d'interruption qui provoque essentiellement la fin de l'échange de données courantes (appelée phase de données courantes) et l'initialisation d'un nouvel échange (une nouvelle phase de données). Cela signifie :
  - qu'aucune donnée ne peut être envoyée pendant cette phase d'interruption,

- que la phase d'interruption ne peut se terminer que lorsque toutes les données en transit dans le réseau ont atteint leur destination et ont été traitées. (Le traitement étant, bien entendu, fonction de la nature de l'interruption).

La phase de communication peut alors être vue comme une succession de phases de données séparées par des phases d'interruption.

* La phase finale au cours de laquelle la fin du dialogue est déclarée.



Fig; 3.3.1. : les différentes phases du dialogue.

Remarques sur la phase d'interruption.

Quand un partenaire entre dans une phase d'interruption, l'autre (le côté receveur) doit en être informé de telle sorte qu'il puisse aussi entrer dans cette phase. De plus, comme une condition d'interruption est généralement utilisée pour reprendre une situation incertaine ou anormale, (par exemple une erreur dans l'application), il est important que le receveur soit informé aussi vite que possible de la condition d'interruption. Il est donc nécessaire de court-circuiter le chemin normal de communication avec ses contraintes de séquencement et de contrôle de flux et d'utiliser un chemin parallèle offrant une priorité pour de courts items d'information (les télégrammes) qui reprendront les conditions d'interruption.

Cela implique, puisque les télégrammes sont essentiellement des items non séquencés, que le côté receveur ne peut distinguer entre les données d'une phase de données interrompue et celles d'une nouvelle phase.

Or, cette distinction doit nécessairement être préservée parce que les données de différentes phases peuvent devoir être traitées différemment : en effet, alors que les données d'une nouvelle phase seront gérées normalement par le terminal ou le processus d'application, les données issues d'une phase de données interrompue pourraient, par exemple, être purgées à l'arrivée côté receveur ou, de manière rétroactive, au niveau des blocs physiques.

Aussi, une séparation claire entre phases de données successives sera effectuée au moyen de marques de synchronisation venant par le chemin normal de communication (au commencement de la phase d'interruption).

3.3.2. Caractéristiques possibles du dialogue.

Caractériser le dialogue revient à caractériser l'organisation de l'échange entre les deux partenaires en communication.

Deux modes d'échanges sont possibles :

- L'échange alterné (alternate mode).
- L'échange libre

1. L'échange alterné

Dans ce mode , chacun des correspondants reçoit alternativement de l'autre le droit de parole. Ce droit ne peut en aucun cas être détenu simultanément par les deux parties et par conséquent, à un instant donné, les données ne peuvent circuler que dans un sens.

En ce qui concerne l' unité de visualisation, ressource utilisée concouremment par les deux partenaires, elle sera dans ce cas, allouée au correspondant qui a la parole. L'autre correspondant, quant à lui, pourra toujours préparer une structure de données alors qu'il n'a pas la parole à condition que le terminal permette le type "ahead" (en avant, go and forward) et que cette structure puisse être mémorisée. Les données ainsi préparées seront visualisées et transmises lorsque le tour de parole aura été acquis.

2. L'échange libre

Dans ce mode, chacun des correspondants laisse son vis à vis libre d'émettre des données à tout moment. Par conséquent, les données peuvent à tout instant circuler dans les deux sens.
Une règle de priorité définie localement déterminera lequel des deux correspondants accèdera à l'unité de visualisation à un instant donné. Pendant ce temps, les données produites par l'autre correspondant doivent être mémorisées de façon à permettre leur affichage ultérieurement.

3.3.3. Conséquences du dialogue.

Nous voyons aisément que la conversation entre deux entités distinctes va nécessiter une modification du schéma de terminal virtuel (fig. 3.1.2.2.), par l'ajout d'un module de gestion du dialogue indépendant du module gérant le langage standard, et la création de primitives relatives aux diverses phases du dialogue.

3.3.3.1. Modification sur le Terminal Virtuel.

La figure 3.3.3.1. ci-dessous fait apparaître la décomposition de chacun des modules d'adaptation, représenté à la figure 3.1.2.2., en deux sous-modules fonctionnels :

- Le sous-module émulateur ou sous-module interface selon le cas - on émule pas une application - qui joue un rôle exclusivement local.
  Ce sous-module, comme émulateur, gère le terminal réel en traduisant ses commandes en commandes interprétables par le terminal virtuel et vice-versa. Nous verrons dans la suite de ce chapitre que l'on peut attribuer à cet émulateur d'autres fonctions liées au P.A.V. telles les gestions de formats,...

  Ce sous-module, comme interface, se limite à une conversion de commandes. (Toutes les fonctions d'émulation sont exclues)

- Le sous-module "Station VTP" (Virtual Terminal Protocol) qui participe au processus de dialogue de bout en bout organise l'échange des primitives de dialogue entre entités distantes.

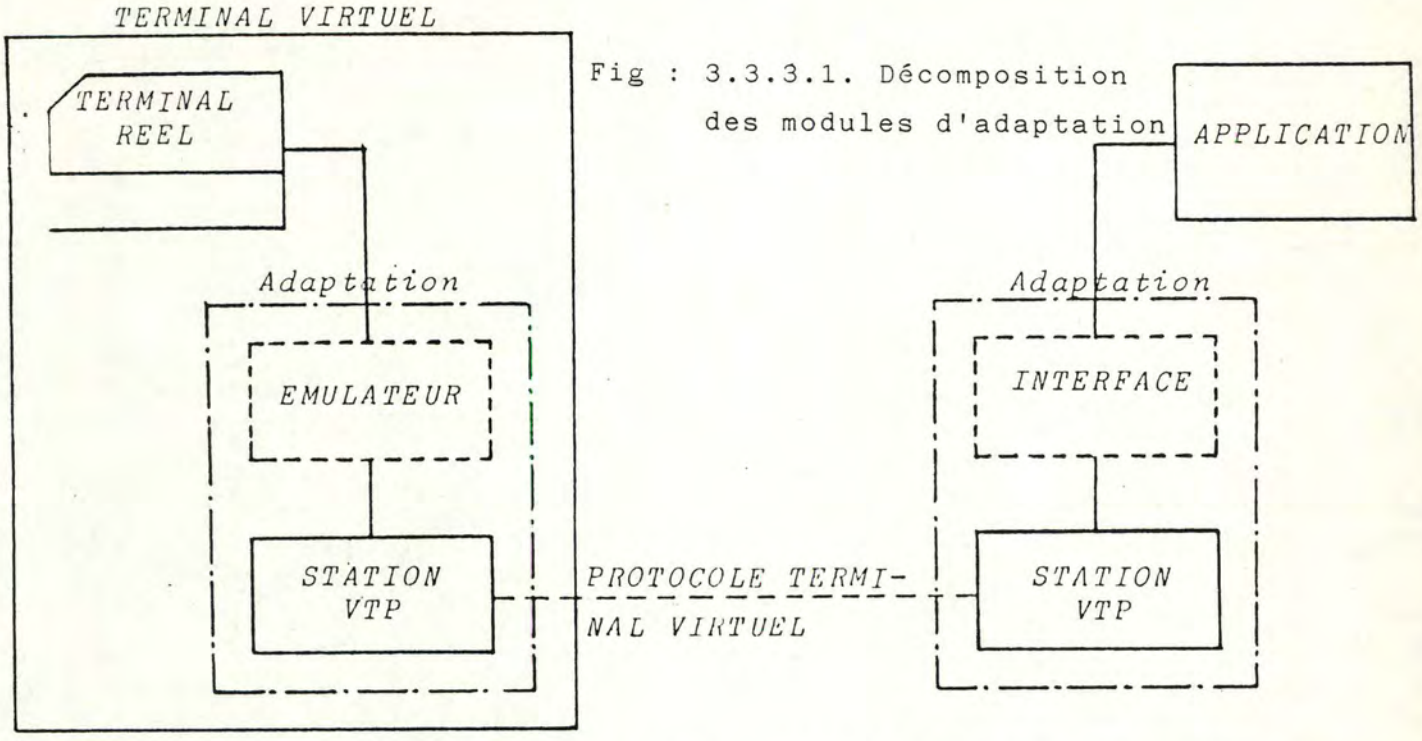


Fig : 3.3.3.1. Décomposition des modules d'adaptation

3.3.3.2. Les primitives du P.A.V. : classification et définition.

La classification des primitives est telle que chacune d'entre elle puisse être resituée dans les différentes phases du dialogue décrites en 3.3.1.
De plus leur définition comprend, en plus du sigle, du nom et de la définition proprement dite, les paramètres éventuels qui y sont associés.

3.3.3.2.1. Primitives appartenant à la phase de connexion.

- Négociation

  * CDCL : Command Document Capability List.
    Commande qui entre dans la phase de négociation. Elle provoque une demande de connexion et transmet la liste des capacités du terminal (voir négociation).

CDCL			STORAGE CAPACITY NEGOTIATION	NM
	NONBASIC TERMINAL CAPABILITIES	NM	GRAPHIC CHARACTER SETS	NM
			CONTROL CHARACTER SETS	NM
			PAGE FORMATS	NM
			MISCELLANEOUS TERMINAL CAPABILITIES	NM
	PRIVATE USE	NM	–	

* RDCLP : Réponse Document Capability List Positive.
Acceptation de CDCL

RDCLP			ACCEPTANCE OF CDCL PARAMETERS	NM
			STORAGE CAPACITY NEGOTIATION	NM
	NONBASIC TERMINAL CAPABILITIES	NM	GRAPHIC CHARACTER SETS	NM
			CONTROL CHARACTER SETS	NM
			PAGE FORMATS	NM
			MISCELLANEOUS TERMINAL CAPABILITIES	NM
	PRIVATE USE		–	–

* CDS : Command Document Start.
Commande qui indique à l'utilisateur le début du document.

CDS			DOCUMENT REFERENCE NUMBER	M
			SERVICE INTERWORKING IDENTIFIER	NM
			DOCUMENT TYPE IDENTIFIER	NM
	NONBASIC TERMINAL CAPABILITIES	NM	GRAPHIC CHARACTER SETS	NM
			CONTROL CHARACTER SETS	NM
			PAGE FORMATS	NM
			MISCELLANEOUS TERMINAL CAPABILITIES	NM
	PRIVATE USE	NM	–	–

* RDSP : Response Document Start Positive

3.3.3.2.2. Primitives appartenant à la phase de communication.
****

- Primitive annonçant les données.

  * CDUI : Command Document User Information.
    Commande qui indique à l'utilisateur que l'information doit être interprétée comme un domaine de données transmises.

CDUI			–	–

- Primitive de marquage.

  *CDPB : Command Document Page
    Indique, à chaque page, à l'utilisateur le numéro du point de repère.

CDPB			CHECKPOINT REFERENCE NUMBER	M

  * RDPBP : Response Document Page Boundary Positive
    Accusé reception positif à la commande CDPB par lequel l'utilisateur indique qu'il accepte toutes les pages jusqu'à y compris celle définie dans le paramètre adjoint et qu'il est prêt à recevoir de nouvelles pages.

RDPBP			CHECKPOINT REFERENCE NUMBER	M
			RECEIVING ABILITY JEOPARDIZED	M

* RDPBN : Response Document Page Boundary Negative.
Accusé réception négatif à la commande CDPB par lequel l'utilisateur indique qu'il n'accepte pas la responsabilité de la page suite à une détection d'erreur ou d'une défaillance.

- Saturation de la capacité mémoire.
- Erreur de séquence : la page reçue n'a pas le point de repère attendu.
- Erreur de terminal local.
- Erreur de procédure impossible à corriger.
- Autres raisons.

La réception de cette commande provoque le passage à une phase d'interruption.

RDPBN			REASON (DOCUMENT)	M

- Primitives de situation d'échange.

* CDRP : Command Document Response Positive.
Commande qui indique à l'utilisateur le dernier point de repère envoyé par CDPB et suscite une réponse de confirmation de la part de ce dernier.

CDRP			RECOVERY POINT REFERENCE NUMBER	M

* RDRPN :
Accusé réception par lequel l'utilisateur indique une discordance entre les points de repère. Une reprise doit être effectuée (CDR) à partir du point de repère joint comme paramètre.

RDRPN			RECOVERY POINT REFERENCE NUMBER	M

- Primitives d'interruption et resynchronisation.

* CDRPR :

Commande qui indique à l'utilisateur que la rupture va s'effectuer à partir du point de repère joint à la commande.

CDRPR			RECOVERY POINT REFERENCE NUMBER	M

* RDRPR :

Accusé réception par lequel l'utilisateur autorise la reprise.

RDRPR			RECOVERY POINT RE-FERENCE NUMBER	M

* CDR : Command Document Resynchronize.
Commande qui indique à l'utilisateur la fin prématurée du document. Cette fin prématurée n'est pas due à une erreur à l'émission.
Une reprise du document est envisageable.

CDR			REASON (DOCUMENT)	NM

* RDRP : Response Document Resynchronize Positive.
Accusé réception positif à la commande CDL.

RDRP			-	-

- Primitive de continuation de transmission.

  * CDC : Command Document Continue.
    Commande qui indique à l'utilisateur la poursuite de la transmission d'un document déjà partiellement transmis.

CDC	DOCUMENT LINKING (ONLY REQUIRED IF LINKING IN A NEW SESSION)	M	DOCUMENT REFERENCE NUMBER	N
			CHECKPOINT REFERENCE NUMBER	M
			TERMINAL IDENTIFIER OF THE CALLING TERMINAL	M
			TERMINAL IDENTIFIER OF THE CALLED TERMINAL	M
			DATE AND TIME	M
			ADDITIONAL SESSION REFERENCE NUMBER	NM
			SERVICE INTERWORKING IDENTIFIER	NM
			DOCUMENT TYPE IDENTIFIER	NM
			DOCUMENT REFERENCE (CURRENT SESSION)	M
			OTHER PARAMETERS OF CDS	NM
	PRIVATE USE	NM	–	–

- Primitives d'interruption et rejet.

  * CDD : Command Document Discard.
    Commande qui indique à l'utilisateur la fin prématurée du document. L'expéditeur est responsable, une reprise n'est pas envisagée.

CDD			REASON (DOCUMENT)	NM

  * RDDP : Response Document Discard Positive.
    Accusé réception positif à la commande CDD.

RDDP			–	–

3.3.3.2.3. Primitives appartenant à la phase de déconnexion.

- Primitive de refus de document.

  * RDGR : Response Document General Reject.
    Indique qu'une erreur de procédure s'est produite et provoque la procédure de rejet du document et une libération de la communication.

RDGR			REFLECT PARAMETER VALUES	M

- Primitives de fin.

  * CDE : Command Document End.
    Commande qui indique à l'utilisateur la fin du document.

CDE			CHECKPOINT REFERENCE NUMBER	M

* RDEP : Response Document End Positive.
Accusé réception positif à la commande de fin de document par lequel l'utilisateur indique qu'il n'a pas décelé d'erreur, qu'il a accepté la responsabilité du document reçu et qu'il est disposé à recevoir un nouveau CDS.

RDEP			CHECKPOINT REFERENCE NUMBER	M

3.3.3.2.4. Primitives de changement de sens.

* CDCS : Command Document Change Sens.
Dans l'échange alterné, cette primitive indique au partenaire qu'on lui donne la main et qu'il peut transmettre à son tour.

* RDCS : Response Document Change sens.
Accusé réception positif à la commande de changement de sens.

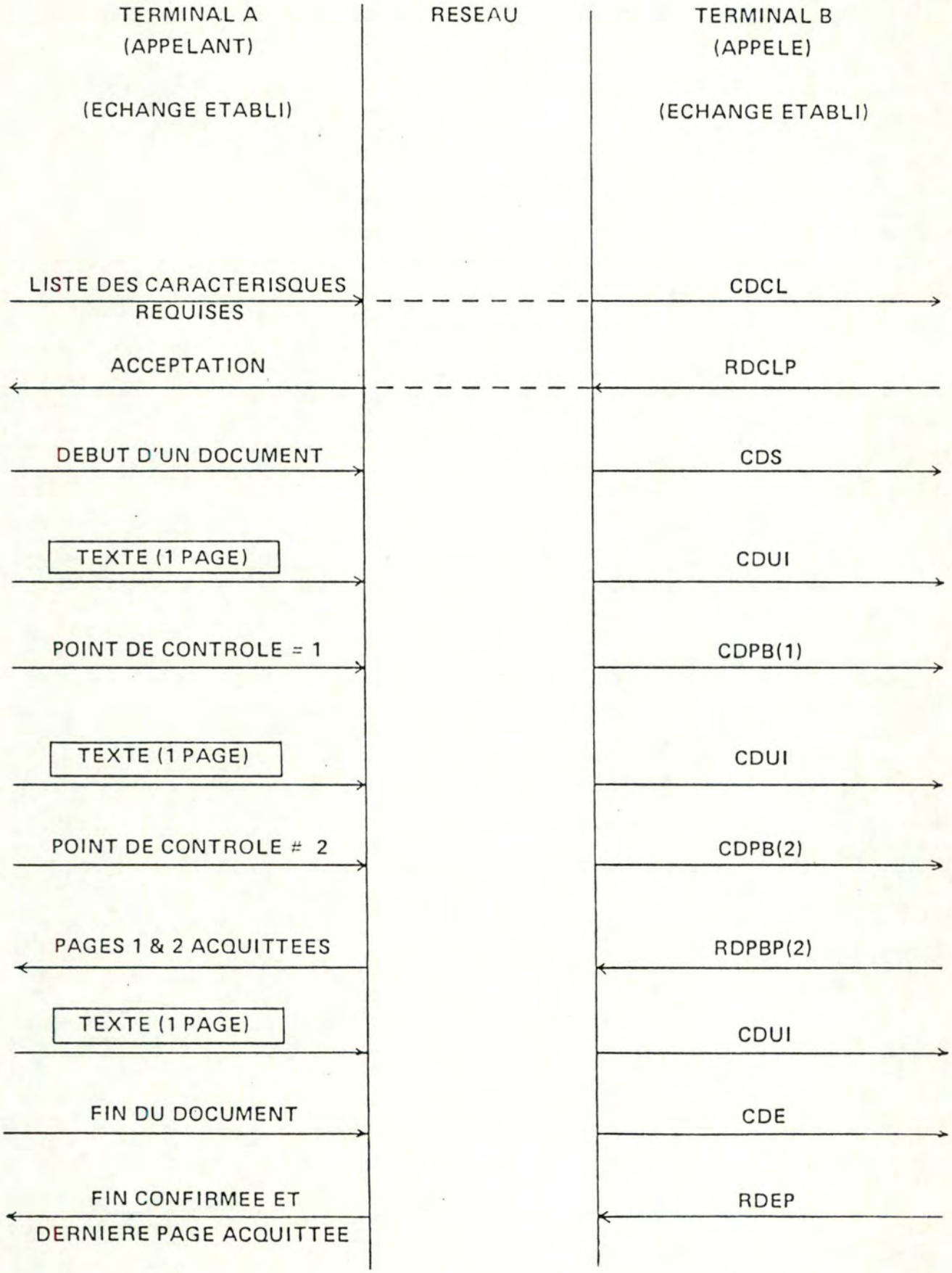
3.3.4. Exemple d'échange de primitives.

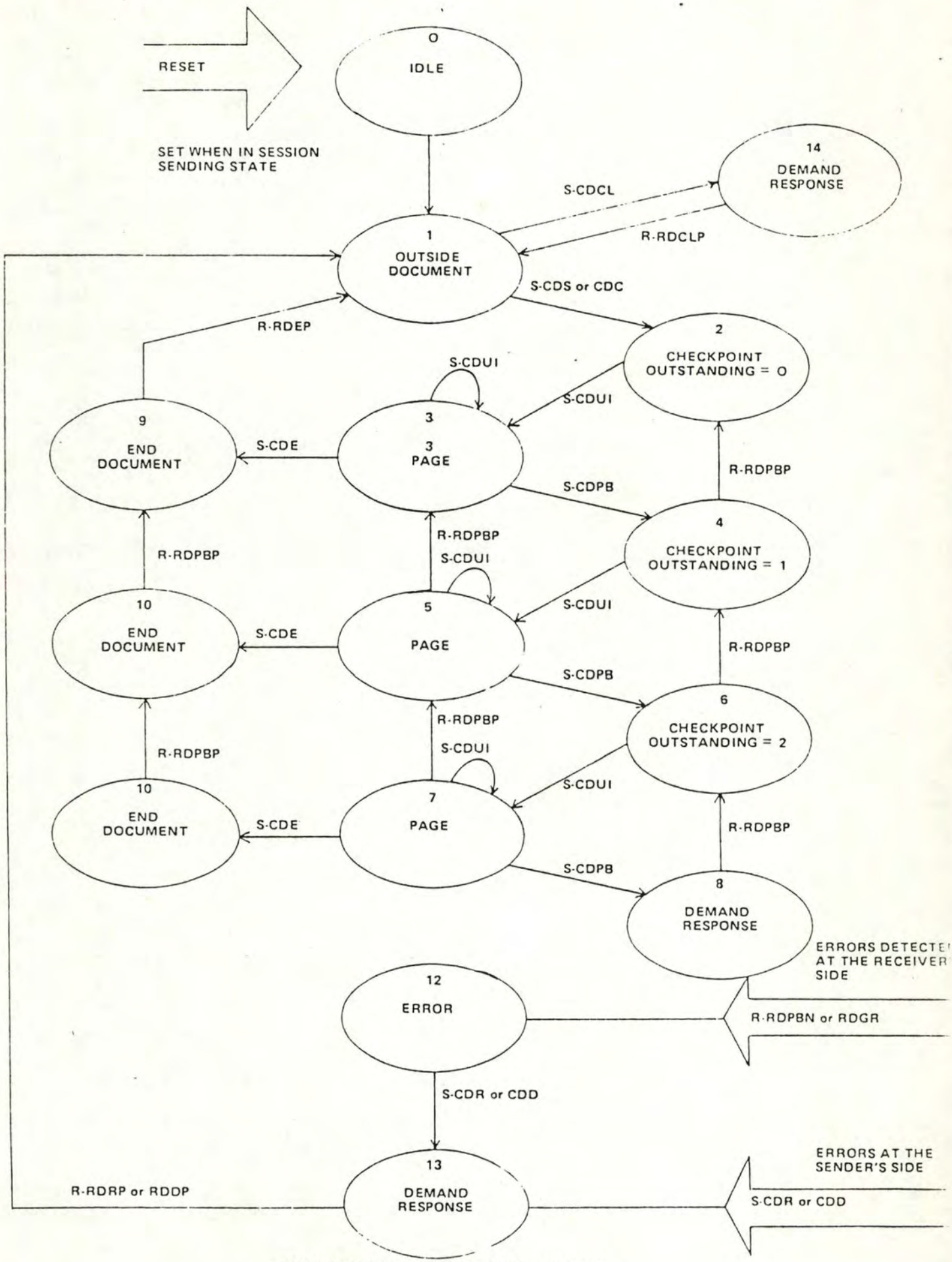
Les schémas suivants montrent un exemple d'échange de primitives dans le cadre d'une simple transmission de document depuis un terminal appelant A vers un terminal appelé B.

CAS 1. Transmission sans anomalie.

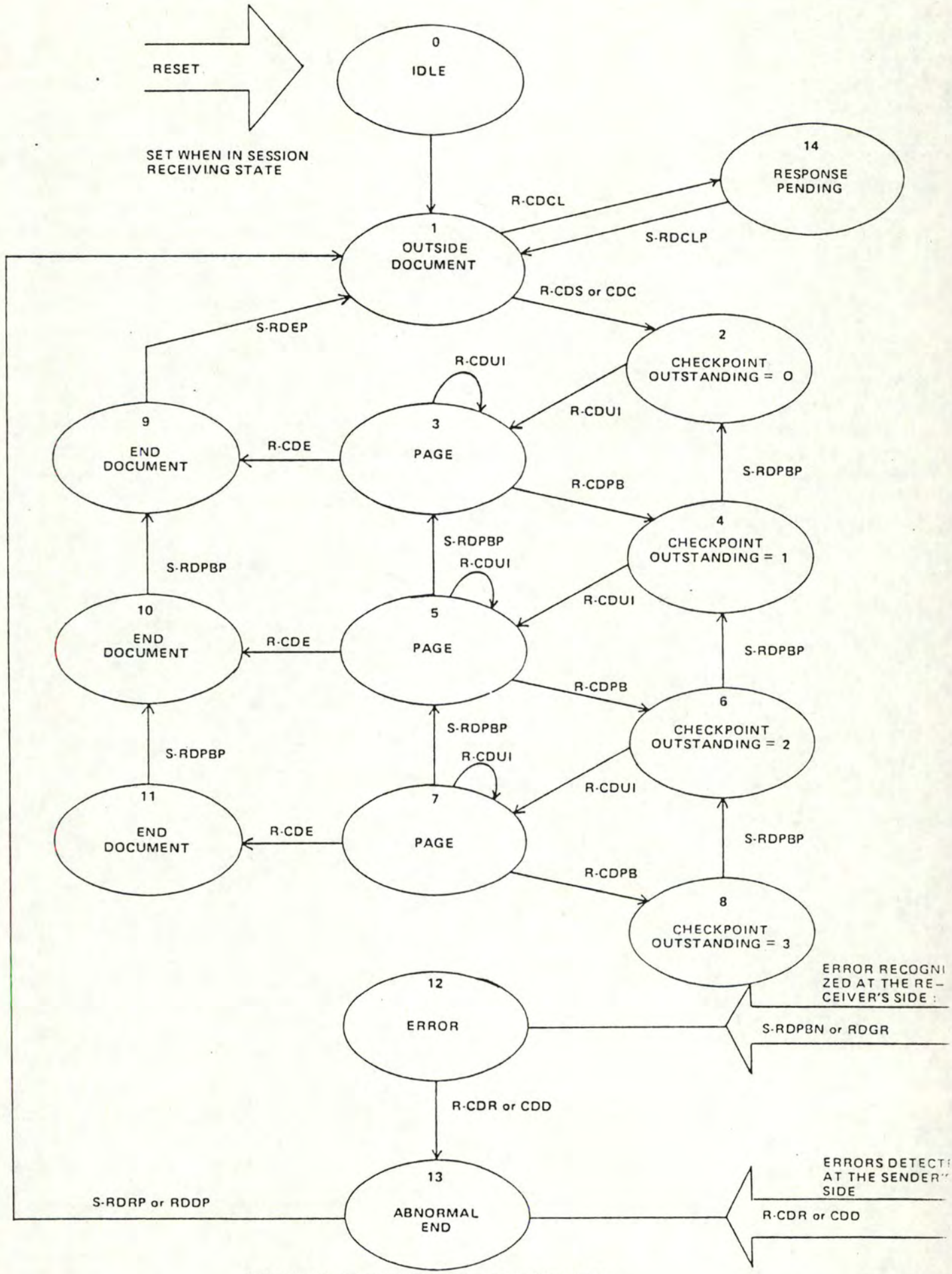
Voir pages 40, 41 et 42.

TRANSMISSION SANS ANOMALIE



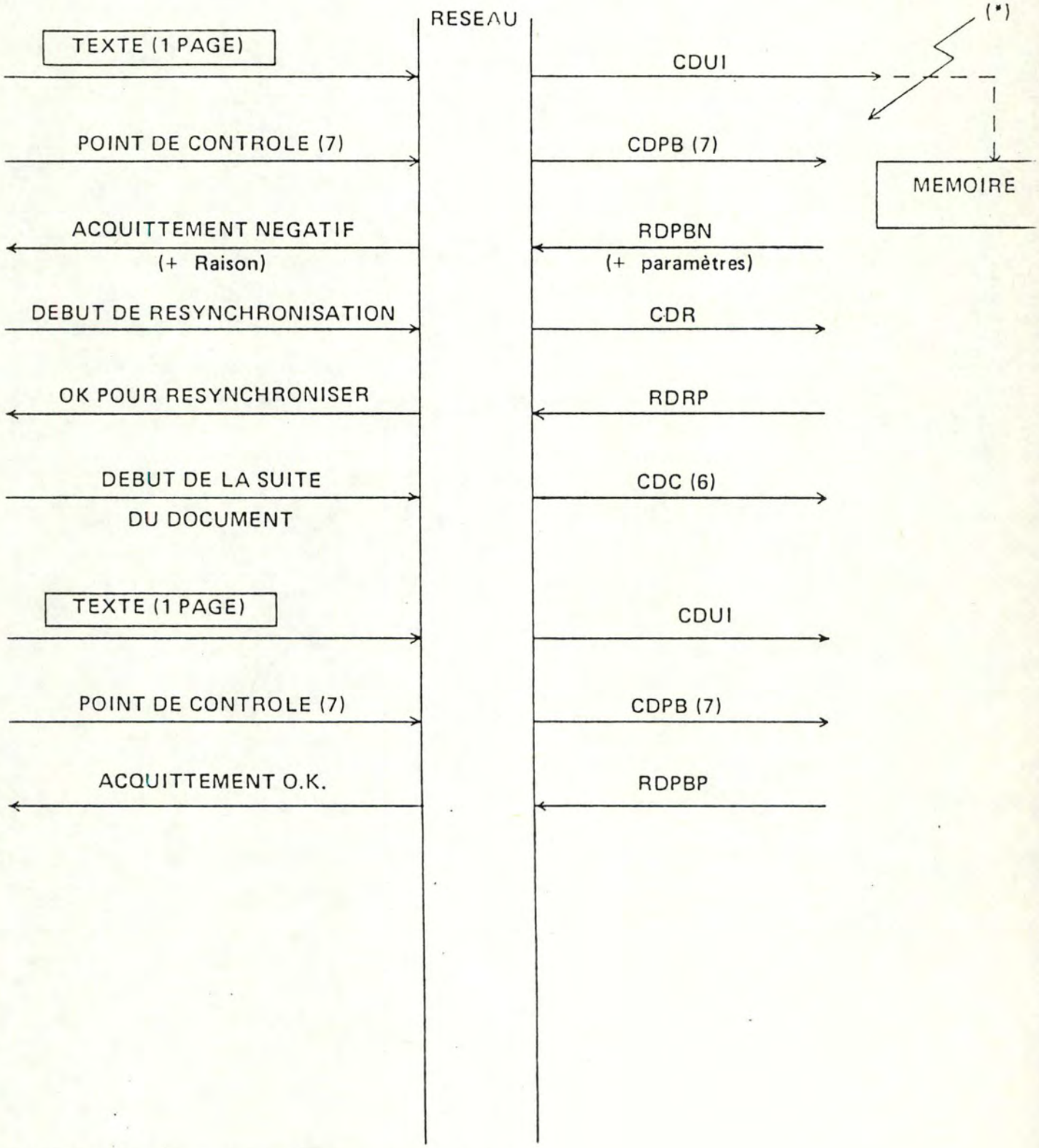


DOCUMENT STATE DIAGRAM FOR A WINDOW SIZE OF 3(SENDING PROTOCOL)



DOCUMENT STATE DIAGRAM FOR A WINDOW SIZE OF 3 (RECEIVING PROTOCOL)

CAS 2 : Transmission avec anomalies.



(*) CAUSES EVENTUELLES DE DEFAILLANCE

- MEMOIRE SATUREE
- RUPTURE DE SEQUENCE
- ERREUR LOCALE DU TERMINAL

3.3.5. Les primitives d'intercouches.

Outre les primitives que nous venons de citer et qui sont des primitives de dialogue horizontal, il en existe qui permettent, au sein même d'une entité qu'elle soit appelante ou appelée, d'intégrer le niveau présentation (couche 6) dans l'ensemble du modèle OSI.
Ces primitives jouent un rôle dans un dialogue intercouche "vertical".
Elles permettent le déroulement du protocole d'intercouche qui reprend à son compte certaines fonctions comme :

- La connexion intercouche.
- La libération intercouche.
- La confirmation au niveau i+1 de ce qui a été négocié au niveau i. En effet, chaque couche négocie les paramètres pour la couche adjacente supérieure et lorsque ces paramètres sont confirmés, ils sont utilisés par cette dernière dans le protocole des échanges "horizontaux".
- Gestion du passage des données entre les couches.

La description de ces primitives et leur gestion ne seront pas abordées dans le cadre de ce travail.

3.4. L'EMULATION ET LA NEGOCIATION.

Les besoins des applications et les caractéristiques disponibles sur le/les terminaux réels utilisés peuvent être très variés. Suite à cela, il existe deux alternatives :

- empêcher radicalement tout dialogue à la moindre incompatibilité, ce qui présente peu d'intérêt.
- permettre le dialogue malgré certaines incompatibilités.

Répondre à cette dernière alternative impose de compenser et/ou gérer les différences entre partenaires par l'adjonction de diverses fonctions dans le sous-module émulateur déjà défini en 3.3.3.1. et demande, préalablement à tout échange de données, de se mettre d' accord sur les caractéristiques à retenir pour la conversation. Cet accord est recherché au cours d'une phase de négociation impliquant l'échange de primitives de négociation entre le terminal et l'application.

La suite du paragraphe va s'attacher à décrire les principes de la négociation en relation avec les classes de terminaux virtuels définies en 3.2., les différents types d'émulation et enfin les deux schémas de phase de négociation possibles.

3.4.1. Les principes de la négociation.

Le fait d'avoir défini plusieurs terminaux virtuels permet de négocier en bloc l'ensemble des caractéristiques retenues dans la définition de ce terminal virtuel particulier.

Il est possible, en plus, de négocier individuellement une ou plusieurs caractéristiques supplémentaires.
Ceci permet de se mettre d'accord sur l'emploi d'un terminal virtuel intermédiaire entre deux terminaux virtuels de base
Le nombre de fonctions utilisables par les deux parties négociantes est accru.
Enfin , il existe des caractéristiques susceptibles d'être négociées individuellement bien qu'elles n'aient aucun rapport direct avec la subdivision des terminaux virtuels en classes.

Par exemple: les dimensions de la page, le nombre de fonctions programmables utilisables, la position des tabulations...

3.4.2. L'émulation et l'émulateur.

Nous pouvons distinguer trois principes totalement différents pour réaliser l'émulateur :

- l'émulation dite vers le bas,
- l'émulation dite vers le haut,
- l'émulation orientée application,

Dans chaque cas, il faudra toujours considérer deux types de négociation :

- le type négociation "application - terminal"
- le type négociation "terminal - terminal"

3.4.2.1. L'émulation dite vers le bas.

3.4.2.1.1. La négociation application - terminal.

Pour ce type de négociation, la classe résultat est celle du terminal qui a créé l'application. Elle constitue le minimum en dessous duquel tout échange de données est impossible.

Nous disposons, par exemple, d'une application qui nécessite un terminal d'une classe donnée . Si le terminal utilisateur appartient à une classe inférieure à celle de l'application, tout échange de données est impossible, tandis que s'il appartient à une classe supérieure ou égale, les échanges de données sont possibles sur base de la classe la plus faible.

3.4.2.1.2. La négociation terminal - terminal.
****

Pour ce type de négociation, la classe résultat est la classe minimale entre les deux terminaux en présence.

3.4.2.1.3. Le traitement des fonctions.
****

En plus de cette négociation globale sur les classes, on négocie individuellement certaines fonctions qui n'appartiennent pas aux fonctions de base de la classe. Celles-ci ne peuvent être utilisées que si elles sont réalisables sur les deux terminaux en communication directe ou via l'application.

3.4.2.1.4. Le traitement du format.
****

Le problème du format est résolu simplement en pliant toutes les lignes qui sont trop longues pour être imprimées sur une seule ligne du terminal récepteur.

3.4.2.1.5. Le rôle de l'émulateur.
****

Dans ce cas, l'émulateur est simple puisqu'il ne contient que deux logiciels dont les rôles sont :
- convertir le code particulier en un code standard et réciproquement,
- réaliser le pliage des lignes.

3.4.2.1.6. Les avantages et les inconvénients du principe.
****

Les avantages :
+ l'adaptation aux terminaux est simple et peu coûteuse puisque l'émulateur est élémentaire.
+ le vecteur de négociation est totalement indépendant du type d'application traitée (pour plus de détails voir le troisième principe).

Les inconvénients :

+ aucune application ne pourra tourner sur un terminal appartenant à une classe inférieure.
+ deux terminaux qui appartiennent à la même classe ne pourront pas nécessairement utiliser toutes les possibilités dont ils disposent.
+ il est impossible à une application d'utiliser la position du curseur sur un terminal, si le module de pliage des lignes est en fonction, car la position du curseur sur l'écran et celle qu'elle devrait avoir dans l'application sont différentes.

3.4.2.2. L'émulation dite vers le haut.

3.4.2.2.1. La négociation application - terminal.

Pour ce type de négociation, la classe résultat est celle du terminal qui a créé l'application comme dans le cas précédent. Ici, cependant, cette classe ne constitue plus une limite en dessous de laquelle tout échange est impossible.

Remarque : Tout ce qui peut empêcher la communication est un manque hardware qu'un ajout de logiciel ne peut combler.
Citons entre autre :

- l'impossibilité pour un TTY de remonter à la ligne précédente
- l'impossibilité pour un TTY d'effacer un caractère
- pas de touches fonctions sur un terminal
- ...

3.4.2.2.2. La négociation terminal - terminal.
****

Pour ce type de négociation, la classe résultat est la classe maximale entre les deux terminaux en présence. La remarque énumérée en 3.4.2.2.1. est aussi applicable ici.

3.4.2.2.3. Le traitement des fonctions.
****

En plus de la négociation globale sur les classes, on négocie individuellement certaines fonctions. Celles-ci peuvent être utilisées si les deux parties en disposent ou si une des deux en dispose et que l'autre peut les émuler à partir des fonctions plus élémentaires qu'elle possède.

3.4.2.2.4. Le traitement du format.
****

Le problème du format est résolu en général par des techniques plus élaborées que le pliage(2depart:4.3.2.3.) Néanmoins, cette dernière peut toujours être utilisée.

3.4.2.2.5. Le rôle de l'émulateur.
****

L'émulateur doit compenser, dans le terminal le plus faible, toutes les fonctions nécessaires pour se trouver dans la classe supérieure. De plus, dans cette même classe, il doit pouvoir réaliser des fonctions présentées sur un terminal et pas sur l'autre. En résumé, en plus des modules énumérés dans le principe d'émulation dite vers le bas (3.4.2.1.5.), l'émulateur doit :

- Réaliser toutes les fonctions softwares manquantes à une classe pour accéder à la classe la plus évoluée: la classe mode Data Entry.
- Réaliser les traitements de formatage des données de manière plus performante que le seul pliage des lignes
- Réaliser toutes les fonctions individuelles manquantes

3.4.2.2.6. Avantages et inconvénients du principe.
****

Les avantages :

+ Accroissement des possibilités des terminaux appartenant aux classes les plus faibles et dans une moindre mesure, celles des terminaux appartenant aux classes les plus évoluées (ajout de certaines séquences de contrôle).

+ Augmentation très forte du nombre de relations possibles entre "application - terminal" et "terminal-terminal". En effet, les terminaux de classes faibles peuvent travailler avec des terminaux de classes plus évoluées pour autant que les règles de négociation le permettent.

+ Toutes les possibilités d'un terminal sont exploitables si elles peuvent être émulées par logiciel sur un autre.

+ Grâce au module de traitement de format, le problème de positionnement du curseur peut être résolu pour les terminaux vidéo.

Les inconvénients :

+ Le coût à supporter par les terminaux des classes faibles peut être important par rapport à leur prix d'achat.

+ Le logiciel étant plus important dans ce principe que dans le précédent son coût en sera plus élevé.

+ On ne tient toujours pas compte du type d'application que l'on va traiter.

3.4.2.3. L'émulation orientée application.

Ce principe n'apporte une amélioration au précédent que dans le cas d'une communication entre application et terminal.

3.4.2.3.1. Principe.

Le vecteur n'est plus unique comme précédemment mais il diffère pour chaque application : créé en même temps que celle-ci, il regroupe seulement les fonctions et le format nécessaire au déroulement normal de l'application.

3.4.2.3.2. Exemples.

Prenons deux exemples pour illustrer ce principe :

A - Soit un terminal disposant de 16 touches de fonctions programmables. Dans les deux premiers principes, le vecteur de négociation contenait un paramètre indiquant la présence de ces 16 touches. Comme celles-ci ne sont pas réalisables par émulation, elles peuvent être la cause d'une incompatibilité lors de la négociation même si l'application ne les utilise pas.
Dans ce cas, par contre, on assigne au paramètre la valeur nulle lors de la création de l'application, et on rend cette dernière accessible à des terminaux ne possédant pas de touches fonctions programmables.
Chaque paramètre du vecteur reçoit ainsi une valeur qui tient bien compte du type de l'application traitée et plus du terminal qui a créé l'application.

B - Supposons un terminal très performant de la classe Data Entry sur lequel nous réalisons une application de traitement de texte.
Soit un télétype qui veut utiliser cette application
Si nous utilisons le premier principe, la communication est impossible car le télétype appartient à une classe inférieure à Data Entry.

Si nous utilisons le second principe, la communication est encore impossible car le télétype ne peut réaliser certaines fonctions hardware nécessaires pour émuler la classe Data Entry. (cursor adress).

Par contre, si nous utilisons le troisième; la communication est possible car aucune des possibilités (attributs, curseur) ne sont employées par une application de type traitement de texte.

3.4.2.3.3. Les avantages et inconvénients du principe.
****

Les avantages :

En plus des avantages déjà énoncés au principe 2, nous avons encore :

+ Le principe n'est pas plus complexe que le précédent car seul le vecteur et certaines règles de négociations changent.

+ Il assure un accroissement de liaisons possibles sans ajout de logiciel à l'émulateur.

Les inconvénients :

+ Il faut une interaction entre la personne qui crée l'application et la station VTP pour remplir le vecteur de négociation à chaque application.

3.4.3. Phase de négociation des caractéristiques du terminal virtuel.

Nous allons décrire deux schémas de négociation possibles.

3.4.3.1. Schéma dissymétrique.

Ce schéma a été introduit dans le protocole terminal virtuel de TELENET et repris pas H. ZIMMERMANN et N. NAFFAH.

Deux cas peuvent se présenter :

A - Le premier est celui où l'application présente une certaine flexibilité qui lui permet de s'adapter à certaines caractéristiques du terminal virtuel. L'échange des primitives "demandes de possibilités" et "voici mes possibilités" (voir fig. 3.4.3.1.a.) permet précisément à l'application de prendre connaissance des caractéristiques du terminal virtuel. La négociation s'achève par l'échange des deux primitives "voici nos paramètres" et "réponse" permettant à l'application d'exprimer ses desiderata et au terminal virtuel d'exprimer son accord ou son désaccord.

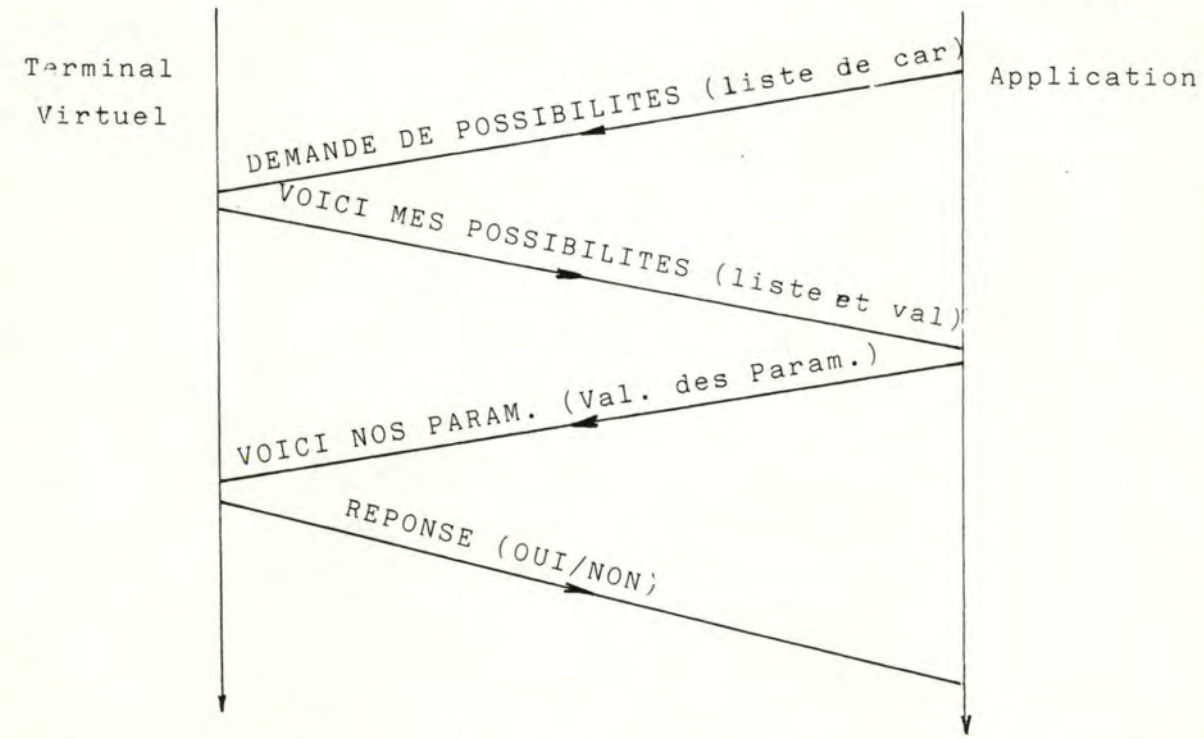
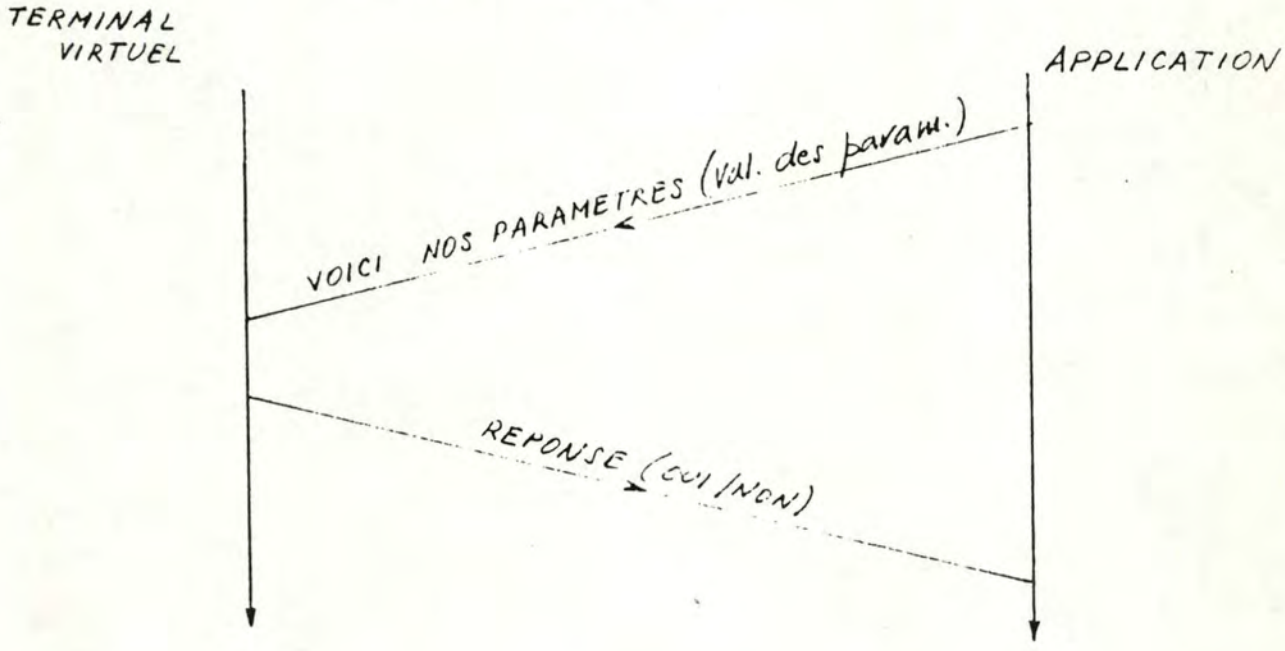


fig. 3.4.3.1.a : Schéma de négociation dissymétrique (1er cas).

B - Le deuxième cas est celui où l'application ne désire pas (ou n'a pas la possibilité de) s'adapter aux caractéristiques du terminal virtuel.
Dans ce cas, la négociation se réduit à l'échange des deux primitives "voici nos paramètres" et "réponse" (voir fig. 3.4.3.1.b.).



fig; 3.4.3.1.b : Schéma de négociation dissymétrique (2ème cas).

Ce schéma de négociation convient aussi bien pour la négociation initiale que pour une renégociation en cours de dialogue.

3.4.3.2. Schéma symétrique.
-.-.-.-.-.-.-.-.-

Ce schéma a été proposé par l'équipe EIN de Zurich .

Au contraire du schéma 1, le schéma de négociation proposé par SCHICKER et DUENKI est parfaitement symétrique. Son principe est le suivant : chaque partie envoie à son correspondant une primitive D-POSSIBILITES dans laquelle elle définit ses propres possibilités (sous forme d'une liste de caractéristiques avec leurs valeurs) : voir fig.3.4.3.2.

Sur réception de la primitive D-POSSIBILITES émise par son correspondant, chacune des deux parties examine individuellement si les deux listes de possibilités (la sienne et celle que son correspondant lui a transmise) sont compatibles de façon à construire une liste de paramètres communs.
Il faut évidemment obtenir que les listes de paramètres, auxquelles les deux parties aboutissent grâce à l'examen qu'elles effectuent indépendamment, concordent. Dans ce but, et afin de lever certaines indéterminations pouvant résulter de la comparaison des deux primitives D-POSSIBILITES, il est nécessaire de définir des règles de priorité précises.
Elles doivent être définies une fois pour toutes et acceptées par l'ensemble des applications et terminaux virtuels appelés à communiquer via le réseau.
A titre de vérification, SCHICKER et DUENKI proposent de faire suivre l'échange des primitives D-POSSIBILITES par un échange de primitives D-VERIFICATION grâce auxquelles chacune des deux parties peut communiquer à l'autre la liste de paramètres à laquelle elle a abouti.

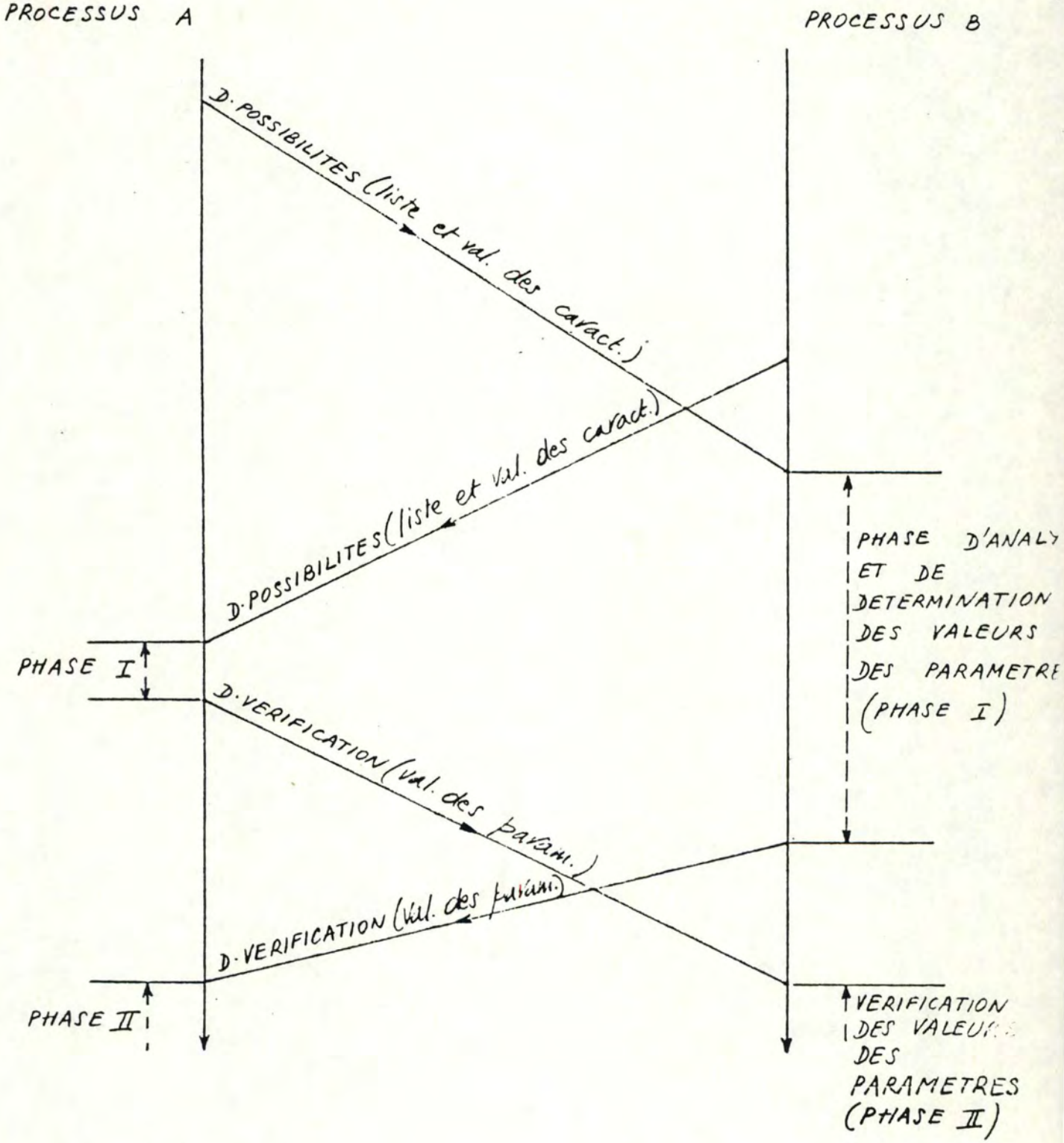


FIG. 3.4.3.2. Schéma de négociation symétrique

3.4.3.3. Avantages et inconvénients de chaque méthode.

Les schémas de négociation symétrique et dissymétrique présentent leurs avantages et leurs inconvénients. Nous en avons fait la liste

AVANTAGES DE LA METHODE

Schéma dissymétrique	Schéma symétrique
- la transmission de la liste des caractéristiques n'est pas toujours nécessaire. - les algorithmes de décision (détermination des paramètres) sont définis de manière locale.	- le schéma est applicable aussi bien dans le cas d'une communication entre processus identiques (terminal-terminal ou application-application) que dans celui de processu différents (terminal-appli cation)

INCONVENIENTS DE LA METHODE

- dans le cas d'une communi-cation entre processus identi-ques (terminal-terminal ou application-application), il est nécessaire de réintroduire artificiellement une dissy-métrie (par ex. : notion d'appelant-appelé).	- il est nécessaire d'envoye la liste complète des ca-ractéristiques. - le schéma suppose l'exis-tence de critères de décision définis au niveau du réseau.

3.5. ROLE DE LA STATION VTP

D'une manière générale, la station VTP a pour rôle d'organiser l'échange entre entités distantes, c'est-à-dire la gestion du dialogue des primitives conformément aux graphes de séquencement déjà évoqué dans l'exemple 3.3.3.3.

La station devra donc avoir, bien entendu, les possibilités de reconnaître les primitives, de valider d'une part leur opportunité et d'autre part les paramètres qui y sont associés, et enfin de prendre les décisions voulues, que le dialogue se passe avec ou sans anomalies.

D E U X I E M E P A R T I E.

I N T R O D U C T I O N.

INTRODUCTION A LA SECONDE PARTIE.

Remarques préliminaires : Définitions

1. Terminal créateur et terminal utilisateur.

Il est nécessaire dans un dialogue de pouvoir distinguer les deux terminaux en présence.

La première idée qui vient à l'esprit est d'appeler un terminal "récepteur" et l'autre "émetteur", mais puisqu'ils peuvent être à tour de rôle émetteur et récepteur, on ne peut prendre en compte cette idée.

Nous avons donc défini une autre notion s'appliquant dans les deux types de dialogue que nous avons considérés :
La notion de terminal "créateur" et "utilisateur".

- dialogue application-terminal.
  - un terminal est appelé terminal-créateur s'il a été utilisé pour réaliser l'application.
  - un terminal est appelé terminal utilisateur s'il utilise l'application.
- dialogue terminal-terminal.

  Pour éviter un traitement particulier lors de ce dialogue, nous imposons que :
  - le terminal qui demande la communication soit le terminal créateur.
  - le terminal qui est appelé dans la communication soit le terminal utilisateur.

Ce système permet de distinguer avec certitude les deux terminaux, où le terminal et l'application mis en présence.

D'après ce qui vient d'être défini, nous devons disposer dans la station VTP de deux vecteurs de négociation.

- un, rempli en fonction de l'application, pour le cas où le terminal est créateur.
- un contient toutes les caractéristiques du terminal pour le cas où le terminal est utilisateur. Ce vecteur est entré une fois pour toute dans l'adaptateur du terminal.

2. Les types d'applications.

Les définitions suivantes portent sur les différents types d'applications traités par notre système.

2.1 Le traitement de texte.

Une application est de type "traitement de texte" si elle contient un texte à envoyer dont les lignes sont justifiées à gauche et à droite à quelques exceptions près.
(pour plus de détails voir le module traitement texte aux chapitres IV et VIII).
Ce texte ne peut pas contenir de tableaux ou de figures.

La caractéristique de transmission sera la suivante :

- il n'y a qu'un seul sens de transmission de données du terminal créateur vers le terminal utilisateur.

2.2 Le "questions-réponses".

Une application est de type "questions-réponses" si elle contient un menu issu d'un terminal créateur et que, via le terminal utilisateur, on puisse, par la manipulation du curseur ou d'un crayon lumineux, pointer une zone choisie.

Il faut impérativement que les terminaux soient de type vidéo car ils sont les seuls à pouvoir utiliser le curseur ou le crayon lumineux.

La transmission se fait dans les deux sens :

- du terminal créateur vers le terminal utilisateur pour poser les questions sous forme d'un menu,
- dans l'autre sens pour envoyer les réponses.

Voici un exemple :

choisir la destination:

- Bruxelles
- Liège
- Namur
- Charleroi

Placer le curseur devant votre choix.
Enfoncer la touche ENTER une fois le choix réalisé.

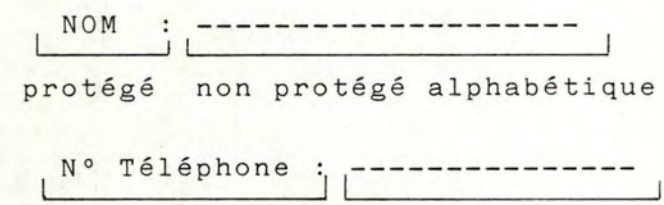
2.3 Les "zones formatées".

Une application est de type "zones formatées" si le terminal créateur a défini des zones protégées, qui ne sont pas accessibles à partir du clavier, et des zones non protégées où les seuls caractères admis, introduits à partir du clavier, correspondent à un type donné (voir annexe II - 8).

La transmission se fait dans les deux sens :

- du terminal créateur vers le terminal utilisateur pour envoyer l'écran formaté,
- dans l'autre sens pour envoyer l'écran rempli.

Voici un exemple :



2.4 Le texte libre 1.
-.-.-.-.-.-.-.-.-

Pour un dialogue application-terminal : une application est de type "texte libre 1" si elle ne peut répondre aux critères des trois premiers cas présentés.

La transmission peut se faire dans les deux sens.

2.5 Le texte libre 2.
-.-.-.-.-.-.-.-.-

Une application est de type "texte libre 2" si on a un dialogue entre deux terminaux.

La transmission peut se faire dans les deux sens.

Démarches suivies dans notre solution.

Après avoir posé les bases théoriques de notre travail, un certain nombre de choix s'offrent à nous : le premier est de définir un langage standard.

Pour ce faire, nous avons recueilli une documentation complète sur les terminaux les plus représentatifs du parc actuel. Nous les avons répartis, ensuite, d'après leurs caractéristiques, dans les cinq classes de terminaux virtuels définis au paragraphe 3.2 de la première partie. (Le classement se trouve au chapitre I de la seconde partie).

Reprenant au moins un terminal par classe, nous avons dressé une liste complète des fonctions de contrôle existantes (annexe I) et, sur cette base, avons cherché une norme définissant et regroupant un maximum de ces fonctions.
Cette norme constitue notre langage standard (chapitre II).

Le second choix, qu'il nous est donné de faire, est celui de l'implantation de l'interface. Le chapitre III explicite le choix de la configuration adoptée et donne une brève description du matériel nécessaire à sa réalisation.

Il reste à ce stade de l'étude, tout le développement du logiciel à insérer dans l'interface. Il est à noter que ce logiciel peut se scinder en deux parties (une partie "logiciel émulateur" et une partie "logiciel station VTP") suite à la modification du schéma du terminal virtuel explicitée en 3.3.3.1. de la première partie.

Le logiciel émulateur.

Les modules qui constituent le logiciel émulateur ont été définis sur base, d'une part, du choix d'une émulation orientée application (voir 3.4.2.3., Première partie) et d'autre part, des types d'applications définis ci-dessus.

Ainsi :

- L'application "traitement de texte", qui envoie au terminal un texte justifié ayant une configuration donnée (paragraphes, titres,...) demande un module assurant la reconstitution de cette structure sur un format plus petit que celui d'origine : le module traitement de texte (4.3.2.2.).

  De plus, comme il est parfois nécessaire de convertir certains caractères accentués afin de rendre au texte sa forme originale, nous avons créé le module "jeu de caractères" (4.3.3.3.).

- L'application "questions-réponses" exige pour ses réponses, soit la connaissance de la position du curseur, soit des touches fonctions. Pour la gestion du curseur, nous avons créé le module "gestion écran" (4.3.2.1.).

  Pour les touches fonctions, un module du même nom se charge de leur programmation suivant les nécessités de l'application (4.3.3.2)

- L'application "zone formatée" nécessite différentes possibilités, à savoir :

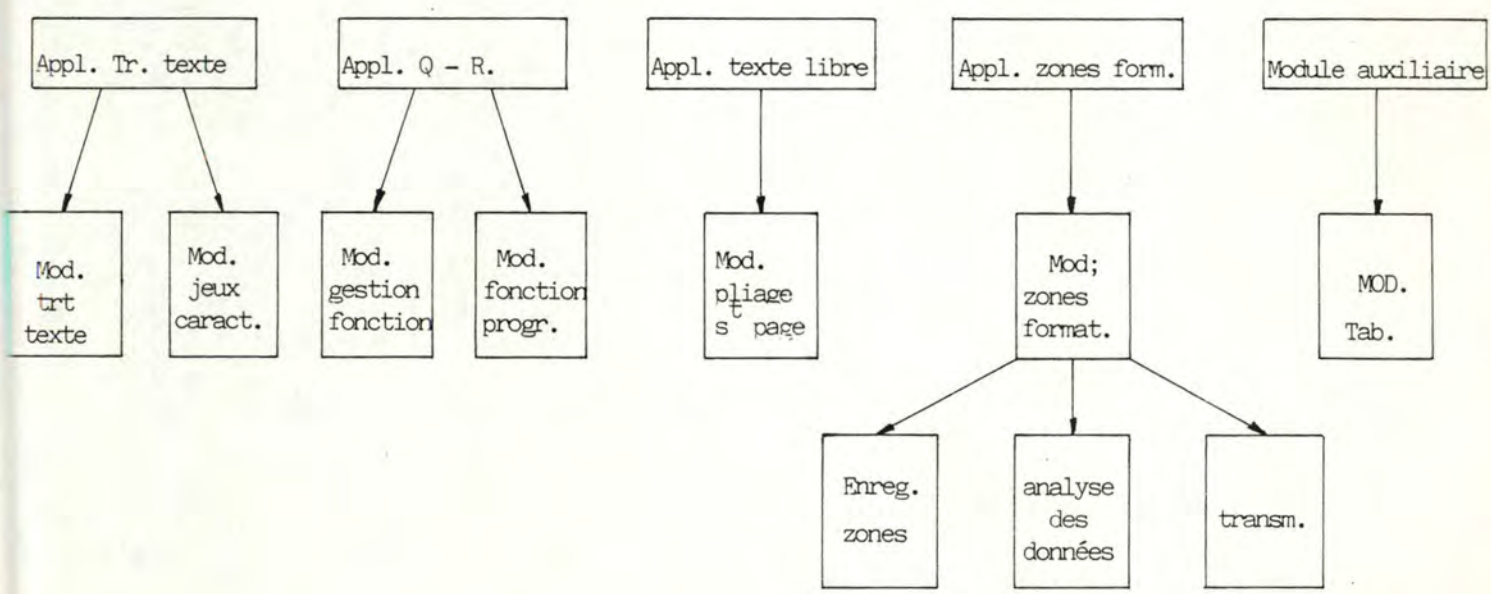
  + L'enregistrement de la position des zones et de leurs attributs
  + L'analyse des données introduites au clavier.
  + La transmission sélective des zones formatées.

  Toutes ces fonctions sont réalisées dans un module "zone formatée" (4.3.3.4.).

- L'application "texte libre" nécessite un module permettant la gestion simple des formats trop grands (4.3.2.3.)

- Enfin, un module de tabulation programmable a été introduit (4.3.3.1.).

Un résumé de tous ces modules est repris ci-dessous.



Le chapitre IV a donc pour but de décrire brièvement chacun de ces modules ainsi que le "convertisseur de code" (4.3.1), alors que le chapitre VIII s'attache à en donner une étude plus approfondie comprenant les entrées/sorties, traitements et algorithmes.
L'interaction entre les modules, quant à elle, est donnée, en fonction des types d'applications, au chapitre V.

Le logiciel "station VTP".

Le logiciel "station VTP"; est issu des derniers choix qu'il nous reste à effectuer suite à la première partie :

- le choix du principe de négociation
- le choix du type d'échanges.

Ainsi, une justification du principe de négociation utilisé se trouve au chapitre VI alors qu'une description très détaillée (définitions et traitements) de toutes les primitives retenues dans notre travail en tenant compte des types d'échanges constitue le chapitre VII.

Pour chacune de ces primitives, tous les paramètres sont décrits avec le maximum de précision afin de lever toute ambiguïté lors des traitements dans la station VTP.

C H A P I T R E I.

L E C L A S S E M E N T D E S T E R M I N A U X.

CHAPITRE I : LE CLASSEMENT DES TERMINAUX.

La première phase de notre travail consiste à classer les terminaux dans les classes définies précédemment en respectant les critères repris dans le tableau.

Rappelons le nom des différentes classes :

* classe scroll mode imprimante
* classe scroll mode télétype
* classe scroll mode vidéo
* classe mode page
* classe mode data entry.

Nous constituons une liste très incomplète des appareils vendus sur le marché mais ils sont les seuls pour lesquels nous disposons d'assez de renseignements pour les classer. Ils ont été choisis car ils sont les plus fournis par les constructeurs et de ce fait, les plus rencontrés chez les utilisateurs.

N.B. : ceux marqués d'une astérisque seront utilisés dans le chapitre suivant.

Tableau classement des terminaux.

1 - Scroll_mode imprimantes.

- BASF : 6603
- CENTRONIC : 150, 152, 154, 159, 351, 352, 353, 354, 358.
- DATA-GENERAL : 4323, 4324, 4325, 4326, 4327, 4328, 4363, 4364, 4365, 4366, 4518, 6190, 6191, 6192, MPT.
- DECISION-DATA : 6541-02
- DIABLO-SYSTEMS ; R0630, R0I650
- DIGITAL : LA 50, LQP02, LETTER PRINTER 100
- MICROLINE: 82A, 83A, 84, 92, 93
- NCR : 6440, 6441, 6442, 6455
- FACIT ; 4512*

- OLIVETTI : PR 320 PR1450 PR1470 PR1480 PR1490 PR2300
  PR2835 PR2890
- PHILIPS : P2931 P2934
- SIEMENS : 8122 9002
- TEKTRONIX : TEK 4643
- TELETYPE-CORP : 40 P 203
- TEXAS - INSTRUMENTS : RO 810

2 - Scroll mode télétypes.
- - - - - - - - - - - -

- DIABLO-SYSTEM : 1640, 1650
- DIGITAL : LA 34*, LA 100, LA 120*
- TELETYPE-CORP : MODELE 43, KSR 43A

3 - Scroll mode vidéos.
- - - - - - - - - - -

- REGENT 100*
- BEE HIVE : MICRO B 1*
- CITOM : CIT 101*
- DATA-GENERAL : D100, D200, MODEL 6110
- DIGITAL : UT 52*, UT 100*, UT 102
- HEWLETT-PACKARD : HP2621B, HP2629C
- HONEYWELL-BULL : DKU 7001
- OLIVETTI : WS 584, TUC 201
- NCR : 7900 MODEL 1, 7900 MODEL 3, 7900 MODEL 4
- VISA : 24

4 - Page mode vidéos.
- - - - - - - - -

- REGENT 200*
- DATA-GENERAL : D400, D450
- EXECUTIVE 80 : MODELE 20, MODELE 30
- HEWLETT -PACKARD : HP 2382A, HP 2622 A
- NCR : 7900 MODELE 2

- OLIVETTI : WS 580
- TELEVIDEO : 910, 912, 914*, 920*, 924*, 925*, 950*
- VISA : 35, 40

5 - Data-Entry-mode vidéos.

- BEEHIVE : MICRO B2
- HEWLETT-PACKARD : HP 2624 B, HP 2626A
- HONEYWELL-BULL : DKU 7005*, DKU 7007
- TELEVIDEO : 970*

C H A P I T R E II.

L A D E F I N I T I O N D U L A N G A G E S T A N D A R D.

CHAPITRE II : LA DEFINITION DU LANGAGE STANDARD.

La phase suivante consiste à dresser une liste la plus complète possible des caractéristiques rencontrées sur les terminaux vendus sur le marché.

Nous avons exclu de celles-ci toutes les séquences intéressant le traitement local des données, à savoir :

- les fonctions d'édition de texte
- plusieurs modes locaux
- attributs du curseur
- ...

ainsi que les séquences concernant la commande d'appareils périphériques comme les imprimantes, les mémoires de masse, les lecteurs optiques...

Nous présentons en annexe I, les séquences de contrôle des terminaux marqués d'une astérisque dans le tableau précédent. Ils sont les seuls pour lesquels nous disposons d'une documentation très complète où chaque séquence est définie en détails.

Ces séquences sont classées comme suit :

- les fonctions d'adressage de l'écran
- les attributs de visualisation des caractères
- les fonctions d'effacement
- les fonctions de tabulation
- les modes différents des transmissions
- les modes de transmission
- les jeux de caractères
- les attributs des zones formatées
- autres (format support, code, touches fonctions...)

En nous basant sur les définitions de la norme ANSI, nous présentons une définition et un codage précis d'un certain nombre de séquences énoncées en annexe I. Celles-ci sont retenues pour former le langage standard qui sera utilisé dans le dialogue entre une application et un terminal.
Ce langage se trouve décrit dans l'annexe II.

C H A P I T R E III.

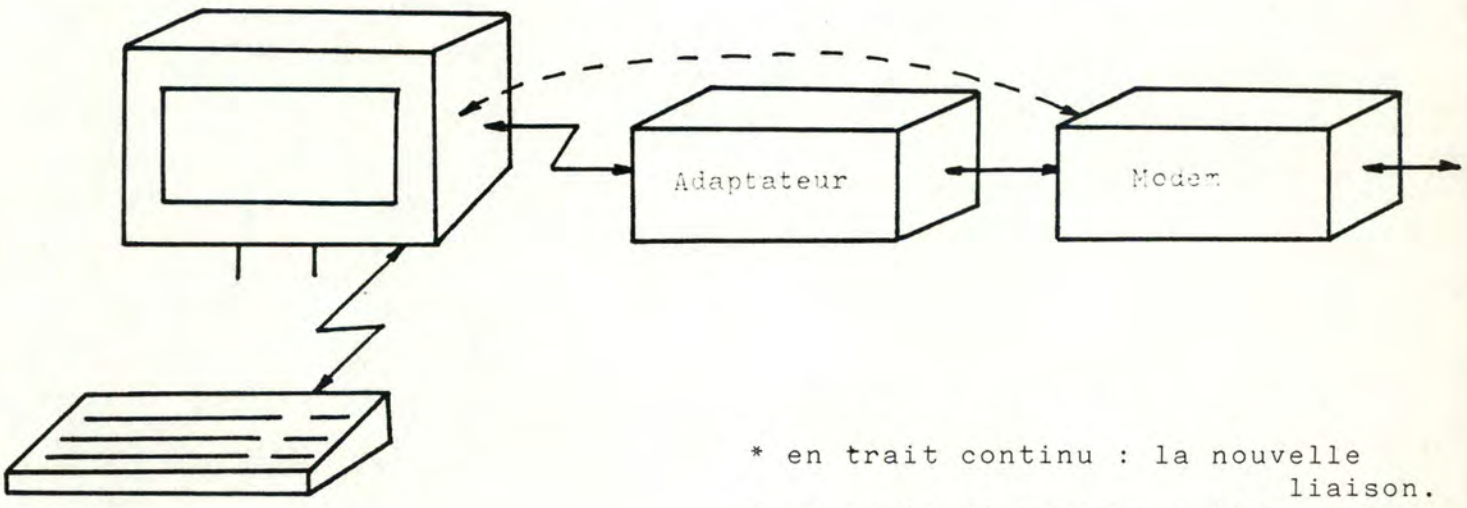
L E M A T E R I E L.

CHAPITRE III : LE MATERIEL.

3.1. LE CHOIX D'UNE SOLUTION.

Notre choix s'est porté sur la solution qui consiste à intercaler un boitier indépendant, contenant l'émulateur et la station VTP, entre le terminal et le modem.

SCHEMA



* en trait continu : la nouvelle liaison.
* en trait discontinu : l'ancienne liaison.

3.2. LES AVANTAGES ET LES INCONVENIENTS.

* Cette solution présente les avantages suivants :

  - elle ne dépend en aucune manière du type de conception du terminal, qu'elle soit mécanique, électronique câblée ou programmée.
  - elle est facile à réaliser car elle ne nécessite aucune modification du matériel existant : il y a une simple insertion d'un boitier d'interface entre le terminal et le modem (voir schéma).
  - elle ne nécessite qu'une seule carte hardware pour tous les terminaux, avec simplement des adaptations du logiciel du convertisseur de langage.

Ceci permet d'en réaliser un grand nombre et d'en diminuer le coût.

* Cette solution présente l'inconvénient suivant :

  - le dialogue entre le terminal et l'adaptateur se fait sur une ligne série. Ce qui ralentit le traitement des fonctions.

3.3. L'ETUDE RAPIDE D'UNE CARTE HARDWARE.

Cet adaptateur nécessite une carte électronique programmable qui contiendra :

* une unité centrale :

  - très rapide afin de réduire au maximum le temps de traitement dans l'adaptateur.

  - possédant un jeu d'instructions permettant :

    a) le transfert de blocs de données
    b) la manipulation facile des piles et des files
    c) la comparaison de chaînes de caractères
    d) le test de bits dans un byte.

    Ces instructions constitueront l'essentiel des traitements réalisés dans l'adaptateur.

* Une mémoire morte (EPROM) programmable.

  - pour mémoriser le logiciel de base de l'émulateur et de la station VTP.
  - nous la préférons à une mémoire morte (ROM) car une adaptation à chaque type de terminal est nécessaire au niveau du logiciel (convertisseur de code).

* Une mémoire vive (RAM)

  - pour mémoriser les variables utilisées par les différents modules constituant l'adaptateur et les tampons.
  - elle aura une capacité suffisante pour maintenir en mémoire toutes les variables lorsque tous les modules sont raccordés dans l'adaptateur.

* Deux sorties-entrées séries.

  - une pour la connexion terminal-adaptateur.
  - une pour la connexion modem-adaptateur.
  - elles seront programmables indépendamment afin de pouvoir répondre à toutes les vitesses de sorties des terminaux et des lignes.

3.4. CONCLUSION.

Nous n'avons fait, à ce niveau, aucune recherche concernant la conception d'une carte ou une étude de marché des cartes disponibles, bien que cela soit important et nécessaire pour réaliser l'adaptateur.

C H A P I T R E IV.

L' E M U L A T E U R.

CHAPITRE IV : L'EMULATEUR.

4.1. CHOIX D'UN PRINCIPE.

Notre émulateur travaille suivant le principe de l'émulation orientée application (voir 1ère partie 3.4.2.3.).

4.2. JUSTIFICATIONS.

* Ce principe permet de satisfaire le plus grand nombre de demandes de liaison entre une application et un terminal, ou entre deux terminaux.
* Comme il faut de toutes manières ajouter du logiciel au terminal pour gérer le protocole de la station VTP, il est intéressant de de créer un logiciel supplémentaire pour émuler toutes les fonctions manquantes.
* Le traitement local des informations avant leur émission et l'envoi de pages à la place de caractères isolés permettent une économie financière dans les transmissions sur le réseau.
* Le nombre des fonctions ajoutées au terminal et le nombre plus grand d'applications accessibles grâce à ces fonctions émulées atténuent le surplus de coût du logiciel.
* La réalisation d'une interface standard permet d'en diminuer le coût.

4.3. LES FONCTIONS DE L'EMULATEUR.

L'émulateur est composé d'un certain nombre de modules que nous allons décrire brièvement.

4.3.1. Le module convertisseur de code.

Le premier rôle de ce module est de transformer les séquences de contrôle propres au terminal en des séquences de contrôle définies dans le langage standard (annexe II) et inversement.

Toutes les séquences de contrôle sont filtrées. Une fois les éventuels paramètres mémorisés et la nature de la fonction déterminée, nous pouvons aux vues de ce qui a été négocié :

a) lors d'un transfert du terminal vers le réseau

- soit envoyer la séquence de contrôle standard correspondante.
- soit ne rien envoyer car :
  *ou bien la séquence de contrôle que nous tentons d'envoyer ne peut pas être reçue par l'autre entité et par conséquent il se produira une erreur.
  *ou bien la séquence de contrôle peut être reçue par l'autre entité mais elle n'aura aucun effet sur celle-ci et par conséquent il ne sert à rien de l'envoyer.

b) lors d'un transfert du réseau vers le terminal

- soit envoyer la séquence de contrôle propre au terminal
- soit émuler cette séquence de contrôle.

Le deuxième rôle que joue le convertisseur de code est celui de moniteur des autres modules.

Comme nous réalisons une émulation orientée application, il est nécessaire de créer un certain nombre de modules qui vont compenser la différence qui existe entre la classe "imprimante scroll mode" et la classe "data entry mode". Tous ces modules sont activés suivant les besoins lors de la négociation et gérés par le convertisseur (rôle de moniteur).

4.3.2. Gestion de format.

Ces modules assurent la gestion du support de visualisation. Ils ne sont activés que si le format du terminal utilisateur est plus petit que celui du terminal créateur. Une seule exception à cette règle: le module pliage qui est activé chez le créateur quand l'utilisateur travaille sur un support plus grand.

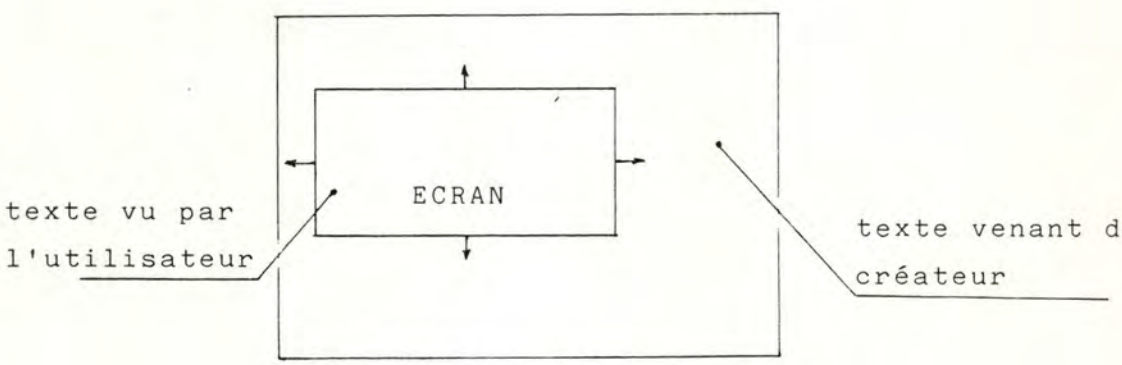
De plus chacun des modules est lié à un type d'application (voir chapitre V).

Ils sont mutuellement exclusifs c'est-à-dire qu'une application ne peut jamais en utiliser plus d'un à la fois.

4.3.2.1. Le module gestion d'écran.

Ce module est uniquement utilisable sur un vidéo.
Il permet de faire déplacer une fenêtre, matérialisée par l'écran sur le texte du créateur, grâce à des commandes simples.

On dispose d'une vue partielle du texte original.



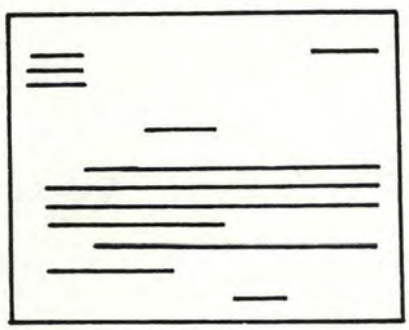
Ceci ne nécessite aucun reformatage du texte et il permet de plus, une gestion simple du curseur.
Celui-ci est aussi émulé car sa position n'est plus celle sur l'écran mais celle dans le texte venant du créateur.

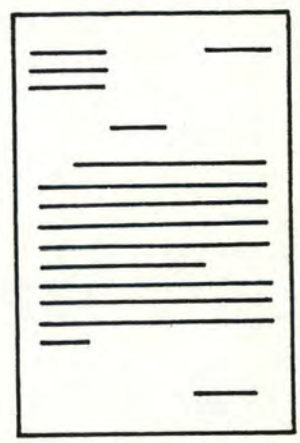
4.3.2.2. Le module de traitement de texte.

Si l'application est de type traitement de texte, nous activons ce module.

Il réalise le reformatage du texte qu'il reçoit du créateur. Nous retrouverons sur le support de visualisation du terminal utilisateur, le texte du créateur sous le même aspect, avec comme différence la longueur des lignes qui est devenue plus faible.

Ce module doit analyser le type de la ligne d'entrée (titre, entête, début de paragraphe,...) afin de les reproduire en sortie sous un format identique.



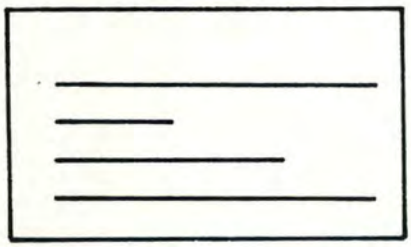


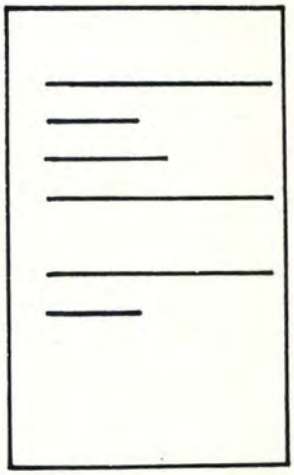
4.3.2.3. Le module de pliage et de saut de page.

Ce module réalise un pliage d'une ligne reçue si sa longueur excède la longueur du support visuel, en tenant compte d'éventuelles marges à gauche et à droite.

Il réalise en plus le saut à la page suivante si le nombre maximum de lignes sur la page, déduction faite des marges en haut et en bas, est atteint.

Il permet de voir tout le texte sans aucune manipulation.





4.3.3. Autres.

4.3.3.1. Le module de tabulation horizontale et verticale.

Ce module va mémoriser les différentes positions des "tabulations stops" pour un terminal qui ne peut pas réaliser cette fonction. Lors de l'utilisation de la séquence de contrôle des tabulations par l'utilisateur, l'émulateur avancera/reculera le curseur jusqu'à la position suivante/précédente mémorisée.

Ce principe est valable aussi bien pour les tabulations horizontales que verticales.

La première mémorisation des positions des "tabulations stops" s'effectue lors de la négociation et elles sont utilisées lorsque la séquence de contrôle de tabulation prise par défaut est activée.

4.3.3.2. Le module touches de fonctions.

Ce module mémorise un texte ou des commandes pour chaque touche de fonction que nous voulons programmer.

A chaque pression sur une touche de fonction, on va envoyer ce qui a été mémorisé pour cette touche.

Il est évident qu'on ne peut pas programmer des touches qui n'existent pas sur le clavier.

4.3.3.3. Le module jeu de caractères.

Si une application est écrite en français sur un clavier AZERTY avec des lettres accentuées il faut convertir le texte avant de l'imprimer sous peine de rendre le texte totalement illisible.

En effet, par exemple le "è" n'est pas codé comme un "é" en ASCII mais bien comme le symbole "{"

Le module va convertir le "é" par un "e" et rendre au texte son aspect original.

4.3.3.4. Le module zones formatées.

Ce module va permettre de valider dans le terminal les données qui sont introduites à partir du clavier.

Il va gérer des zones sur l'écran déterminant celles où on pourra écrire des données et les autres. Dans ces dernières, en fonction de leurs attributs, il va vérifier que la donnée est conforme. Dans l 'affirmative, il va l'enregistrer dans sa mémoire sinon il la rejette et attend une nouvelle donnée valide.

Il gère un certain nombre de séquences de transmission qui permettent d'envoyer de manière sélective les données mémorisées.

Ce module permet à un terminal scroll mode de devenir un terminal data entry mode.

C H A P I T R E V.

U T I L I S A T I O N D E S M O D U L E S.

CHAPITRE V : UTILISATION DES MODULES.

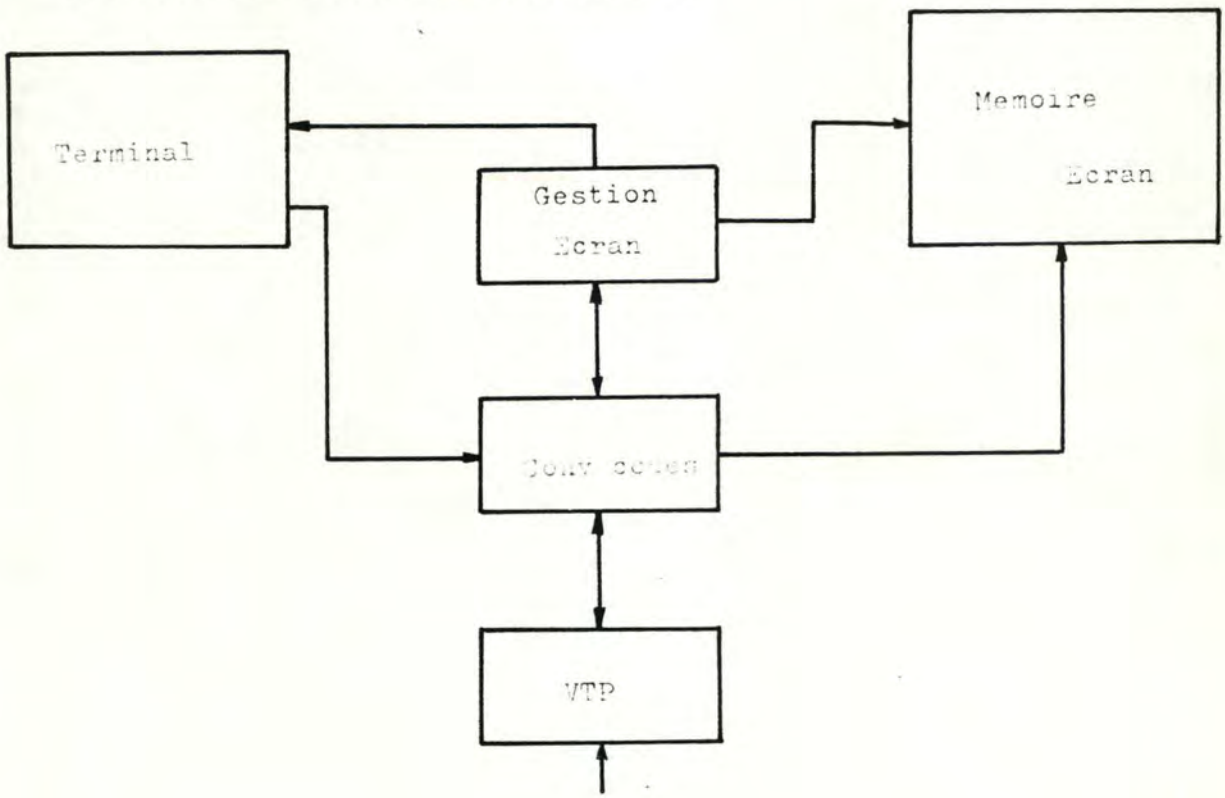
5.1. INTRODUCTION.

Nous allons étudier comment les informations venant soit du réseau via la station VTP, soit du terminal sont traitées en fonction des différents modules raccordés dans l'interface.

Nous allons reprendre chaque type d'application et brancher dans l'interface les modules dont elle a besoin pour fonctionner.

Les modules indépendants d'un type d'application seront ensuite décrits séparémment. Il est bien évident que la combinaison de ces modules avec ceux propres aux applications est possible mais une explication générale n'apporterait rien de plus si ce n'est une plus grande complexité.

5.2. L'APPLICATION QUESTIONS-REPONSES.



Deux cas sont à considérer :

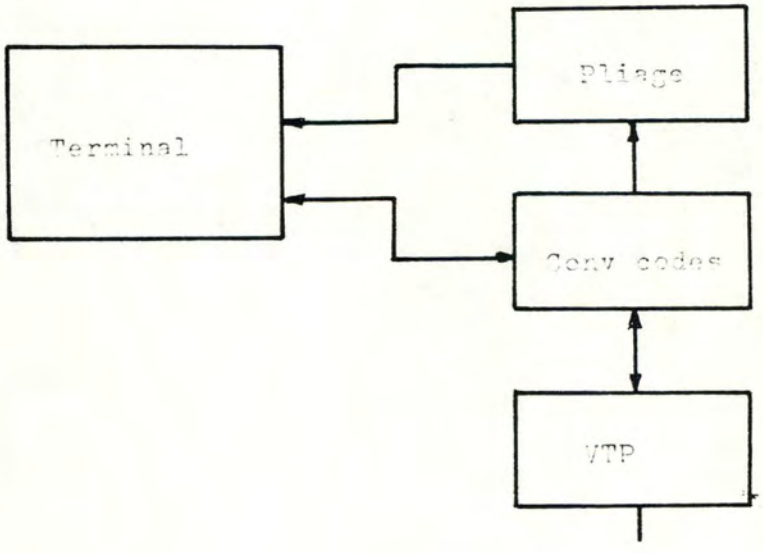
5.2.1. La réception des caractères venant du réseau.

+ la station VTP reçoit une page du réseau.
+ elle le signale au convertisseur de code.
+ celui-ci va mémoriser les caractères dans la mémoire écran et traiterles séquences de contrôle.
+ il appelle le module gestion écran pour positionner la fenêtre en fonction du curseur.

5.2.2. La réception des caractères venant du terminal.

+ le convertisseur de code reçoit la séquence de contrôle donnant la position du curseur au terminal.
+ Le convertisseur de code va lire la position du curseur fictif (position du curseur qui correspond à celle se trouvant dans l'écran du créateur).
+ il l'insère dans la séquence standard.
+ il la dépose dans la station VTP.
+ il lui signale que des données doivent être envoyées.

5.3. L'APPLICATION TEXTE LIBRE.



Deux cas sont à considérer :

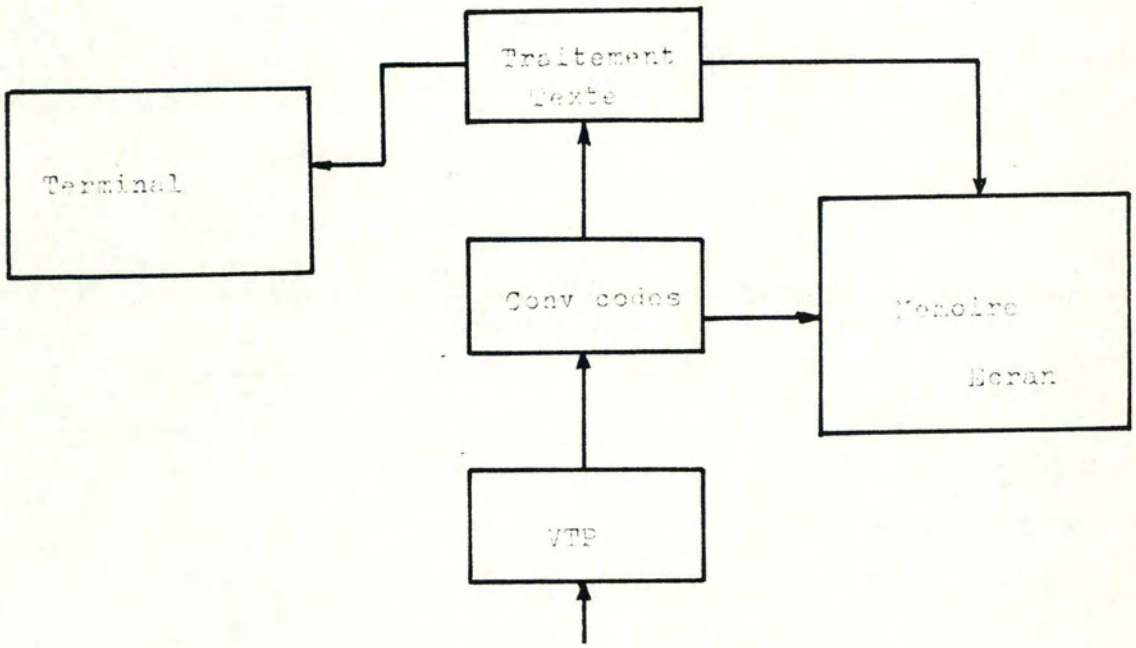
5.3.1. La réception des caractères venant du réseau.

+ réception d'une page dans la station VTP.
+ elle le signale au convertisseur de codes.
+ si le caractère traité est différent d'une séquence de contrôle, on l'envoie au module pliage - celui-ci va vérifier s'il faut ajouter un FF ou CR+LF avant de l'envoyer au terminal.
+ si le caractère traité est une séquence de contrôle le convertisseur va le convertir et l'envoyer au terminal.

5.3.2. La réception des caractères venant du terminal.

+ réception d'un caractère, venant du clavier, dans le convertisseur.
+ si le caractère est une séquence de contrôle, le convertisseur va le traduire en langage standard.
+ il le dépose dans la station VTP.
+ si le caractère est différent d'une séquence, le convertisseur le dépose dans la station VTP sans aucun traitement.

5.4. L'APPLICATION TRAITEMENT DE TEXTE.



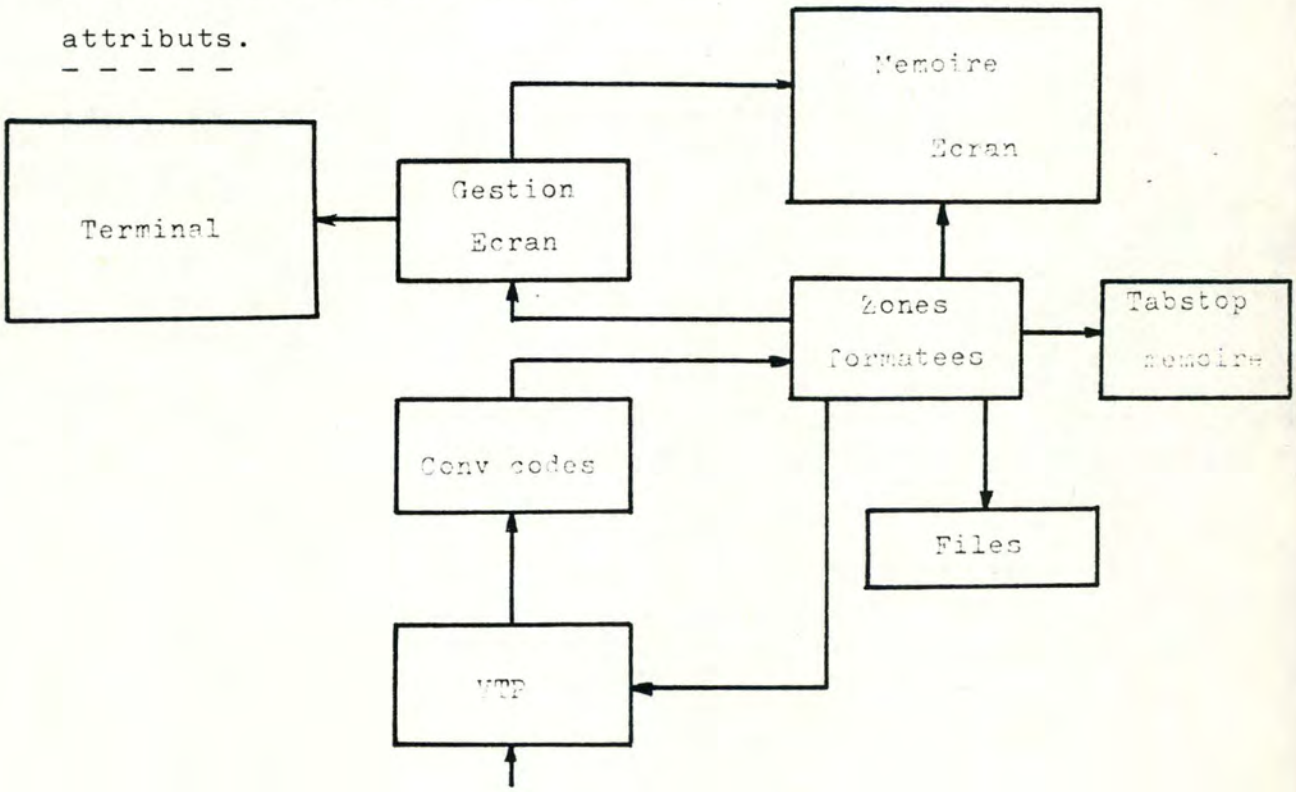
+ la station VTP reçoit une page du réseau,
+ elle le signale au convertisseur de code,
+ celui-ci met la page en mémoire,
+ il active le module traitement de texte,
+ ce dernier va traiter chaque ligne de la page,
+ il envoie au terminal les lignes reformatées.

N.B. : Il est évident que pendant que le module traitement de texte travaille sur des lignes, le module convertisseur de code peut continuer à remplir la mémoire.

5.5. L'APPLICATION ZONES FORMATEES.

Il faut discerner trois étapes dans ce type d'application :

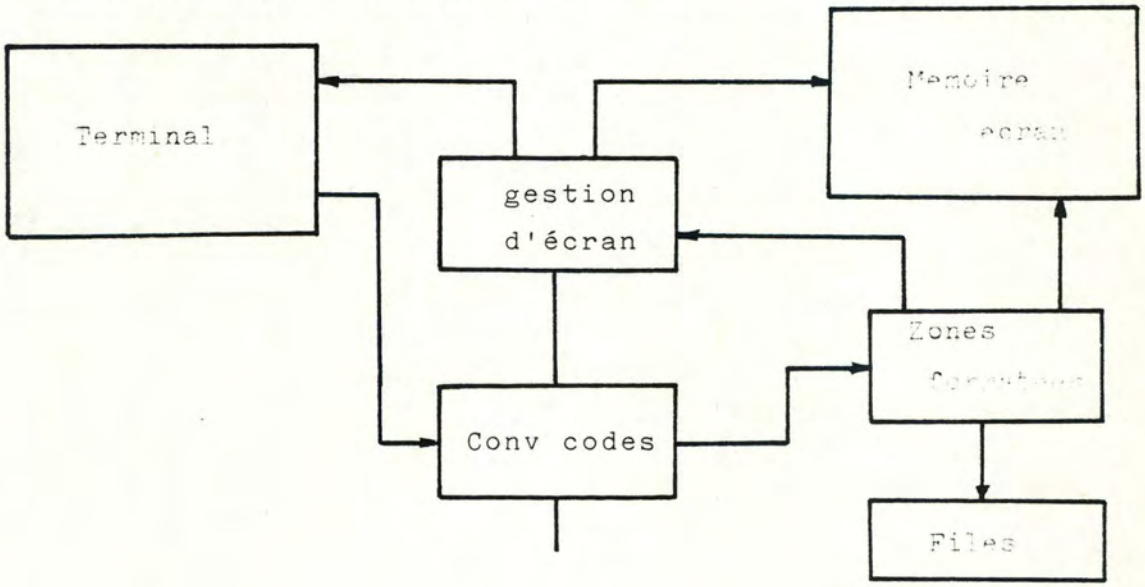
5.5.1. L'enregistrement de la position de chaque zone et de ses attributs.



+ la station VTP signale au convertisseur de code qu'il y a une page disponible en mémoire VTP,
+ le convertisseur de code détecte la séquence de contrôle déterminant le début d'un formatage d'écran,
+ le convertisseur de code analyse cette séquence,
+ il appelle le module "gestion zonées formatées",
+ celui-ci va prendre les caractères directement dans la mémoire de la station VTP,
+ en suivant le codage standard de chaque zone, il en mémorise dans une file, le début, la fin et les attributs,

+ il mémorise le début des zones non-protégées dans une zone mémoire utilisée pour les tabulations,
+ s'il existe du texte, il est envoyé dans la mémoire écran,
+ à chaque caractère ainsi mémorisé, il appelle le module "gestion écran" pour vérifier s'il faut ou non l'imprimer au terminal,
+ le module "zone formatée" rend la main lorsqu'il détecte la séquence de contrôle indiquant la fin du formatage.

5.5.2. L'écriture, à partir du clavier, des données dans les zones non-protégées avec leur validation.



5.5.2.1. Un caractère normal

+ Un caractère est reçu et analysé dans le convertisseur de code,
+ celui-ci active le module "gestion zones formatées",
+ ce dernier vérifie si on doit, ou non, accepter ce caractère et ce, en fonction des attributs de la zone où se trouve le curseur,

+ s'il est accepté

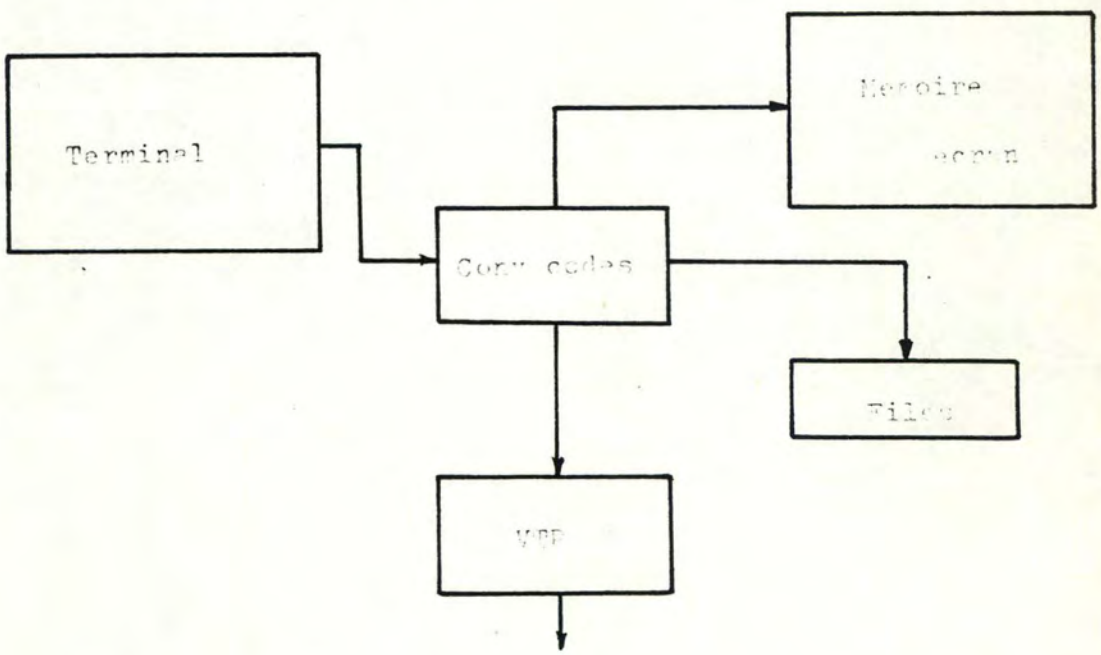
  -on l'envoie dans la mémoire écran
  -on appelle le module "gestion écran" qui va imprimer l'écran mis à jour,
  -on rend la main au convertisseur de code,

+ on attend le caractère suivant.

5.5.2.2. Une séquence de contrôle différente de celles de transmission.

+ une séquence est envoyée par le terminal,
+ le convertisseur de code l'analyse et la traite de façon locale,
+ attente du caractère suivant.

5.5.3. La transmission d'une partie ou de la totalité de l'écran en respectant le mode sélectionné.

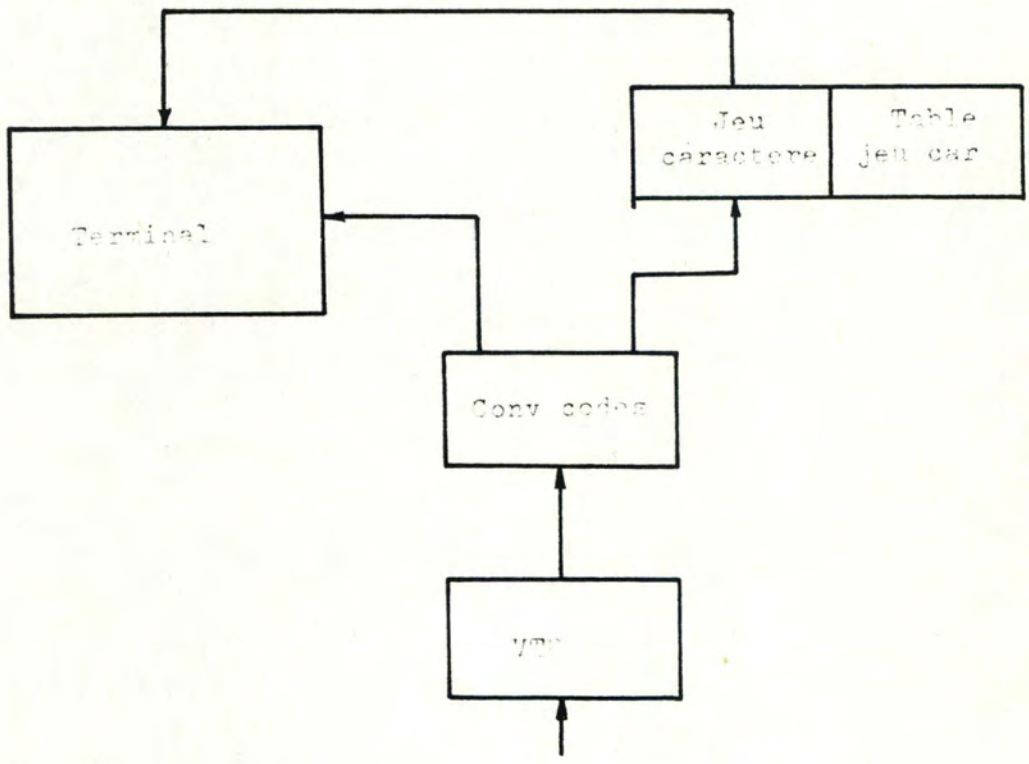


+ le convertisseur de code reçoit une séquence de contrôle demandant la transmission,
+ en fonction de la séquence, il retire de la mémoire écran les données qu'il faut envoyer

+ il met tous les caractères à envoyer dans la mémoire de la station VTP,
+ il lui signale qu'elle peut envoyer les données sur le réseau.

5.6. MODULES INDEPENDANTS DU TYPE D'APPLICATION.

5.6.1. Le module jeu de caractères.



Ce module ne travaille que sur la réception des caractères venant du réseau.

Les étapes sont :

+ la station VTP reçoit une page du réseau,
+ elle le signale au convertisseur de code,
+ celui-ci va distinguer trois types de caractères,

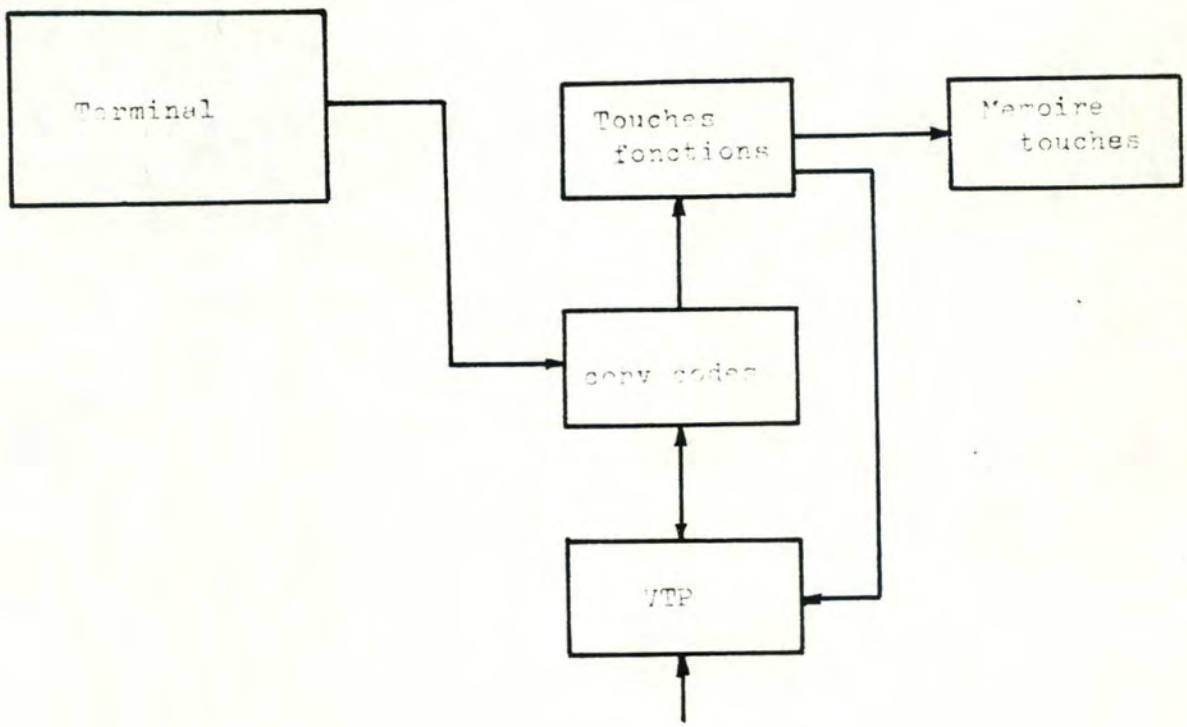
1) la séquence de contrôle qu 'il va envoyer au terminal après transformation du code.
2) un caractère appartenant à un langage émulé

- le convertisseur de code l'envoie au module "jeu caractère"
- celui-ci va le convertir en un caractère imprimable grâce à une table,
- il va l'envoyer au terminal.

3) un caractère appartenant à un langage non émulé

- le convertisseur de code l'envoie au terminal directement.

5.6.2. Le module des touches fonctions.



Deux cas sont à considérer :

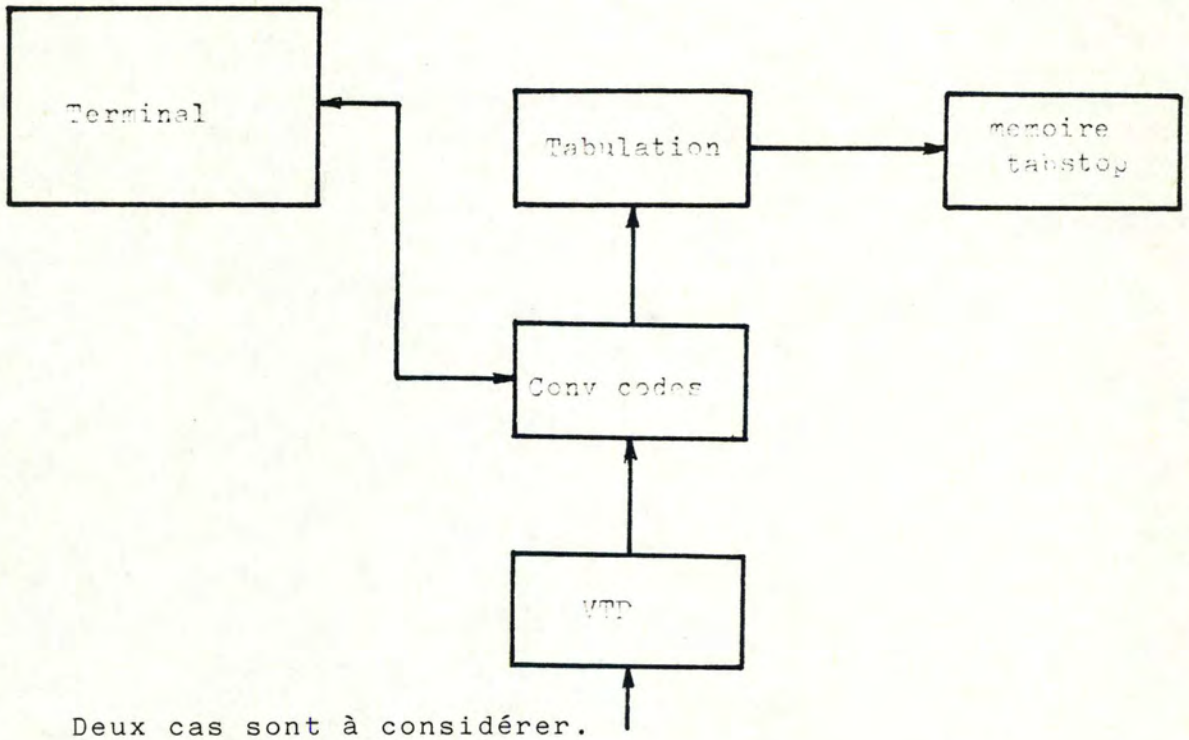
5.6.2.1. Lecture du programme d'une touche.

+ on enfonce une touche au terminal
+ le convertisseur de code reçoit la commande
+ il active le module touches fonctions
+ celui-ci va lire la séquence de contrôle où le texte dans la mémoire touche
+ il la dépose dans la mémoire de la station VTP
+ il rend la main au convertisseur de code
+ le convertisseur de code active la station VTP pour qu'elle envoie le texte

5.6.2.2. Ecriture du programme d'une touche.

+ la station VTP signale au convertisseur-code qu'elle a reçu une page du réseau
+ le convertisseur-code décode la séquence de contrôle de programmation des touches fonctions
+ il active le module touches fonctions
+ celui-ci enregistre le texte pour chaque touche dans la mémoire "touche" après validation
+ il rend la main au convertisseur de code après la détection de fin de séquence.

5.6.3. Le module tabulation.



Deux cas sont à considérer.

5.6.3.1. La réception d'une séquence de tabulation venant soit du réseau soit du terminal.

+ la station VTP (terminal) signale au convertisseur de code qu'il a des caractères à traiter
+ celui-ci décode la séquence de tabulation
+ il active le "module tabulation"
+ ce dernier va chercher en mémoire la position du "tabstop" demandé (position en avant ou en arrière)
+ il met à jour la position du curseur
+ il rend la main au convertisseur de code
+ celui-ci envoie la séquence de contrôle qui va positionner le curseur au bon endroit.

5.6.3.2. L'enregistrement des tabstops.

Elle peut se produire à deux moments.

1) lors de la négociation, la station VTP va mémoriser la position des "tabstops" qui seront utilisés par défaut si d'autres ne sont pas programmés au début du document.

2) pendant le travail sur l'application.

   Les séquences peuvent venir soit du réseau soit du terminal.

   + la station VTP (terminal) signale au convertisseur de code qu'il a des caractères à traiter
   + celui-ci décode la séquence d'enregistrement de tabstops
   + il active le module tabulation
   + ce dernier va enregistrer en mémoire la position du tabstop
   + il rend la main au convertisseur de code.

C H A P I T R E VI.

P R I N C I P E D E N E G O C I A T I O N.

CHAPITRE VI : PRINCIPE DE NEGOCIATION.

6.1 CHOIX D'UN SCHEMA DE NEGOCIATION.

Nous adaptons le schéma de négociation symétrique proposé par Schicker et Duenki qui est détaillé au chapitre III - 4.

6.2. JUSTIFICATION.

Notre étude se veut la plus générale possible or, seul le schéma symétrique n'est pas restrictif au niveau des partenaires en dialogue.

En effet, il ne limite pas le dialogue seulement entre une application et un terminal mais il permet de l'étendre aussi aux dialogues entre deux terminaux.

Le nombre de ces dialogues est non négligeable puisqu'il regroupe toutes les communications télex et télétex.

Ce schéma oblige à normaliser au niveau du réseau les critères de décision pour l'obtention du vecteur de négation réponse. Ceci prolonge l'idée d'un langage standard et d'une normalisation générale au niveau du réseau.

C H A P I T R E VII.

LA STATION V.T.P. ET LE MECANISME DES PRIMITIVES.

CHAPITRE VII : LA STATION VTP ET LE MECANISME DES PRIMITIVES.

7.1. REGLES GENERALES DE FONCTIONNEMENT DE LA STATION VTP.

7.1.1. Fonction et limitation de la station.

Dans le cadre de notre solution, la fonction de la station VTP reste celle déjà énoncée plus avant, c'est-à-dire gérer le dialogue des primitives.

Le mécanisme du dialogue est dépendant du fait que notre émulateur ne peut contenir que l'équivalent d'une page.Le créateur ne peut donc transmettre qu'une page à la fois et, autrement dit, ne peut donc envoyer la page suivante qu'après avoir reçu l'accusé réception positif de la page précédente.

La taille de la page est définie au cours de la phase de négociation.

Cela suscite au moins deux remarques :

- Chaque commande de marquage des documents (CDPB) doit obligatoirement provoquer un accusé réception, ce qui pourrait ne pas être le cas pour une taille de fenêtre supérieure à une page. (Une réponse pour acquittement de plusieurs pages à la fois).
- Les problèmes de saturation de mémoire ne se posent plus. En effet, lorsqu'on travaille avec une taille de fenêtre de 2, 3 ou 4, il peut y avoir un accusé négatif (RDPBN) suite à une saturation de mémoire, auquel cas il faut interrompre la transmission et effectuer une reprise, ou un accusé positif (RDPBP) qui doit contenir comme paramètre l'aptitude de l'utilisateur à encore recevoir des pages.

7.1.2. La gestion des changements de sens.

La gestion des changements de sens peut être gérée de diverses manières.

7.1.2.1. Système maître-esclave.

De cette manière, le créateur reste maître en ce sens que les demandes de changement de sens et donc l'envoi de la primitive adéquate ne peut venir que de lui.

L'utilisateur rend la main après chaque envoi de données c'est-à-dire après chaque page.

Exemple d'application.

Les zones formatées :

Le créateur envoie la page avec l'écran formaté puis la commande de changement de sens.
L'utilisateur remplit les zones et renvoie les données (1 page).

7.1.2.2. Système avec changement de maître.

De cette manière, le créateur passe la main à l'utilisateur qui devient maître.
L'envoi de n'importe quelle primitive excepté celles de négociation pourrait être autorisée au nouveau maître dans le respect du protocole (éventuellement un changement d'application, un arrêt de la liaison...)

Exemple d'application :

Texte libre.

Le scénario de transmission décrit ci-dessous permet de définir les commandes essentielles nécessaires au bon déroulement d'un transfert.

Le système de changement de sens est le système maître-esclave. Ce sont les seules commandes qui seront détaillées dans la suite de ce chapitre.

* a - <u>négociation</u>

CDCL →

← RDCLP

← RDCLN

* b - <u>dialogue</u>

CDS →

← RDSP

Page 1

CDUI →

Page acceptée

CDPB (1) →

← RDPBP (1)

Page 2

CDUI →

CDPB (2) →

Page refusée

← RDPBN

Page 2

CDUI →

CDPB (2) →

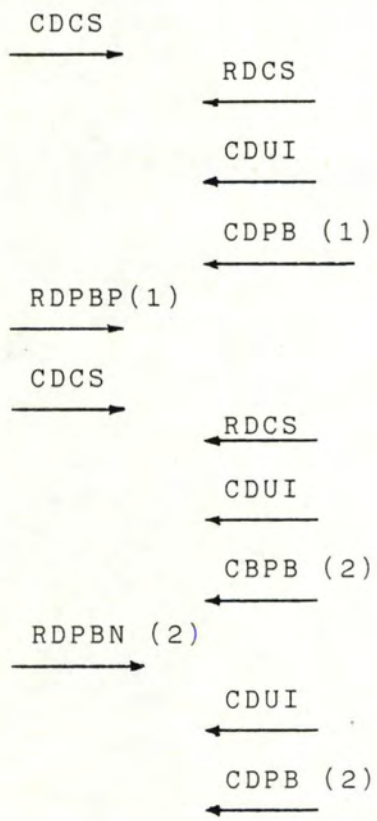
Page acceptée

← RDPBP (2)

Remarques :

On n'autorise qu' une et une seule retransmission. Cela suppose qu'il faut mémoriser le nombre de transmission d'une même page. Au cas où le créateur reçoit deux accusés négatifs pour une même page, il impose une interruption du transfert.

* Changement de sens du système maître-esclave.



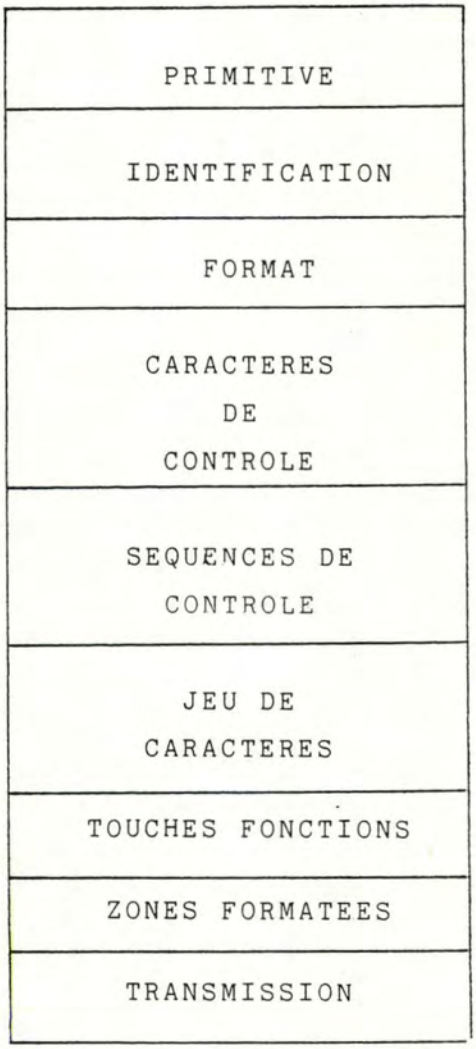
7.2. DESCRIPTION DES PRIMITIVES.

7.2.1. La primitive de négociation (CDCL).

7.2.1.1. Définition.

Le rôle de cette primitive a été défini au chapitre III. Nous rappelons simplement que préalablement à tout échange de données, cette primitive permet de se mettre d'accord sur les caractéristiques à retenir pour la communication.

7.2.1.2. Schéma.



7.2.1.3. Description complète de chaque zone.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-

7.2.1.3.1. Zone primitive.
****

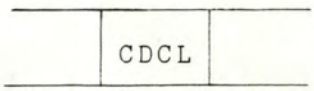
1 - Cette zone est composée d'une partie.

- Définition : c'est la primitive de négociation : CDCL (Command Document Capability List)..

- Caractéristiques : + code hexa : 80
+ codée sur un byte.

2 - Le format complet de la zone.

- Exemple :



- Valeurs limites des pointeurs.

7.2.1.3.2. Zone identification.
****

1 - Cette zone est composée de trois parties.

A) Partie I : le nom

- Définition : cette partie contient le nom du constructeur du terminal: soit celui qui a servi à faire l'application ou soit celui qui désire l'utiliser.

- Caractéristiques :

* un pointeur
* le nom + en ASCII (2/1 → 7/14)
* un tiret doit être ajouté entre les mots formant des noms composés

* si le nom est une abréviation il n'y aura pas de point entre les lettres.

  Ex. : IBM et pas I.B.M.

* le nom contiendra 40 caractères au maximum
* partie facultative.

B) Partie II : le numéro.

- Définition : cette partie contient le numéro du terminal dans la marque.

- Caractéristiques :

  * un pointeur
  * le numéro : + en ASCII (2/1 → 7/14)
    + en toute lettre
    + le numéro contiendra 40 caractères au maximum
    + partie facultative.

C) Partie III :

- Définition : Cette partie donne le type d'application traitée lors de ce dialogue, le type et le mode de l'appareil. La définition de ces différents termes a été donnée dans les chapitres précédents.

- Caractéristiques :

  * un pointeur
  * le type de l'appareil

TYPE	CODAGE
VIDEO	00
TELETYPE	01
IMPRIMANTE	10
X	11

* le mode de l'appareil

MODE	CODAGE
SCROLL	001
PAGE	010
DATA ENTRY	100

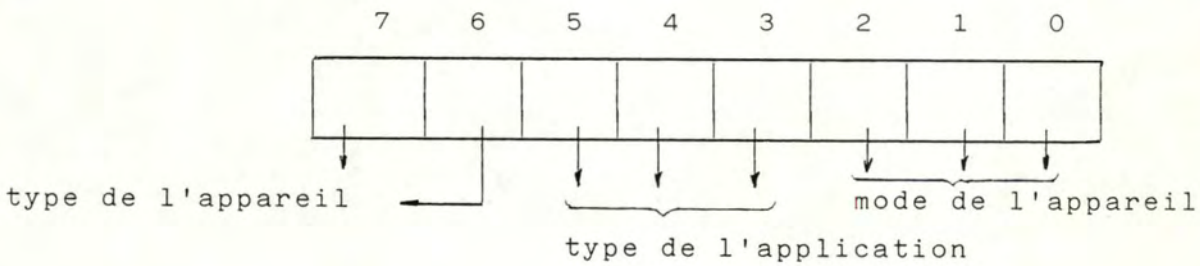
N.B. : ces codages sont les seuls permis, un autre entraînerait une erreur et le passage au mode SCROLL.

* le type d'application

TYPE	CODAGE
Traitement texte	001
Question-réponse	010
Zones formatées	011
Texte libre 1	100
Texte libre 2	101

N.B. : ces codages sont les seuls permis, un autre entraînerait une erreur.

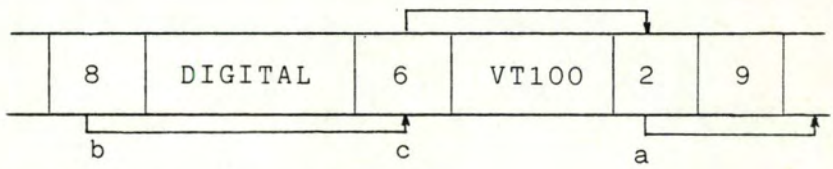
* ces trois derniers termes sont regroupés dans un byte comme suit :



ce byte est obligatoire.

2 - Le format complet de la zone.

A) Exemple



B) Valeurs limites des pointeurs.

a : pointeur = 2

b : $0 \leqslant$ pointeur $\leqslant 40$

c : $0 <$ pointeur $\leqslant 40$

7.2.1.3.3. Zone format.
****

1 - Cette zone est composée de neuf parties.

A) Partie 1 : le nombre de caractères par ligne.

- Définition :

Cette partie donne la liste des différents formats présents sur le terminal vidéo ou la liste des formats pris par défaut sur un télétype et une imprimante.
La liste du vecteur créateur renferme seulement le format nécessaire à l'application tandis que la liste du vecteur utilisateur renferme tous les formats disponibles.

- Caractéristiques :

* un pointeur
* la liste + codée sur un byte par format
+ en entier
+ en caractère par ligne
+ la liste ne peut pas être vide.

B) Partie 2 : la largeur du formulaire.

- Définition :

Cette partie donne la largeur du formulaire utilisé sur le terminal. Si celui-ci est évidemment un télétype ou une imprimante.

- Caractéristiques :

* un pointeur
* la largeur + codée sur un byte
  + en entier
  + donne la largeur du formulaire *10 afin de supprimer la partie fractionnaire.
  + exprimée en inch * 10

C) Partie 3 : l'interligne.

- Définition :

Cette partie donne la liste des interlignes possibles sur les imprimantes et les télétypes.

- Caractéristiques :

* un pointeur
* la liste + codée sur un byte par interligne
  + en entier
  + en lignes/inch
  + liste non vide

D) Partie 4 : le nombre de lignes par page.

Idem que 7.2.1.3.3. 1A sauf que "caractères par lignes" est remplacé par "lignes par page".

E) Partie 5 : la longueur du formulaire.

Idem que 7.2.1.3.3. 1B sauf que "largeur" est remplacé par "longueur".

F) Partie 6 : l'espacement entre caractères.

Idem que 7.2.1.3.3. 1C sauf que "lignes/inch" est remplacé par "caractères par inch *10".

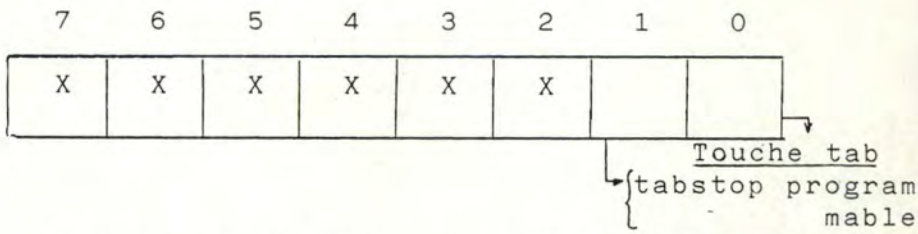
G) Partie 7 : la tabulation horizontale.

- Définition :

Cette partie indique si le terminal dispose et sait programmer des "tabulations stops". Puis elle donne la liste des Tabstops.

- Caractéristiques :

* un pointeur
* un mot d'état



bit 0 : valeur 1 : il existe une touche TAB sur le clavier du terminal.

0 : il n'en existe pas.

bit 1 : valeur 1 : les tabstops sont programmables.

0 : les tabstops ne sont pas programmables.

bits 2-7 : valeurs indéfinies.

Ce byte est obligatoire.

* la liste + codée sur un byte
+ chaque byte correspond à un tabstop horizontal
+ les tabstops sont en ordre croissant

+ tous les tabstops doivent être distincts
+ 0 ≤ un tabstop ≤ max_CAR ou MAXCART
+ en entier
+ zone facultative.

N.B. cette liste est celle des valeurs prises par défaut et sert aussi comme tabstops normaux si on n'en programme pas d'autres au début du document.

H) Partie 8 : la tabulation verticale.

Idem que ci-dessus sauf que

0 ≤ tabstop ≤ MAXLIG ou MAXLIGT

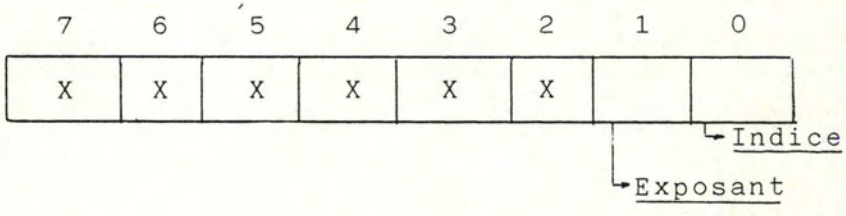
I) Partie 9 : les exposants et les indices.

- Définition :

Cette partie indique si le terminal peut reproduire un indice (A_2) ou un exposant (A^2).

- Caractéristiques :

* un pointeur
* un mot d'état



bit 0 : valeur 1 : Il y a moyen d'imprimer un indice
0 : il n'y a pas moyen.

bit 1 : valeur 1 : il y a moyen d'imprimer un exposant.

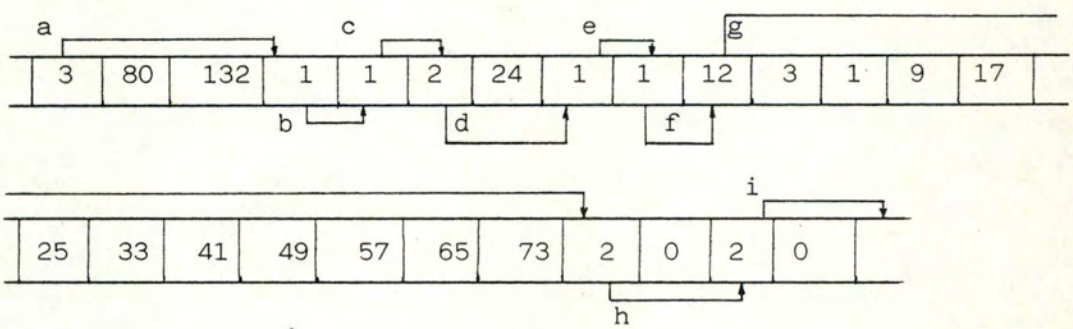
0 : il n'y a pas moyen.

bit 2-7 : valeurs indéfinies.

Cette partie est obligatoire.

2 - Le format complet de la zone.

A) Exemple.



B) Valeurs limites des pointeurs.

a : pointeur $\geq$ 2

b : pointeur $\geq$ 1

c : pointeur $\geq$ 1

d : pointeur $\geq$ 2

e : pointeur $\geq$ 1

f : pointeur $\geq$ 1

g : 2 $\leq$ pointeur $\leq$ MAXCAR

h : 2 $\leq$ pointeur $\leq$ MAXLIG

i : pointeur =2

7.2.1.3.4. Zone caractères de contrôle IA5
****

1 - Cette zone est composée d'une partie.

- Définition :

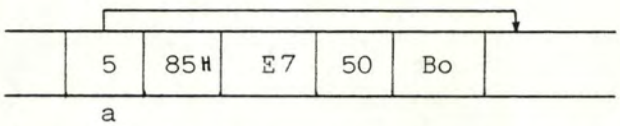
Certains terminaux utilisent les caractères de contrôle comme ceux-ci sont définis dans la norme, par contre d'autres les utilisent pour créer leurs propres séquences de contrôle.

- Caractéristiques :

* un pointeur
* la liste + 4 bytes
+ 1 byte par caractère de contrôle
+ bit : valeur 1 : si on suit la norme
0 : si on ne suit pas la norme.
+ la définition des caractères de contrôle se trouve dans l'annexe IV.

2 - Le format complet de la zone.

A) Exemple.



B) Valeurs limites des pointeurs.

a : pointeur = 5

7.2.1.3.5. Zone séquences de contrôle.
****

1 - Cette zone est composée d'une partie;

- Définition :

Cette partie indique pour chaque séquence de contrôle reprise dans l'annexe II la manière dont le terminal la traite.

- Caractéristiques :

  * un pointeur
  * une liste

    + 17 bytes
    + 1 byte reprend 4 séquences de contrôle

I	II	III	IV

    + les 2 bits de chaque séquence représente

      - 00 : la séquence n'existe pas sur le terminal et il n'est pas possible de la réaliser par la combinaison d'autres séquences.
        Exemple : "cursor up" sur un télétype

      - 01 : la séquence est admise par le terminal bien qu'il ne puisse pas la réaliser.
        Exemple : attribut de visualisation.

      - 10 : la séquence est réalisée par la combinaison d'autres fonctions présentes sur le terminal.
        Exemple : cursor next line = CR + Pn*LF

      - 11 : la séquence est réalisée directement par le terminal.

    + les différents bytes comprennent :

      - byte 1 : I cursor up
        II cursor down
        III cursor forward
        IV cursor Backward

      - byte 2 : I cursor next line
        II cursor previous line
        III index
        IV reverse index

- byte 3 : I Relative column movement
  II relative line positioning
  III cursor addressing
  IV absolute column movement

- byte 4 : I absolute line positioning
  II request cursor position report
  III answer cursor position report
  IV

- byte 5 : I taille des caractères D D
  II DW DH^{+}
  III DW DH^{-}
  IV SW SH

- byte 6 : I taille des caractères DW SH
  II SW DH
  III SW DH^{+}
  IV SW DH^{-}

- byte 7 : I attributs des caractères: normal
  II + gros
  III - gros
  IV invisible

- byte 8 : I attributs des caract.: souligné
  II clignotant
  III inverse
  IV italique

- byte 9 : I set horizontal tabstop
  II set vertical tabstop
  III clear horizontal tabstop (0)*
  IV clear horizontal tabstop(3)*

- byte 10 : I clear horizontal tabstop (0)*
  II clear vertical tabstop (1)*
  III clear vertical tabstop (4)*
  IV horizontal tab

- byte 11 : I horizontal backtab
  II vertical tab
  III X
  IV X

- byte 12 : I marge gauche-droite
  II marge haut - bas
  III indice
  IV exposant

- byte 13 : I effacement d'une ligne (0)*
  II (1)*
  III (2)*
  IV effacement d'une page (0)*

- byte 14 : I effacement d'une page (1)*
  II (2)*
  III effacement d'une zone (0)*
  IV (1)*

- byte 15 : I effacement d'une zone (2)*
  II effacement d'un champ (0)*
  III (1)*
  IV (2)*

- byte 16 : I autowrap mode
  II LF/NL mode
  III erasure mode
  IV form build mode

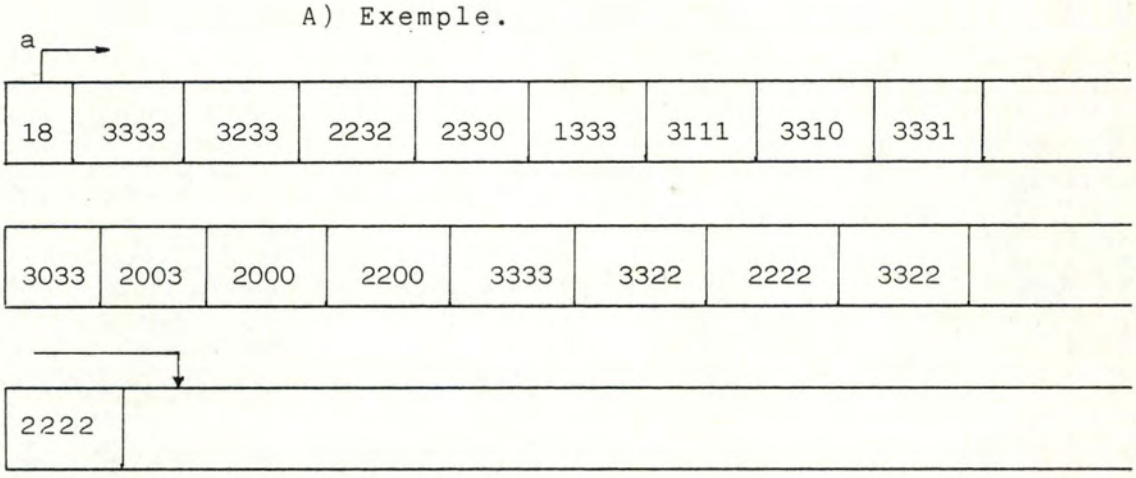
- byte 17 : I protect form mode
  II autotab mode
  III block mode
  IV X

+ tous les bytes sont obligatoires.

* voir en annexe II la signification du chiffre entre parenthèses.

2 - le format complet de la zone.

A) Exemple.



B) Valeurs limites des pointeurs

a : pointeurs = 18

7.2.1.3.6. Zone jeu de caractères.
****

1- Cette zone est composée d'une partie.

- Définition :

Cette liste reprend tous les jeux de caractères repris dans l'annexe II et elle indique ceux que le terminal possède.

- Caractéristiques :

* un pointeur
* la liste + 2 bytes
+ chaque bit représente un alphabet
+ bit : valeur 1 : le terminal dispose de cet alphabet
0 : le terminal n'en dispose pas.

+ byte 1

7	6	5	4	3	2	1	0
A	B	E	F	G	I	N	P

bit 0 : portugais
1 : norvégien
2 : italien
3 : allemand
4 : français
5 : espagnol
6 : américain
7 : anglais

+ byte 2

S	X	X	X	X	X	X	X

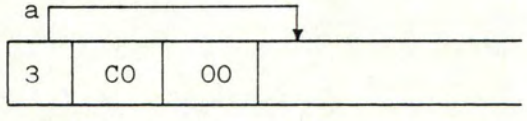
bit 7 suédois
Bits 0-6 valeur indéfinie

+ ces deux bytes sont obligatoires.

+ les bits restants peuvent être utilisés pour augmenter le nombre d'alphabets exploitables.

2 - Le format complet de la zone.

A) Exemple.



B) Valeurs limites des pointeurs.

a : pointeur = 3

7.2.1.3.7. Zone touches fonctions.
****

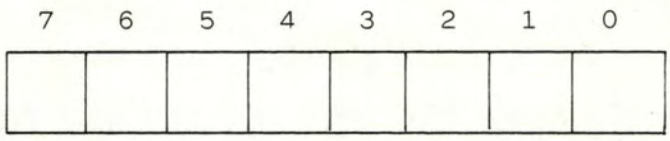
1 - Cette zone est composée d'une partie.

- Définition :

Cette partie indique si le terminal dispose de touches fonctions sur le clavier et si on peut les programmer.

- Caractéristiques :

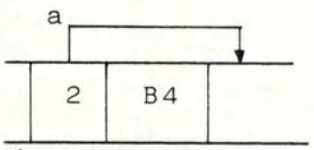
  * un pointeur
  * un mot d'état



- bit 7 : valeur 1 : le terminal dispose de touches fonctions
  0 : le terminal n'en dispose pas
- bit 6 : valeur 1 : les touches sont programmables
  0 : les touches ne sont pas programmables
- bits 0-5 : indique le nombre de touches présentes sur le clavier.
  + en entier
  + $1 \leqslant$ touches $\leqslant 64$

2 - Le format complet de la zone.

A) Exemple.



B) Valeurs limites des pointeurs.

a : pointeur = 2

7.2.1.3.8. Zone zones formatées.
****

1 - Cette zone est composée de deux parties.

A) <u>Partie 1</u>

- Définition :

Cette partie indique si le terminal peut protéger des zones sur l'écran et si il peut utiliser des attributs.

- Caractéristiques :

  * un pointeur
  * un mot d'état

7	6	5	4	3	2	1	0
x	x	x	x	x	x		

bit 0 : valeur 1 : le terminal dispose de la faculté de protéger des zones sur l'écran

0 : le terminal n'en dispose pas

bit 1 : valeur 1 : le terminal permet d'utiliser des attributs différents de la simple protection.

0 : le terminal ne le permet pas

bit 2-7 : valeurs indéfinies.

Ce byte est obligatoire.

2 - Partie 2 : attributs

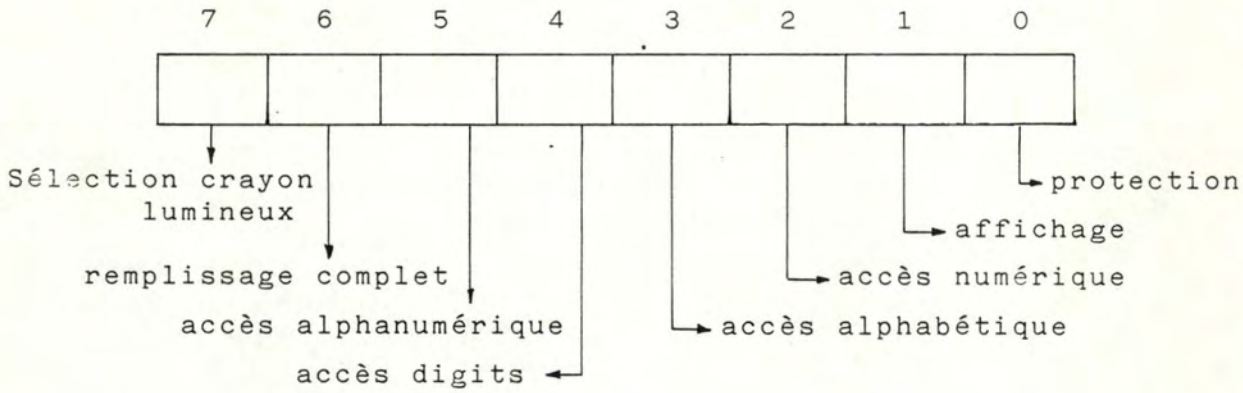
- Définition :

Cette partie indique les attributs que le terminal peut assigner à une zone.

- Caractéristiques :

  * un mot d'état

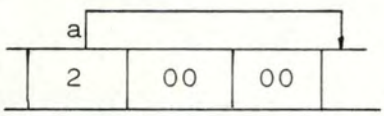
    + chaque bit correspond à un attribut
    + bit : valeur 1 : dispose de l'attribut
      0 : ne dispose pas de l'attribut.



+ ce byte est obligatoire

2 - Le format complet de la zone.

A) Exemple :



B) Valeurs limites des pointeurs.

a : pointeur = 3

7.2.1.3.9. Zone transmission.
****

1 - Cette zone est composée d'une partie.

- Définition :

Cette partie indique pour chaque séquence de contrôle de transmission reprise dans l'annexe II la manière dont le terminal la traite.

- Caractéristiques :

* un pointeur
* une liste

+ 3 bytes
+ 1 byte reprend 4 séquences de contrôle

I	II	III	IV

+ les deux bits dans chaque séquence ont la même signification qu'au paragraphe 1.3.5.1.

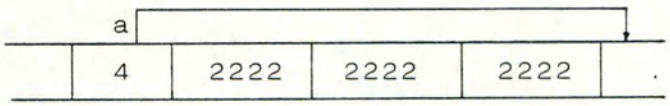
+ les différents comprennent :

- byte 1 : I immediate transfer
  II defer transfer
  III set transmit state sequence
  IV transmit sequence
- byte 2 ; I transfer partiel page
  II transfer entire page
  III transfer display
  IV transfer line
- byte 3 : I transfer one field
  II transfer all field
  III transfer only inguarded areas
  IV transfer all areas

+ ces trois bytes sont obligatoires.

2 - Le format complet de la zone

A) Exemple.



B) Valeurs limites des pointeurs.

a : pointeur = 4

7.2.1.4. Les règles de traitements.

Nous allons étudier comment à partir de deux vecteurs de négociation, l'un venant du créateur et l'autre venant de l'utilisateur, nous obtenons un troisième vecteur appelé vecteur résultat. Celui-ci constitue la base de la classe du terminal virtuel utilisée pour le dialogue entre l'application et le terminal ou entre les deux terminaux.

Nous allons énoncer un ensemble de règles précisées qui permettent de créer sans aucune ambiguïté le vecteur résultat.

7.2.1.4.1. Le traitement général.

La succession des opérations va être la suivante.

Ou bien un terminal désire entrer en communication avec une application ou un autre terminal.

Actions + envoyer la primitive CDCL avec comme paramètres le vecteur de négociation créateur si on désire dialoguer avec un autre terminal, ou le vecteur utilisateur si on désire dialoguer avec une application.

+ passer dans un état attente.

Ou bien un terminal ou une application reçoit un CDCL

Actions + envoyer la primitive CDCL avec comme paramètres le vecteur de négociation créateur si on est une application, ou le vecteur utilisateur si on est le terminal.

+ analyser les paramètres du vecteur reçu dans CDCL pour créer le vecteur résultat (voir1.4.2

+ s'il existe une erreur, on envoie une primitive de réponse négative RDCL N avec la cause de l'erreur (incompatibilités des terminaux ou une erreur de paramètres).

+ s'il n'existe pas d'erreur, on envoie une primitive de réponse positive RDCL P avec le vecteur de résultat.

+ passer dans un état d'attente ou libérer la communication.

7.2.1.4.2. Le traitement de chaque zone.
****

1 - Validation des pointeurs.

Au début de chaque partie, on valide les pointeurs. Ceci se fait sur base des valeurs limites trouvées dans le paragraphe intitulé "valeurs limites des pointeurs"(voir description de chaque zone.)

2 - Zone primitive.

3 - Zone identification.

+ Analyse du nom et du numéro

Nous pouvons aborder le traitement de cette zone de deux manières différentes:

*La première consiste à traiter tous les cas de négociation comme des cas généraux. Si les noms et les numéros des terminaux sont identiques, on les considère comme différents et on utilise, dans l'émulateur, le convertisseur de code. Celui-ci ne servira à rien.
En effet, le convertisseur de l'émetteur va convertir le code A en un code standard B puis l' envoyer sur le réseau via la station VTP.

A son arrivée dans l'émulateur du récepteur, on effectue la conversion inverse (B→A) pour retrouver le code A. Les deux conversions sont inutiles et elles font perdre du temps.

*Le deuxième consiste à traiter les cas où les deux terminaux sont identiques comme des cas particuliers. On n'utilise pas l'émulateur. On évite toutes les conversions inutiles.

- Règles.

Si on suit le premier principe, on ne se soucie pas du nom et du numéro du terminal et on passe à l'analyse de la zone suivante.
Par contre dans le deuxième principe, on va tester l'égalité des deux noms puis des deux numéros. Si celle-ci se vérifie, on arrête la négociation et on envoie une réponse positive (RDCLP) avec comme paramètre le vecteur de négociation du créateur. Sinon on revient au cas général.

En même temps qu'on teste l'égalité du nom et du numéro, on doit vérifier s'ils sont conformes aux spécifications.

- pour le nom : a) vérifier qu'il n'y a pas de blanc (2/0) dans la partie.
  b) vérifier qu'il n'y a pas de point (2/14) dans la partie.
- pour le numéro : vérifier qu'il n'y a pas de blanc (2/0) dans la partie.

+ Analyse des types d'application.

Application traitement de texte.

Si l'application est du type traitement de texte, on peut considérer le terminal utilisateur comme une simple imprimante.

La nature du terminal utilisateur importe peu et elle est toujours suffisante.
Le vecteur résultat indique le type d'application égal au type d'application du vecteur créateur.
Le mode du vecteur résultat est le mode scroll.
Le type d'appareil du vecteur résultat est le type du vecteur créateur.

Pour les terminaux qui peuvent travailler dans plusieurs modes, il faut réaliser le passage en mode scroll en envoyant au terminal la séquence de contrôle adéquate.

Récapitulatif	
Vecteur résultat :	mode: scroll
	type d'appareil: type du créateur
	type d'application: type du créateur
Séquence de contrôle :	passage en mode SCROLL.

Tableau récapitulatif analyse

Application zones formatées.

Si l'application est du type zones formatées il faut que le terminal créateur soit du mode page ou data entry. Il faut, en plus, que le terminal utilisateur soit différent d'une imprimante, puisqu'elle ne permet pas le remplissage des zones, et d'un télétype, puisqu'il ne permet pas l'effacement de données et le positionnement du curseur en abscisse et en ordonnée.

Le mode du terminal pour le vecteur résultat est trouvé dans le tableau suivant :

Mode et type du terminal utilisateur	Mode du terminal créateur.	
	Page	Data entry
Imprimante ligne		
Télétype ligne		
Vidéo ligne	Page*	data entry*
Vidéo page	Page	data entry*
vidéo data entry	Page	data entry

Tableau 1 : Le mode du terminal dans le vecteur résultat pour une application zones formatées.

Le type d'application du vecteur résultat est celui du vecteur créateur. Le type d'appareil du vecteur résultat est de type vidéo.
Après remplissage du vecteur résultat, il faut mettre à jour le mot d'état (STATUS-EMUL) de l'émulateur de l'utilisateur pour indiquer que ses modules "zones formatées" et "pages" doivent être utilisés.
Il faut mettre le terminal en mode "bloc" par l'émission de la séquence de contrôle adéquate.

Si une incompatibilité existe, on envoie la primitive de réponse négative (RDCLN) avec la cause du refus de dialogue et on termine la négociation.

Récapitulatif
Vecteur résultat : mode le l'appareil : voir tableau 1 type de l'appareil : VIDEO type d'application : type du créateur Séquence de contrôle : passage en mode "bloc" Module émulateur utilisé : zones formatées.

Application questions-réponses.

Si l'application est du type questions-réponses, il faut que le terminal utilisateur soit de type VIDEO.

Le mode du terminal pour le vecteur résultat est trouvé dans le tableau suivant :

		Mode du terminal créateur.		
		Scroll	Page	Data entry
Mode du terminal utilisateur	Scroll	Scroll	Page*	Data entry*
	Page	Scroll	Page	Data entry*
	Data Entry	Scroll	Page	Data entry

Tableau 2 : Le mode du terminal dans le vecteur résultat pour une application questions-réponses.

Le type de l'application du vecteur résultat est celui du vecteur créateur. Le type d'appareil du vecteur résultat est de type vidéo.

---

*signifie que la solution est obtenue par émulation.

Après remplissage du vecteur, il faut mettre à jour le mot d'état (STATUS-EMUL) de l'émulateur de l'utilisateur pour indiquer que les modules "page" et "zone formatées" doivent être utilisés. Il faut commuter le terminal dans un mode en accord avec le mode du vecteur résultat ("conv" pour scroll et "bloc" pour page et data entry) en envoyant la séquence de contrôle adéquate.

Récapitulatif
Vecteur résultat : mode de l'appareil : voir tableau 2 type de l'appareil : VIDEO type d'application : Type du créateur Séquence de contrôle : passage en mode "conv" ou en mode "bloc" Modules émulateurs utilisés : - zones formatées - page

Si une incompatibilité existe, on envoie la primitive de réponse négative (RDCLN) avec la cause du refus de dialogue et on termine la négociation.

Application texte libre 1.

Le mode du terminal pour le vecteur résultat se trouve dans le tableau suivant.

Tableau 3

Mode et type du terminal utilisateur	Mode et type du terminal créateur			
	Télétype Scroll	Vidéo Scroll	Vidéo Page	Vidéo data entry
Imprimante scroll	scroll	scroll		
Télétype scroll	scroll	scroll		
Vidéo scroll	scroll	scroll	Page*	data entry
Vidéo page	scroll	scroll	page	data entry*
Vidéo data entry	scroll	scroll	page	data entry

Le type de l'application du vecteur résultat est celui du vecteur créateur. Le type d'appareil du vecteur résultat est celui du vecteur créateur.

Après remplissage du vecteur, il faut mettre à jour le mot d'état (STATUS-EMUL) de l'émulateur de l'utilisateur pour indiquer que les modules "page" et "zones formatées" doivent être utilisés.
Il faut commuter le terminal dans un mode en accord avec le mode du vecteur résultat ("conv" pour scroll et "bloc" pour page et data entry) en envoyant la séquence de contrôle adéquate.

Si une incompatibilité existe, on envoie la primitive de réponse négative (RDCLN) avec la cause de refus de dialogue et on termine la négociation.

Récapitulatif
Vecteur résultat : mode de l'appareil : voir tableau 3 type de l'appareil : type du créateur type d'application : type du créateur Séquence de contrôle : passage en mode "conv" ou en mode "bloc" Modules émulateurs utilisés : - zones formatées - page

Application texte libre 2.

Le mode et le type du terminal pour le vecteur résultat se trouvent dans le tableau suivant :

Mode et type du terminal utilisateur	Mode et type du terminal créateur			
	TTY scroll	Vidéo scroll	Vidéo page	Vidéo data entry
Imprimante scroll	Imprimante scroll	Imprimante scroll	X	X
Télétype scroll	Télétype scroll	Télétype scroll	X	X
Vidéo scroll	Télétype scroll	Vidéo scroll	Vidéo Page*	Vidéo data entry*
Vidéo page	X	Vidéo page*	Vidéo page	Vidéo data entry*
Vidéo data entry	X	Vidéo data entry*	Vidéo data entry*	Vidéo data entry

Tableau 4

Le type de l'application du vecteur résultat est celui du vecteur créateur.
Après remplissage du vecteur, il faut mettre à jour le mot d'état (STATUS-EMUL) de l'émulateur de l'utilisateur ou du créateur, pour indiquer que les modules "page" ou "zones formatées" doivent être utilisés. Il faut commuter les deux terminaux dans un mode en accord avec le mode de vecteur résultat ("conv" pour scroll et "bloc" pour page et data entry) en envoyant la séquence adéquate.

Si une incompatibilité existe, on envoie la primitive de réponse négative (RDCLN) avec la cause du refus de dialogue et on termine la négociation.

Récapitulatif.
Vecteur résultat : mode de l'appareil : voir tableau 4 type de l'appareil : voir tableau 4 type d'application : type du créateur. Séquence de contrôle : passage en mode "conv" ou en mode "bloc". Modules émulateurs utilisés : - zones formatées - page.

Autres cas.

Si le type d'application n'est pas un des cinq traités, on arrête la négociation après l'émission d'une primitive de réponse négative (RDCLN) accompagnée de la cause du refus de dialogue.

4 - Zone format.

+ Analyse de la taille du formulaire (partie 1,2,3, 4,5,et 6).

On prend comme point de départ le format en longueur (nombre de lignes par page) et en largeur (nombre de caractères par ligne) du terminal créateur de l'application.

On tente, dans la liste des formats du terminal utilisateur, d'en trouver un qui puisse contenir le plus justement possible celui du terminal créateur.

On remplit le vecteur résultat avec le format du terminal créateur.

S'il n'y a pas de format dans la liste de l'utilisateur qui puisse convenir, deux cas se présentent :

* les imprimantes et les télétypes :
  on peut encore changer les formats en utilisant la liste des caractères par inch et la largeur du formulaire, ou la liste des interlignes et la longueur du formulaire.

  Si après calcul, on trouve un format utilisable, on envoie les séquences de contrôle nécessaires pour commuter le format.

* les vidéos :

  on ne peut rien faire de plus.

Si on n'arrive pas à trouver un format, on peut adopter une des trois techniques suivantes :

* <u>Module gestion d'écran.</u>

  Le fonctionnement de ce module est détaillé au chapitre IV.
  Cette technique est utilisée pour les applications de type "questions-réponses", "zones formatées" et "texte-libre 1 et 2" sur terminal vidéo.

Lors de la négociation, chez l'utilisateur et si c'est nécessaire, on met à jour le mot d'état (STATUS_EMUL) pour indiquer que le module gestion écran doit être utilisé.
On mémorise la longueur (lignes/page) et la largeur (caractères/ligne) de l'écran créateur et on commute le terminal sur son format maximum en envoyant les séquences de contrôle adéquates.

* Module reformateur

Le fonctionnement de ce module est détaillé au chapitre IV.
Cette technique est utilisée pour les applications de type"traitement de texte". Dans l'émulateur de l'utilisateur et si c'est nécessaire, on met à jour le mot d'état (STATUS-EMUL) pour indiquer que le module reformateur doit être utilisé.
On mémorise la longueur (lignes/page) et la largeur (caractères/ligne) de l'écran du créateur et on commute le terminal sur son format maximum en envoyant les séquences de contrôle adéquates.

* Module pliage.

Le fonctionnement de ce module est détaillé au chapitre IV.
Cette technique est utilisée pour les applications de type"texte libre 1 et 2"sur terminal, imprimante ou télétype.
Dans l' émulateur de l'utilisateur et si c'est nécessaire, on met à jour le mot d'état (STATUS-EMUL) pour indiquer que le module pliage doit être utilisé. On mémorise la longueur et la largeur de l'écran du créateur et on commute le terminal sur son format maximum en envoyant les séquences de contrôle adéquates.

Récapitulatif :

- Si un format de l'utilisateur le format du créateur commute sur format + module pliage + mémoriser longueur et largeur
- Si un format de l'utilisateur le format du créateur commuter le terminal de l'utilisateur sur ce format
- Si aucun format de l'utilisateur au format du créateur voir tableau.

	Type d'application du créateur.			
Type du terminal utilisateur	trait. texte	Questions-Réponses	Zones-formatées	Texte libre 1 et 2
Imprimante	Module reformateur Mémoriser l et L Commuter l			Module pliage Mémoriser l et L Commuter l et L
Télétype	Module reformateur Mémoriser l et L Commuter l et L			Module pliage Mémoriser l et L Commuter l et L
Vidéo	Module reformateur Mémoriser l et L Commuter l et L	Module écran mémoriser l et L Commuter l et L	Module écran Mémoriser l et L Commuter l et L	Module écran Mémoriser l et L Commuter l et L

l : nombre de caractères par ligne

L : nombre de lignes par page.

+ Analyse des tabulations horizontales et verticales.

* La première chose à faire est de vérifier l'existence de tabulations sur les terminaux mis en présence.

  Le vecteur résultat dépend du type d'application. Pour les types de négociation application-terminal, on obtient le vecteur résultat suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	0
1	0	1 b
1	1	1

Résultat = Créateur

b : signifie qu'il y a émulation de la fonction dans le terminal utilisateur.

Il faut mettre à jour le mot d'état (STATUS-EMUL) dans l'émulateur de l'utilisateur pour indiquer que les modules"tabulation horizontale et/ou verticale" doivent être utilisés.

Pour le type de négociation terminal-terminal, on obtient le vecteur résultat suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	1 A
1	0	1 B
1	1	1

Résultat = créateur OU utilisateur

A,B : signifie qu'il y a émulation de la fonction dans le terminal utilisateur pour B, et créateur pour A.

Dans ce cas, l'échange peut se faire dans les deux sens.
Il faut mettre à jour le mot d'état (STATUS-EMUL), dans l'émulateur du créateur ou de l'utilisateur si nécessaire, pour indiquer que les modules "tabulation horizontale et/ou verticale" doivent être utilisés.

* Si la fonction de tabulation existe, on teste si elle est programmable.

  On obtient le vecteur résultat de la même manière que précédemment.

  Il faut mettre à jour le mot d'état (STATUS-EMUL), dans l'émulateur du créateur ou de l'utilisateur, pour indiquer que les modules tabulations doivent être utilisés si le terminal dispose de tabulations mais qu'il ne sait pas les programmer.

* Soit on mémorise dans l'émulateur **la liste** des "tabstops" se trouvant dans le vecteur créateur (ou utilisateur pour une application texte libre 2).

  Soit on envoie au terminal la séquence de contrôle adéquate pour programmer les tabstops à partir de la liste du vecteur créateur.

  On mémorise aussi cette liste dans le tableau (TAB-DEF) de l'émulateur où elle sera utilisée comme valeur prise par défaut.

Il faut aussi vérifier que, pour toutes paires de tabstops consécutives dans la liste reçue (soit A et B) :

Si A est avant B dans la liste, A doit être strictement plus petit que B.

Si une erreur existe, on envoie une primitive de réponse négative (RDCLN) avec une cause égale a une erreur de paramètre et on arrête la communication.

* Si la fonction de tabulation n'existe pas, on passe à l'analyse de la partie suivante.
  Si la tabulation ne doit pas être programmée on passe la liste des tabstops.

* Le vecteur résultat ne contient pas la liste des tabulations stops verticales et horizontales.

+ Analyse des indices et exposants.

* Le vecteur résultat est obtenu en suivant le tableau :

Créateur	Utilisateur	résultat
0	0	0
0	1	1^{A}
1	0	1^{B}
1	1	1

A,B : signifie qu'il y a émulation de la fonction dans le terminal utilisateur pour B, et créateur pour A

Résultat = Créateur OU Utilisateur

5 - Zone "caractères de contrôle IA5"

Le vecteur résultat est un ET logique entre les deux vecteurs utilisateur et créateur.

6 - Zone "séquences de contrôle et transmission"

Il faut traiter chaque séquence de contrôle séparément.
Dans le vecteur résultat et pour chaque séquence, on retrouve les quatre cas suivants :

00 : la séquence ne peut pas être utilisée par les 2 terminaux sinon un message d' erreur apparaît à l'émetteur de la séquence.

01 : si la séquence est utilisée par l'un des deux terminaux il ne se passera rien sur l'autre.

10 : la séquence existe sur l'un des deux terminaux et elle est émulée sur l'autre à partir de séquences plus simples.

11 : la séquence existe sur les deux terminaux sans émulation.

	Créateur			
Utilisateur	00	01	10	11
00	00	00	00	00
01	00	00	00	01
10	00	00	00	10
11	00	01	10	11

7 - Zone "jeux de caractères".

Le vecteur résultat est obtenu en faisant un OU logique entre les deux vecteurs origines (créateur et utilisateur).

Si un alphabet est retenu mais pas présent sur le terminal, on met à jour le mot d'état de l'émulateur pour indiquer que le module "jeux caractères" doit être utilisé.

On mémorise dans l'émulateur une variable (LANG) contenant les alphabets à transformer s'ils sont utilisés car ils n'existent pas sur le terminal.

LANG = RESULTAT XOR CREATEUR UTILISATEUR

8 -Zone "touches fonctions".

* On vérifie que les deux terminaux possèdent des touches de fonctions.

Si celui de l'utilisateur n'a pas ces touches sur le clavier, il est impossible de les émuler par du software.

On obtient le vecteur résultat suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	0
1	0	0
1	1	1

Résultat = Créateur ET Utilisateur

Si l'application a besoin de ces touches et que l'utilisateur n'en a pas, il faut envoyer une primitive de réponse négative (RDCLN) et terminer la communication.

* Si les deux terminaux possèdent ces touches, il faut savoir si elles sont programmables.

On obtient le vecteur résultat suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	1^{A}
1	0	1^{B}
1	1	1

A,B : signifie qu'il y a émulation de la fonction dans le terminal utilisateur pour B, et créateur pour A.

Résultat = Créateur OU Utilisateur

On met à jour le mot d'état (STATUS-EMUL) de l'émulateur pour indiquer que le module "touches fonctions" doit être utilisé si nécessaire.

Si un des terminaux ne peut pas les programmer on passe à l'étape suivante de l'analyse du vecteur après lecture des autres parties.

* Il faut savoir le nombre de touches dont on dispose. Deux cas se présentent suivant le type de négociation :

-Application-terminal :

Il faut absolument le nombre de touches du terminal créateur. Si le terminal utilisateur dispose d'assez de touches il n'y a pas de problème.
On place dans le vecteur résultat le nombre de touches du créateur.
Par contre, s'il n'y en a pas assez, on envoie une primitive de réponse négative (RDCLN) et on termine la communication.

-Terminal-terminal :

On place dans le vecteur résultat le nombre minimum de touches entre les deux terminaux.

9- Zones formatées.

* On commence par vérifier que nous avons un mode d'appareil "data entry" ou "page" dans le vecteur résultat.
Si ce n'est pas le cas, on saute la zone et on analyse la zone de transmission.

Par contre, dans l'autre cas, on vérifie si les deux terminaux peuvent protéger des zones à l'écran.

On obtient le vecteur résultat suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	1^{A}
1	0	1^{B}
1	1	1

A,B : signifie qu'il y a émulation de la fonction dans le terminal utilisateur B, et créateur A.

* Si la possibilité de protéger des zones existe, on va vérifier si on peut utiliser des attributs autres que la simple protection de zone.

On obtient le vecteur résultat suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	1
1	0	1
1	1	1

Résultat = Créateur OU Utilisateur

* On relève les types d'attributs utilisables ou émulables.

On divise en deux classes les attributs.

-Attributs 0,1,2,3,4,5,et 6 qui sont émulables par un ajout de logiciel dans l'émulateur.

On obtient le vecteur résultat en appliquant à chaque attribut le tableau suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	1^{A}
1	0	1^{B}
1	1	1

Résultat = Créateur OU Utilisateur

A,B : signifie qu'il y a émulation de la fonction dans le terminal utilisateur pour B, et créateur pour A.

-Attribut 7 qui n'est pas émulable par un ajout de logiciel.

On obtient le vecteur résultat en appliquant à cet attribut le tableau suivant :

Créateur	Utilisateur	Résultat
0	0	0
0	1	0
1	0	0
1	1	1

Résultat = Créateur ET Utilisateur

Si l'application nécessite l'attribut 7 et s 'il n'est pas présent sur le terminal utilisateur, on envoie la primitive de réponse négative (RDCLN) et on termine la négociation.

Tous les paramètres de cette zone seront utilisés dans le module " zones formatées " de l'émulateur.

7.2.1.5. Les algorithmes.

Voir annexe III.

7.2.2. La primitive de réponse positive à la négociation : RDCLP.

7.2.2.1. Définition.

Si on est arrivé à la fin de l'analyse du vecteur de négociation et qu'aucune erreur de paramètres ou aucune incompatibilité ne s'est produite nous envoyons la primitive de réponse positive (RDCLP). Comme nous avons adopté un schéma de négociation symétrique, nous renvoyons la liste des paramètres sortis de la négociation.

7.2.2.2. Schéma.

PRIMITIVE
IDENTIFICATION
FORMAT
CARACTERES DE CONTROLE
SEQUENCES DE CONTROLE
JEU DE CARACTERES
TOUCHES FONCTIONS
ZONES FORMATEES
TRANSMISSION

7.2.2.3. Description de chaque zone.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.

Cette primitive reprend les paramètres de la primitive de négociation (CDCL) avec plusieurs simplifications notamment la suppression des pointeurs au début de chaque partie.

7.2.2.3.1. Zone primitive.
****

1) Cette zone est composée d'une partie.

- Définition : c'est la primitive de réponse positive à la négociation : RDCLP (Response Document Capability List Positive).

- Caractéristiques :

  * code hexa : 81
  * codée sur un byte.

2) Le format complet de la zone.

- Exemple :

	RDCLP	

- Valeurs limites des pointeurs : néant.

7.2.2.3.2. Zone identification.
****

1) Cette zone est composée d'une partie.

- Définition : cette partie est constituée d'un byte qui retient le type et le mode de l'appareil, et le type d'application né de la négociation.

- Caractéristiques :

  * voir la primitive CDCL § 7.2.1.3.2.1.(c)

2) Le format complet de la zone.

- Exemple :

- Valeurs limites des pointeurs : néant.

7.2.2.3.3.Zone format.
****

1) Cette zone est composée de cinq parties.

a- Partie 1.

- Définition : Cette partie est constituée d'un byte qui indique le nombre de caractères par ligne retenu lors de la négociation.

- Caractéristiques :

* voir la primitive CDCL § 7.2.1.3.3.1.(A)

b- Partie 2.

- Définition : Cette partie est constituée d'un byte qui indique le nombre de lignes par page retenu lors de la négociation.

- Caractéristiques :

* voir la primitive CDCL § 7.2.1.3.3.1.(D)

c- Partie 3.

- Définition : cette partie est constituée d'un byte qui représente le mot d'état des tabulations horizontales retenues après la négociation.

- Caractéristiques :

* voir la primitive CDCL § 7.2.1.3.3.1.(G)

d- Partie 4.

- Définition : cette partie est constituée d'un byte qui représente le mot d'état des tabulations verticales retenues après la négociation.

- Caractéristiques :

  * voir la primitive CDCL § 7.2.1.3.3.1.(H)

e- Partie 5.

- Définition : cette partie est constituée d'un byte qui représente le mot d'état des indices et exposants retenus après la négociation.

- Caractéristiques :

  * voir la primitive CDCL § 7.2.1.3.3.1. (I)

2) Le format complet de la zone.

- Exemple :

	80	24	3	0	0	

- Valeurs limites des pointeurs : néant.

7.2.2.3.4. Zone caractères de contrôle IA5.
****

* cette zone est identique à celle de la primitive CDCL du § 7.2.1.3.4.

7.2.2.3.5. Zone séquences de contrôle.
****

* cette zone est identique à celle de la primitive CDCL du § 7.2.1.3.5.

7.2.2.3.6. Zone jeux de caractères.
****

* cette zone est identique à celle de la primitive CDCL du § 7.2.1.3.2.

7.2.2.3.7. Zone touches-fonctions.
****

* cette zone est identique à celle de la primitive CDCL du § 7.2.1.3.7.

7.2.2.3.8. zone zones formatées.
****

* cette zone est identique à celle de la primitive CDCL du § 7.2.1.3.8.

7.2.2.4. Les règles de traitement.
-.-.-.-.-.-.-.-.-.-.-.-.-

La succession des opérations va être la suivante :

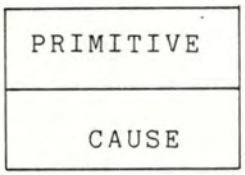
- la station VTP compare la liste des paramètres qu'elle a créée lors de la négociation avec celle qu'elle reçoit dans la primitive RDCLP.
- s'il n'y a pas de différence, nous passons à l'étape suivante dans la transmission des documents.
- s'il y a une discordance, nous réinitialisons la négociation par un envoi d'une primitive CDCL. Si nous avons déjà essayé une fois nous stoppons la communication.

7.2.3. La primitive de réponse négative à la négociation : RDCLN

7.2.3.1. Définition.

Si lors de la négociation, un paramètre est erroné ou si une incompatibilité au niveau matériel ou de l'application est détectée, nous envoyons la primitive de réponse négative (RDCLN) avec la cause de la rupture et nous terminons la communication.

7.2.3.2. Schéma



7.2.3.3. Description complète de la zone.

7.2.3.2.1. Zone primitive.

1) Cette zone est composée d'une partie.

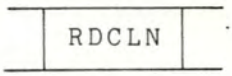
+ Définition : c'est la primitive de réponse négative à la négociation (RDCLN) (Response Document Capability List Negative).

+ Caractéristiques :

  * code hexa : 82
  * codée sur un byte.

2) Le format complet de la zone.

+ Exemple :



+ Valeurs limites des pointeurs:

7.2.3.3.2. Zone cause
****

1) Cette zone se compose d'une partie.

+ Définition : elle indique de manière codée la nature de la rupture dans la négociation.

+ Caractéristiques :

* codée sur un byte.
* Valeur 1 : une erreur de paramètre s'est produite.
2 : une incompatibilité de matériel ou de l'application.
* les autres codages appellent à un rejet de la primitive et l'envoi d'un RDGR.

2) Le format complet de la zone.

+ Exemple :

+ Valeurs limites des pointeurs : /

7.2.3.4. Les règles de traitement.
-.-.-.-.-.-.-.-.-.-.-.-.-.

La succession des opérations va être la suivante:

* s'il y a incompatibilité entre les terminaux qui négocient, il faut libérer la liaison car il n'y aura jamais de possibilité de travail.

* s'il y a une erreur de paramètre, nous réinitialisons la négociation par un envoi de la primitive CDCL. Si nous avons déjà essayé une fois de réémettre, nous stoppons la communication et nous signalons aux terminaux créateur et utilisateur qu'il y a un problème avec un des vecteurs de négociation.

7.2.4. La commande de début de document : CDS.

7.2.4.1. Définition.

Cette commande indique à la station VTP le début du document. A cette commande sont associés des paramètres qui permettent l'identification du document et l'initialisation de certaines caractéristiques du terminal utilisateur.

7.2.4.2. Schéma :

PRIMITIVE
IDENTIFICATION DU DOCUMENT
PARAMETRES D'INITIALISATION DU TERMINAL POUR LE DOCUMENT.

7.2.4.3. Description complète de chaque zone.

7.2.4.3.1. Zone primitive.

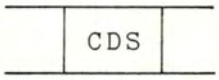
1) Cette zone est composée d'une partie.

- Définition : c'est la primitive CDS (Command Document Start).

- Caractéristiques :
  * code hexa 83
  * codée sur un byte

2) Format complet de la zone.

- Exemples :



- Valeurs limites des pointeurs : /

7.2.4.3.2. Zone identification du document.
****

1) Cette zone se compose d'une partie.

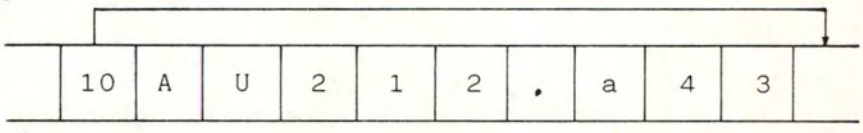
- Définition : Cette partie donne l'identification du document.

- Caractéristiques :

  * un pointeur
  * l'identificateur + en ASCII (3/0 → 7/14)
    + contient 20 caractères maximum
    + partie obligatoire.

2) Format complet de la zone.

- Exemples :



- Valeurs limites des pointeurs :

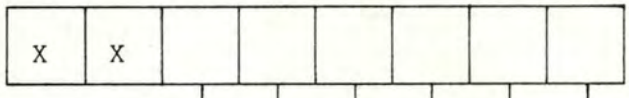
  a : $1 \leqslant$ pointeur $\leqslant 20$

7.2.4.3.3. Zone des paramètres d'initialisation du terminal pour le document.
****

1) Cette zone est composée de 6 parties.

a) Partie 1 : Sélection des modes de travail du terminal.

- Définition : cette partie donne les modes qui doivent être sélectionnés au terminal pour que l'application puisse se dérouler normalement.
  Ces modes sont de deux types :
  - Modes différents des modes de transmission (voir aussi annexe II
  - Modes de transmission.
- Caractéristiques :
  * un pointeur
  * les modes différents des modes de transmission : + 1 bit par mode
    + partie obligatoire.
    + repris sur un byte comme suit :



Block-Conversation Mode

0 : Conversation
1 : Block

Autowrap Mode;

0 : OFF
1 : ON

LF/NL

0 :LF
1 :NL

Autotab Mode.

0 : on reste en fin de zone non protégée courante.
1 : on passe à la zone non protégée suivante.

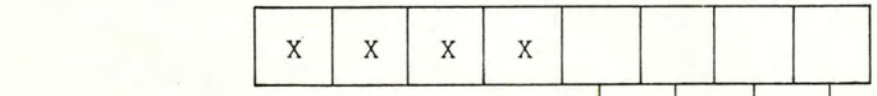
Erase Mode

0 : tout
1 : données zone non-protégée

Protect Form Mode

0 : on n'est pas dans ce mode.
1 : on travaille sur écran formaté

* les modes de transmission :
  + 1 bit par mode
  + repris sur un byte comme suit :



Transfert d'une ligne

0 : ligne
1 : page

Transfer Execution

0 : immédiate
1 : différée

Transfert zone protégée

0 : toutes
1 : où se trouve le curseur.

Transfer Termination

0 : page entière
1 : partie de page

  + les modes ne peuvent être utilisés que si le mode block est positionné : partie facultative

b) Partie 2 : Marges du document.

- Définition : Cette partie donne la position des marges verticales et horizontales d'une page en terme de positions absolues respectivement dans la ligne et dans la colonne.

- Caractéristiques :
  * un pointeur
  * positions des marges verticales :
    + ordre des paramètres :
      marge verticale gauche : MVG
      marge verticale droite : MVD
    + $\emptyset \leqslant MVG < MVD \leqslant Maxcar/MaxcarT$
    + en entier
    + obligatoires

* positions des marges horizontales :

  + ordre des paramètres :
    Marge horizontale supérieure : MHS
    Marge horizontale inférieure : MHI
  + $\emptyset \leqslant MHS < MHI \leqslant MaxligT$
  + en entier
  + obligatoires.

c) Partie 3 : Tabulations horizontales.

- Définition : Cette partie donne les positions des tabulations horizontales en terme de positions absolues dans ligne.

- Caractéristiques :

  * un pointeur
  * les positions des tabulations :
    + en entier
    + les positions sont dans un ordre strictement croissant.
    + non obligatoires

d) Partie 4 : Tabulations verticales.

- Définition : Cette partie donne les positions des tabulations verticales en terme de positions absolues dans la colonne.

- Caractéristiques :

  * un pointeur
  * les positions des tabulations:
    + en entier
    + en ordre croissant
    + non obligatoires

e) Partie 5 : Les programmes des fonctions.

- Définition : Cette partie donne les programmes à affecter aux différentes touches programmables.

- Caractéristiques :

  * un pointeur
  * le nombre de fonctions :
    + en entier
    + obligatoire
  * pour chaque fonction
    + un pointeur
    + le programme de la fonction.

f) Partie 6 : Jeu de caractères à employer pour le document.

- Définition : Cette partie permet de connaître le jeu de caractère requis pour un affichage correct du document.

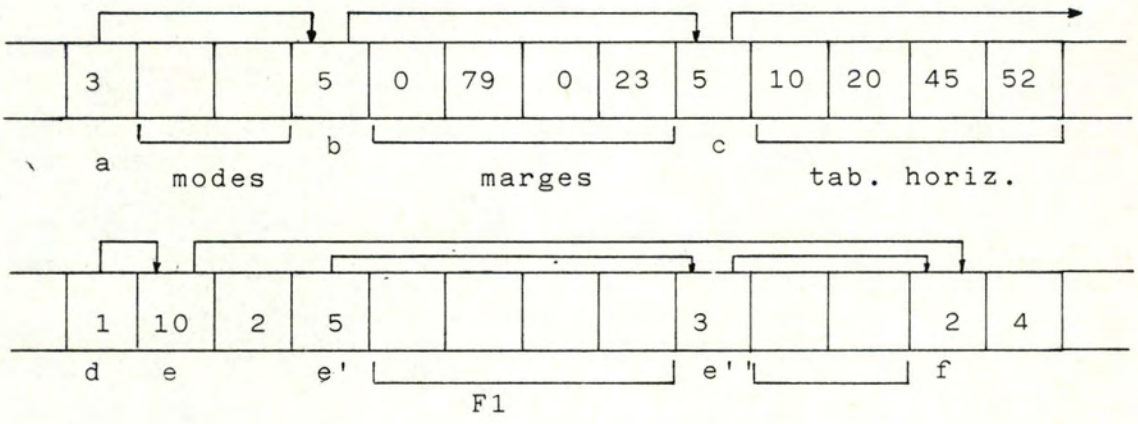
- Caractéristiques :

  * un pointeur
  * en entier obligatoire dont la valeur dépendra de la langue choisie :

    0 : anglais
    1 : U.S.A.
    2 : espagnol
    4 : français
    8 : allemand
   16 : italien
   32 : norvégien
   64 : portugais
  128 : suédois

2) Format complet de la zone.

- Exemple :



- Valeurs limites des pointeurs :

a : Min = 2
Max = 8

b : Valeur unique = 5

c : Min = 1
Max $\leqslant$ Maxcar

d : Min = 1
Max $\leqslant$ Maxlig

e : Min = 1
Max = 704

e', e"... = Min = 2
Max = 11

f : Valeur unique: 2

7.2.4.4. Les règles de traitements.

A la réception de la commande CDS, un certain nombre d'opérations différentes vont devoir être effectuées dont, d'une part, l'analyse des paramètres, l'initialisation du terminal sur base de ces données et, d'autre part, l'envoi d'un accusé réception positif ou négatif en fonction de la validité des paramètres.

7.2.4.4.1. Le traitement en général.

La succession des opérations va être la suivante :

* Analyse des zones contenant les paramètres de la commande (voir 7.2.4.4.2
* Si pas d'erreur :
  - si le module "gestion d'écran" est demandé, il y a mise à blanc de la mémoire écran et positionnement du curseur fictif en (0.0)
  - si "application demandant changement de sens" (maître-esclave), dernier numéro de page envoyé = Ø
  - renvoi vers le créateur d'une commande RDSP.
* Sinon, analyse des erreurs, éventuellement rejet par envoi de la commande RDGR avec comme paramètre la nature de l'erreur.

7.2.4.4.2. Analyse des zones contenant les paramètres de la commande.

Validation des pointeurs.

Au début de chaque partie, on valide les pointeurs. Ceci se fait sur base des valeurs limites trouvées dans le paragraphe intitulé "valeurs limites des pointeurs" de la description de chaque zone.

1. Zone primitive.

2. Zone identification du document.

   + Validation du type de caractères constituant l'identification du document.

   + Mémorisation dans la station VTP.

3. Zone des paramètres d'initialisation du terminal pour le document.

   + Analyse des modes de travail du terminal.

     * Modes différents des modes de transmission.

       - Validation : si "protect form mode" alors "block" dot être positionné
       - Pour tout bit envoyer la séquence correspondante au terminal.

     * Modes de transmission.

       - Validation
       - si mode block positionné alors envoi des séquences au terminal.

   + Analyse des marges.

     - Validation
     - Si le module "gestion d'écran" n'est pas demandé

       envoi des séquences d'initialisation du terminal avec les paramètres de la partie 2.

     Initialiser les variables de marge MVG, MVD, MHS et MHI avec les paramètres respectifs.

   + Analyse des tabulations horizontales.

     - Validation :

       . Le premier paramètre pris en considération sera celui dont la valeur entière sera supérieure ou égale à MVG. Les autres sont ignorés.

. Le dernier paramètre pris en considération sera celui dont la valeur entière sera inférieure ou égale à MVD. Les autres sont ignorés.

- Si les modules "tabulation horizontale" et "gestion d'écran" sont sélectionnés :

  chaque paramètre est stocké dans le vecteur adéquat de l'émulateur qui reprend les positions des tabulations spécifiques (TAB-AUX-H).

  Sinon

  envoyer la séquence d'initialisation du terminal avec les paramètres reçus.

+ Analyse des tabulations verticales.

  - Validation

    Idem sauf MVG → MHS
    MVD → MHI

  - Idem sauf "tabulation verticale" → "tabulation horizontale"

    " TAB-AUX-H" → "TAB-AUX-V".

+ Analyse des fonctions programmables.

  - Validation
  - Si le module "touches-fonctions" est sélectionné :

    .Initialiser le tableau de mémorisation des fonctions à Ø.

    .Mémoriser les divers programmes de fonctions dans le tableau de mémorisation des fonctions.

    Sinon

    Envoyer vers le terminal la séquence adéquate de programmation des touches.

+ Analyse du jeu de caractères à employer pour le document.

  - Validation

  - Mémoriser la valeur des paramètres dans la variable indiquant le langage utilisé.

    Si la langue demandée n'est pas émulée :

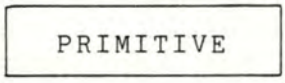
    envoi de la séquence correspondante au terminal.

7.2.5. Accusé réception positif au CDS : RDSP.

7.2.5.1. Définition.

Cette commande est un accusé réception par lequel la station VTP signale qu'elle a bien reçu tous les paramètres associés au CDS, que toute l'initialisation s'est effectuée correctement et qu'elle est prête à recevoir la première page d'information.

7.2.5.2. Schéma.



7.2.5.3. Description complète de la zone.

Zone primitive
****

Cette zone est composée d'une partie.

- Définition :

  C'est la primitive RDSP (ResponseDocument Start Positive).

- Caractéristiques :

  * code hexa 84
  * codée sur 1 byte

7.2.5.4. Les règles de traitement .

La réception de la commande RDSP autorise l'envoi des données suivantes si elles existent :

- Initialisation diverses
  - . Compteur d'émission à 1
  - . Dernier numéro de page envoyé = Ø
  - . Si "application demandant changement de sens" dernier numéro de page reçu = Ø.

. Status de mémorisation des causes possibles de refus de la page courante (voir RDPBN).

- Envoyer la commande CDUI + données.
- Envoyer la commande CDPB qui suscitera l'avis de réception.

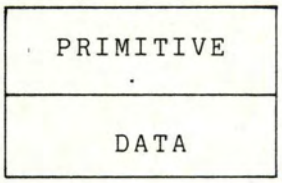
Au cas où il n'existe pas d'information à transmettre, une interruption s'impose et donc l'envoi de la commande CDD.

7.2.6. Primitive d'annonce de données : CDUI.

7.2.6.1. Définition.

Cette commande indique à la station VTP le début d'une zone de données.

7.2.6.2. Schéma.



7.2.6.3. Description complète de chaque zone.

7.2.6.3.1. Zone primitive.

Cette zone est composée d'une partie

- Définition : c'est la primitive CDUI.
- Caractéristiques :
  * code hexa 85
  * codée sur 1 byte.

7.2.6.3.2. Zone data.

Cette zone est composée d'une partie.

- Définition : les données effectives du transfert
- Caractéristiques :
  * caractères : ASCII.
  * longueur : 1 page émetteur
  * obligatoire.

7.2.6.4. Les règles de traitements.
-.-.-.-.-.-.-.-.-.-.-.-.-.-

Si le créateur et l'utilisateur sont déclarés, aux vues de la négociation, compatibles, tous les caractères reçus sont directement envoyés sans passer par le convertisseur de code.

Sinon, prévenir le convertisseur de code qu'il existe des caractères à traiter.

7.2.7. Primitive de marquage : CDPB.

7.2.7.1. Définition.

Cette commande indique à la station VTP le numéro du point de repère de la page.

7.2.7.2. Schéma.

PRIMITIVE
NUMERO DU POINT DE REPERE

7.2.7.3. Description complète de la zone.

7.2.7.3.1. Zone primitive.

Cette zone est composée d'une partie.

- Définition : la primitive CDPB
- Caractéristiques :
  * code hexa 86
  * codée sur 1 byte

7.2.7.3.2. Zone numéro de point de repère.

Cette zone est composée d'une partie.

- Définition : le numéro permet le séquencement des pages de données.
- Caractéristiques :
  * en entier
  * obligatoire

7.2.7.4. Les Règles de traitement.

La réception de la commande CDPB provoque le traitement suivant :

- Validation du séquencement

  . Mémorisation du numéro de page reçu.
  . Le numéro reçu est comparé au numéro précédemment reçu.
  . Si la séquence n'est pas respectée une erreur de séquence est positionnée dans le mode STATUS (STATUS-ERROR)

- Contrôle de STATUS-ERROR.

  . S'il n'existe pas d'erreur enregistrée
    envoi de la commande RDPBP avec comme paramètre le numéro de la page courante.

- Sinon

  . Côté créateur (cas des changements de sens) alors
    si le numéro de point de repère courant = le numéro de point de repère précédemment reçu alors

      Interruption et envoi de la commande CDD

    Sinon

      Mémoriser le numéro de point de repère de la page courante

      Envoi de la commande RDPBN.

  . Côté utilisateur :

    envoi de la commande RDPBN avec comme paramètre le mot de Status.

7.2.8. Primitive d'accusé réception positif : RDPBP.

7.2.8.1. Définition.

Cette commande, envoyée à la suite de la réception d'un CDPB, indique que le transfert de la page s'est déroulé correctement et que toutes les données ont été acceptées.

7.2.8.2. Schéma.

PRIMITIVE
NUMERO DU POINT DE REPERE

7.2.8.3. Description complète de chaque zone.

7.2.8.3.1. Zone primitive.

Cette zone est composée d'une partie.

- Définition : la primitive RDPBP.
- Caractéristiques :
  * code hexa 87
  * codée sur 1 byte.

7.2.8.3.2. Zone numéro du point de repère.

Cette zone est composée d'une partie.

- Définition : le numéro indique jusqu'à quel numéro de page l'acceptation est réalisée : indispensable dans le cas de fenêtre supérieure à une page.

- Caractéristiques :
  * en entier
  * obligatoire
  * dans notre cas, le numéro doit obligatoirement être égal à la valeur du point de repère reçu par le CDPB précédent.

7.2.8.4. Règles de traitement.
-.-.-.-.-.-.-.-.-.-.

La réception d'un RDPBP du côté créateur provoque les traitements suivants :

- Validation du numéro.
- S'il existe encore des données à transférer alors
  . Demander la page suivante
  . Envoi CDUI + données
  . Incrémenter la valeur du dernier numéro de page envoyé
  . Initialiser le compteur d'émission à 1
  . Mise à zéro des bits qui sont concernés par le transfert dans le STATUS-ERROR
  . Si la fin du document n'est pas dans la page envoyée
    - envoyer la commande CDPB qui suscitera l'avis de réception.
- S'il existe plus de données à transférer ou si la fin du document est dans la page envoyée alors
  . Envoyer la commande CDE avec comme paramètre le dernier numéro de page envoyé.

La réception d'un RDPBP du côté utilisateur ne provoque aucun traitement.

7.2.9. Primitive d'accusé réception négatif : RDPBN.

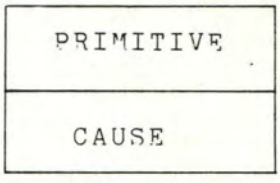
7.2.9.1. Définition.

Cette commande, envoyée à la suite de la réception d'un CDPB, indique qu'il n'accepte pas la page car il s'est passé une erreur ou une défaillance.

Les causes possibles du refus de la page sont:

- Saturation de la capacité mémoire (cas de fenêtre supérieure à 1).
- Erreur de séquence : la page reçue n'a pas le point de repère attendu.
- Erreur de terminal local : une séquence incluse dans la partie donnée (suite au CDUI) n'a pas été comprise par le terminal.
- Erreur de procédure.
- Autres raisons

7.2.9.2. Schéma.



7.2.9.3. Description complète de chaque zone.

7.2.9.3.1. Zone primitive.

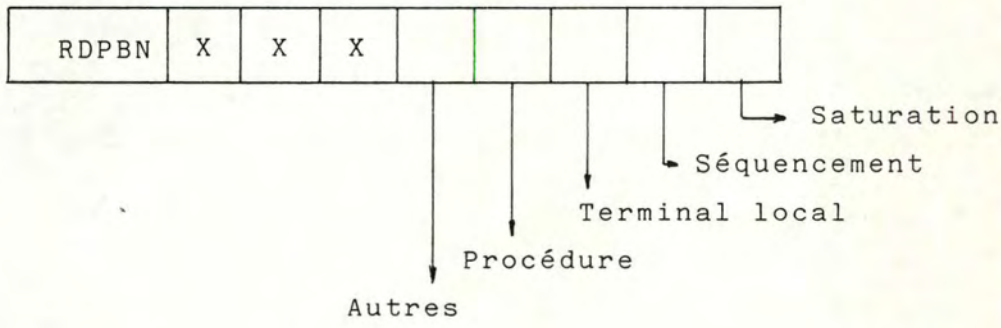
Cette zone est composée d'une partie.

- Définition : la primitive RDPBN.
- Caractéristiques :
  * code hexa 88
  * codée sur 1 byte.

7.2.8.3.2. Zone des causes du refus.
****

Cette zone est composée d'une partie.

- Définition :
  Causes du refus.
- Caractéristiques :
  * 1 bit positionné par cause de refus.
  * 1 byte.
  * Obligatoire.
  * Reçoit la valeur qui se trouve au moment de l'envoi de l'accusé dans le mot de STATUS (STATUS-ERROR).
- Format de la zone.



7.2.9.4. Règles de traitement.
-.-.-.-.-.-.-.-.-.-.-.-

La réception d'une commande RDPBN du côté créateur provoque le traitement suivant :

Si le compteur d'envoi = 2 alors interruption et envoi de la commande CDD.

Si le compteur d'envoi = 1 alors

. Analyse des erreurs.
. Incrémenter le compteur d'émission.
. Eliminer la cause de l'erreur si l'erreur = saturation, erreur séquence, terminal local

Envoi CDUI et les données.
Envoi CDPB avec le numéro de page envoyé

sinon interruption et envoi de la commande CDD.

La réception d'une commande RDPBN du côté utilisateur provoque le traitement suivant : sachant que cette commande n'est envoyée que pour provoquer la réémission de la page :

. Envoi CDUI et les données.
. Envoi CDPB avec le numéro de page envoyé.

En effet, l'utilisateur recevant cette commande uniquement lors des changements de sens, c'est le créateur qui se charge de compter le nombre d'émission que l'utilisateur a fait et envoie le RDPBN uniquement si nécessaire.

7.2.10 Primitive de fin de document : CDE.

7.2.10.1. Définition.

Cette commande indique la fin du document, en plus de transmettre à la station VTP le numéro du point de repère de la dernière page envoyée.

7.2.10.2. Schéma.

PRIMITIVE
NUMERO REPERE

7.2.10.3. Description complète de chaque zone.

7.2.10.3.1. Zone primitive.

Cette zone est composée d'une partie.

- Définition : la primitive CDE.
- Caractéristiques:
  * code hexa 89
  * codée sur 1 byte.

7.2.10.3.2. Zone du numéro de point de repère.

Cette zone est composée d'une partie.

- Définition : Ce numéro permet d'acquiesser les dernières informations de l'application et du document.
- Caractéristiques :
  * en entier
  * obligatoire.

7.2.10.4. Règles de traitement.

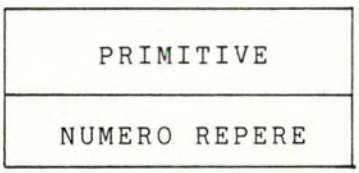
Idem CDPB sauf que si pas d'erreur envoi RDEP.

7.2.11. Accusé réception au CDE : RDEP.

7.2.11.1. Définition.

Cette commande prévient le créateur que l'utilisateur a bien reçu la commande de fin de document et a bien reçu la dernière page envoyée.

7.2.11.2. Schéma.



7.2.11.3. Description complète de chaque zone.

7.2.11.3.1. Zone primitive.

Cette zone est composée d'une partie.

- Définition : la primitive RDEP.
- Caractéristiques :
  * code hexa 8A
  * codée sur 1 byte.

7.2.11.3.2. Zone du numéro de point de repère.

Cette zone est composée d'une partie.

- Définition :
- Caractéristiques :
  * en entier
  * obligatoire.

7.2.11.4. Règles de traitement.

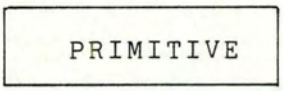
Effectue la libération de la communication.

7.2.12. Commande de changement de sens : CDCS.

7.2.12.1. Définition.

Cette commande, envoyée par le créateur, autorise l'utilisateur à envoyer une et une seule page de données.

7.2.12.2. Schéma.



7.2.12.3. Description complète de chaque zone.

Zone primitive.

Cette zone est composée d'une partie

- Définition : la primitive CDCS.
- Caractéristiques :
  * code hexa 8 B
  * codée sur 1 byte

7.2.12.4. Règles de traitement.

A la réception du CDCS, l'utilisateur :

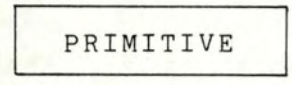
- Envoit RDCS.
- Demande la page.
- Envoi CDUI + données.
- Incrémenter la valeur du dernier numéro de page envoyée.
- Envoyer la commande CDPB avec le numéro de page envoyé.

7.2.13. Accusé réception du changement de sens : RDCS.

7.2.13.1. Définition .

Cette commande, envoyée par l'utilisateur signifie que ce dernier accepte le changement de sens.

7.2.13.2. Schéma.



7.2.13.3. Description complète de chaque zone.

Zone primitive.

Cette zone est composée d'une partie.

- Définition : la primitive RDCS.
- Caractéristiques :
  * code hexa 8C
  * codée sur 1 byte.

7.2.13.4. Règles de traitement.

Le créateur qui reçoit un RDCS

. Mise à zéro des bits de STATUS - ERROR de la page à recevoir.

C H A P I T R E VIII

ETUDE DETAILLEE DES MODULES DE L'EMULATEUR.

CHAPITRE VIII : ETUDE DETAILLE DES MODULES DE L'EMULATEUR.

Ce chapitre détaille les modules suivants :

1 - Le module "pliage"
2 - le module "saut de page"
3 - le module "gestion d'écran"
4 - le module "traitement de texte"
5 - le module "touches fonctions"
6 - le module "jeu de caractère"
7 - le module "zones formatées"
8 - le module "convertisseur de codes".

8.1. Le module "pliage".

Ce module plie en deux toutes les lignes qui sont plus grandes que le format accepté par le terminal.

8.1.1. Les entrées.

Le module reçoit un caractère
Celui-ci : appartient à la grille ASCII n°5
est imprimable au terminal (32 → 128)
n'appartient pas à une séquence de contrôle.

8.1.2. Le traitement.

Le module teste,en lisant la position du curseur dans la ligne, si l'impression du caractère ne nécessite pas un passage à la ligne.

Si oui, on envoie, avant le caractère, les caractères de contrôle CR et LF.

On envoie le caractère au terminal.

8.1.3. Les sorties.

Le caractère ASCII d'entrée et éventuellement un CR et LF.

N.B. Pour le calcul du saut à la ligne, il est tenu compte d'éventuelles marges verticales à gauche et/ou à droite.

8.1.4. Algorithme.

Voir annexe III

8.2. Le module "saut de page".

Ce module répartit sur deux pages, le contenu d'une page "origine" dont le format est plus grand que celui accepté par le terminal.

8.2.1. Les entrées.

Le module reçoit un caractère.
Celui-ci : appartient à la grille ASCII n°5
est imprimable au terminal (32 → 128)
n'appartient pas à une séquence de contrôle.

8.2.2. Le traitement.

Le module teste si l'impression du caractère ne nécessite pas un passage à la page suivante sur le terminal, TTY ou imprimante.
Si oui, il envoie avant le caractère d'entrée, le caractère de contrôle FF.
On envoie le caractère.

8.2.3. Les sorties.

Le caractère ASCII d'entrée et éventuellement un FF constituent les données en sortie.

N.B. Pour le calcul du saut de page, il est tenu compte des éventuelles marges en haut et/ou en bas de la page.

8.3. Le module "gestion-écran".

Introduction

Si le format du terminal utilisateur est plus petit que celui sur lequel a été crée l'application, nous devons plier certaines lignes trop longues (voir module pliage n° 8.1.).
Il devient alors impossible de connaître la position du curseur dans son écran d'origine. Or, certaines applications ne travaillent que par positionnement du curseur et lecture de sa position.
Il était nécessaire de remédier à ce problème en créant le module "gestion - écran".
Ce système n'est utilisable que sur les terminaux de type vidéo, car ils sont les seuls à pouvoir effacer un texte imprimé et à en réécrire un autre au même endroit.

Principes.

Voyons en détails le fonctionnement de ce module.
Nous disposons dans l'émulateur d'une matrice pouvant contenir 160 caractères par ligne et 48 lignes par page. (Ces dimensions correspondent aux longueur et largeur maximales rencontrées. Ces deux valeurs ne sont pas limitatives et peuvent être augmentées à volonté).
Cette matrice contient le code des caractères à imprimer.
Une seconde matrice semblable a été créée pour contenir les attributs de visualisation des caractères.

Prenons un exemple :

Dans la matrice caractère, à la position 20,21 se trouve le code 65 qui correspond à la lettre A.

Dans la matrice attribut, à la même place, se trouve le code 5 qui indique que le caractère A est souligné (voir annexe II)

Sur la matrice caractère, on définit une "fenêtre mobile" de longueur (caractères/ligne) et largeur (lignes/page) égales au format de l'écran du terminal utilisateur. Cette fenêtre permet de visualiser une partie de la matrice caractère.

Pour voir, les parties se trouvant hors de la fenêtre, nous disposons de quatre commandes :

- une pour remonter la fenêtre d'une ligne
- une pour descendre la fenêtre d'une ligne
- une pour avancer la fenêtre d'un caractère vers la droite
- une pour reculer la fenêtre d'un caractère vers la gauche

Nous allons faire deux remarques importantes :

- ces commandes bien qu'elles ressemblent aux touches du curseur n'ont rien à voir avec elles. Elles appartiennent à un clavier indépendant.
- même si nous disposons d'une matrice caractère de 160*48, nous utilisons en réalité une matrice de longueur et de largeur correspondant aux valeurs négociées au début de l'application.

En résumé, on doit distinguer trois dimensions bien différentes à savoir :

- la matrice caractère de 160*48
- la matrice utile dont la longueur et la largeur sont issues de la négociation
- la fenêtre dont la longueur et la largeur sont définies par le support visuel.

Modifications des séquences de contrôle.

Ce système nécessite la modification du traitement de la plupart des séquences de contrôle du terminal.
En effet, si nous désirons effacer une ligne, il ne faut pas seulement effacer la ligne dans la fenêtre mais en plus il faut effacer la ligne dans la matrice utile.

Introduction d'un caractère.

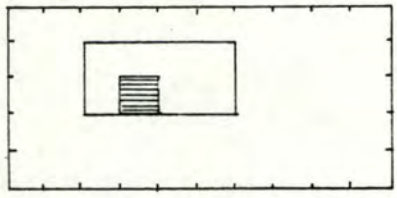
Voyons maintenant ce qui se passe quand nous introduisons un caractère au terminal.

- tant que nous nous trouvons dans les limites de la fenêtre, il ne se passe rien
- si nous atteignons une limite de la fenêtre, tout le texte doit se décaler dans la direction voulue pour pouvoir insérer le prochain caractère
- si nous atteignons les limites de la matrice utile, la fenêtre ne bouge plus et le nouveau caractère est surimprimé sauf si le mode autowrap est utilisé auquel cas on passe à la ligne suivante et la fenêtre se déplace.

Gestion du curseur.

L'émulateur va gérer un curseur que nous appellerons curseur fictif: Il correspond au curseur qui se trouverait sur un écran ayant les dimensions de l'écran de l'émetteur.
Il est évident que c'est ce curseur qui est utilisé lors du dialogue entre l'application et l'émulateur.
Sa position correspond exactement à celle utilisée lors de la création de l'application.
Par contre, le curseur du terminal utilisateur ne sert plus qu'à pointer le prochain caractère à introduire sur l'écran.

Prenons un exemple :



matrice utile 10*5
fenêtre 4 *2
position du curseur fictif : 4,3
position du curseur écran : 1,1

l'application suppose que le curseur se trouve en 4,3 donc dans la position du curseur fictif.

Remarques.

Le principe peut s'adapter à tous les modes de terminaux scroll, page ou data entry.
Dans le cas d'un scroll mode, le remplissage de la matrice utile se fait en rouleau.
Dans les deux autres cas, le remplissage commence toujours à la position "home" du curseur.

Avantages et inconvénients du système.

Avantages :

- Ce module permet de gérer le curseur facilement et la matrice utile comme un écran réel. Ainsi l'application ne voit aucune différence avec un écran réel.
- Le texte apparaît dans son format d'origine.

Inconvénients :

- Il nécessite de modifier presque toutes les séquences de contrôle.
- Il nécessite une place mémoire importante.
- Il ne parmet pas de voir tout le texte en une fois.
- Il nécessite l'usage d'un clavier annexe.

Conclusions

D'après ce qui précède, il semble normal que nous ayons limité l'usage de ce système aux applications qui utilisent le curseur de manière intensive, c'est-à-dire les applications "questions-réponses" et les "zones formatées". (Celles-ci travaillent toujours en"mode page" et "data entry".).

Ce module se découpe en 5 parties :

- interprétation des touches de commande
- affichage de la fenêtre
- mémorisation d'une page
- Lecture d'une page ou d'une partie de page
- mise à jour du curseur fictif.

8.3.1. Interprétation des touches de commande.

8.3.1.1. Les entrées.

Le sous-module reçoit le caractère de contrôle du clavier indépendant.

Caractéristiques des touches.

↑ : codage : Ctrl A
signification : commande utilisée pour remonter la fenêtre d'une ligne vers le haut.
valeur limite : la ligne zéro de la matrice utile.

↓ : codage : Ctrl B
signification : commande utilisée pour descendre la fenêtre d'une ligne.
valeur limite : la dernière ligne de la matrice utile.

→ : codage : Ctrl C
signification : commande utiliséepour avancer la fenêtre d'un caractère vers la droite.
valeur limite : le dernier caractère de la matrice utile.

← : codage : Ctrl D
signification : commande utilisée pour avancer la fenêtre d'un caractère vers la gauche.
valeur limite : le caractère zéro de la matrice utile.

8.3.1.2. Le traitement.
-.-.-.-.-.-.-.

Le traitement consiste à mettre à jour la position du coin supérieur gauche de la fenêtre dans la matrice utile suivant les règles ci-après :

touche enfoncée	abscisse	ordonnées
→	+ 1	X
←	- 1	X
↑	X	- 1
↓	X	+ 1

et à afficher la nouvelle fenêtre.

8.3.1.3. Les sorties.
-.-.-.-.-.-.-.

8.3.1.4. Algorithmes.
-.-.-.-.-.-.-.

Voir annexe III.

8.3.2. Affichage de la fenêtre.

8.3.2.1. Les entrées.

8.3.2.2. Le traitement.

En connaissant deplX et deplY, nous envoyons au terminal les caractères compris dans un rectangle dont les sommets ont comme coordonnées :

- deplX, deplY
- deplX + maxcarT, deplY
- deplX, deplY + maxligT
- deplX + maxcarT, deplY + maxligT

8.3.2.3. Les sorties.

Les données de sortie sont constituées par les caractères compris dans le rectangle précédemment défini.

8.3.2.4. Algorithme.

Voir annexe III.

8.3.3. Mémorisation d'une page.

8.3.3.1. Les entrées.

Les données d'entrée sont formées par une page de caractères venant de l'application.

8.3.3.2. Le traitement.

Il faut bloquer le clavier afin d'empêcher l'utilisateur d'introduire des données pendant que l'application envoie des caractères.

On mémorise tous les caractères envoyés en partant de la position X = 0 et Y = 0 de la matrice utile.
On libére le clavier.

8.3.3.3. Les sorties.
-.-.-.-.-.-

Les données de sortie sont constituées par la matrice utile remplie par les caractères de la page d'entrée.

8.3.3.4. Algorithme.
-.-.-.-.-.-

Voir annexe III.

8.3.4. Lecture d'une page ou d'une partie.
- - - - - - - - - - - - - - - - - - -

Cette partie est traitée par le convertisseur de code dans le traitement des séquences de contrôle de transmission.

8.3.5. Mise à jour du curseur fictif.
- - - - - - - - - - - - - - - - -

8.3.5.1. Les entrées
-.-.-.-.-.-

Les entrées sont produites lorsqu'on introduit un caractère au clavier, ou lorsqu'on utilise une séquence de contrôle relative au curseur.

8.3.5.2. Le traitement.
-.-.-.-.-.-.-

On met à jour la position du curseur X et Y.
On vérifie s'il faut déplacer la fenêtre afin que le curseur sur l'écran soit toujours visible et pointe bien le prochain caractère à introduire.

8.3.5.3. Les sorties.
-.-.-.-.-.-

Une nouvelle fenêtre peut être affichée sur l'écran et le curseur réel peut avoir changé de place.

8.3.5.4. Algorithme.
-.-.-.-.-.-

Voir annexe III.

8.4. Le module traitement de texte.

8.4.1. Introduction.

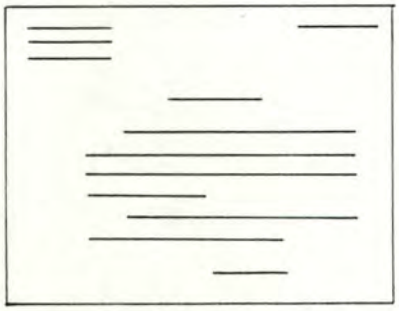
Ce module est activé uniquement lorsqu'un utilisateur traite une application de type traitement de texte.

Il permet de reformater le texte reçu du créateur après avoir déterminé le type de chaque ligne.

Le texte doit avoir les caractéristiques suivantes :

- ne contenir que des lignes de texte dont la définition du type est donnée au paragraphe suivant,
- être justifié à gauche et à droite à l'exception de certaines lignes telles les débuts et fins de paragraphes,..
- contenir éventuellement des marges à gauche, à droite, en haut et en bas,
- ne pas contenir de dessins ou de tableaux
- ne pas contenir de séquences de contrôle.

Reprenons l'exemple du chapitre IV.



Après reformatage sur un support visuel plus petit.

Nous remarquons bien que l'aspect général du texte est resté celui donné à l'origine par le créateur.
Ainsi, on y retrouve les titres, les paragraphes et autres caractéristiques.
Nous constatons que les lignes sont plus courtes mais évidemment plus nombreuses.
Les marges en haut, en bas, à gauche et à droite persistent mais sont réduites d'un facteur égal au facteur de réduction du format de la page.

Remarques :

- Les tableaux et les figures ne peuvent être traités comme du texte sous peine de les retrouver, en sortie, justifiés à gauche et à droite, et étalés sur plusieurs lignes.
  Si un texte en possède, il faut considérer l'application comme une application de texte libre et utiliser le module "gestion-écran".
- Si des séquences de contrôle apparaîssent dans le texte elles sont comptées comme des caractères imprimables.
  C'est pourquoi la justification s'en trouve faussée.
  Notons que normalement, ce genre de séquence n'est pas présente dans un texte..

8.4.2. Définition des types de lignes.

Le module ne reformate convenablement que les lignes répondant aux définitions suivantes :

Voir pages 194,195.

Si une ligne a un format n'appartenant à aucun type repris ci-dessus, son traitement devient aléatoire.

Nous reprenons dans le tableau suivant page 194 toutes les combinaisons possibles de succession de 2 types de lignes.
Celles marquées d'une croix sont considérées comme impossibles à trouver dans un texte.

8.4.3. Gestion de la mémoire texte.

8.4.3.1. Fonctionnement général.

Les lignes de texte sont mémorisées par page entière dans l'émulateur.

Le traitement d'une ligne se fait en deux temps :

+ Dans un premier temps, le module analyse la ligne caractère par caractère et différents renseignements sont mémorisés :
  - le début de chaque mot avec sa longueur
  - le nombre total de mots dans la ligne
  - le dernier caractère différent d'un blanc et sa position
  - la longueur du retrait au début de la ligne pour un paragraphe ou une énumération
  - s'il existe un blanc en fin de ligne
  - s'il existe un grand blanc (10 caractères) dans la ligne au milieu des mots, sa longueur et sa position par rapport au début de la ligne

Ligne blanche

2 ----------

Ligne de paragraphe

11 -----———————————

12 -----———————————·---------

13 -----———————————:---------

14 ———————————————————

15 ———————————————·---------

16 ———————————————:---------

Ligne d'énumération

21 --------———————————

22 --------———————————---------

23 --------———————————:---------

Lignes de titre

31 ——————————·····

32 - - - - - - - ——————————- - - - - -

Lignes de soulignés

41 ——————————- - - - - - - - - - -

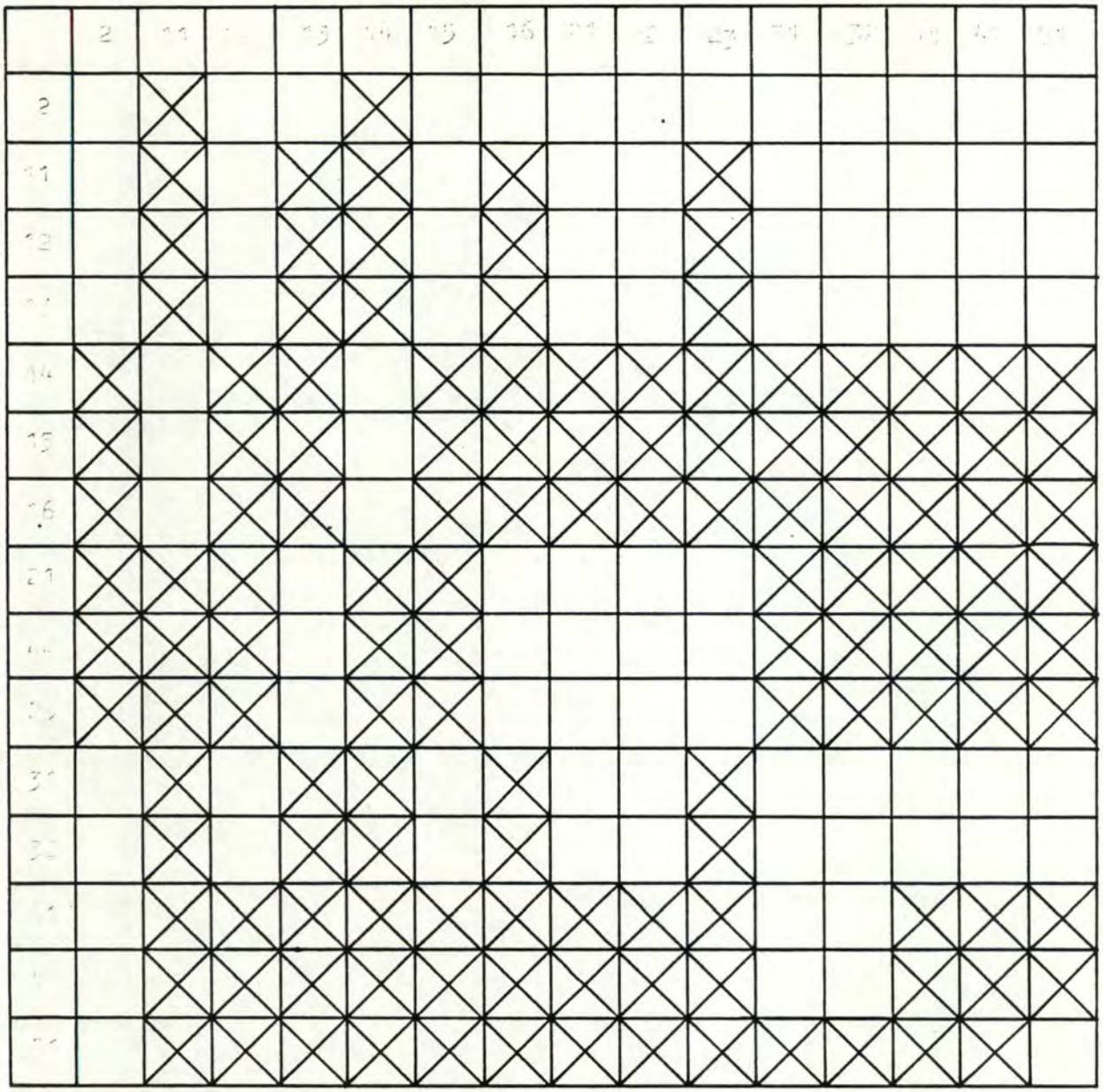
42 - - - - - - - - ——————————- - - - - - - -

Lignes d'entête

51 - - - - ——————— - - - - - - - - - ——————— - - -

Légende ; ——— texte

- - - - - - blancs



Dans le tableau ci-dessus, les nombres inscrits en abscisse et ordonnée représentent les types de lignes définis dans les tableaux p194,195.

Ce tableau permet de définir quel type de lignes (en ordonnée) peut (carré blanc) ou ne peut pas (carré indiqué d'une croix) suivre un type de ligne donné (abscisse)

- si la ligne est composée d'une suite du même caractère, et le caractère utilisé.

Grâce à ces renseignements, le module tente de déterminer le plus justement possible le type de la ligne afin de pouvoir traiter la précédente.

+ Dans un second temps, il reprend cette ligne pour la reformater.
Le fait de connaître la nature de la ligne suivante nous permet d'ajouter un raffinement supplémentaire dans le choix du type de la ligne.
Une fois ce type déterminé, on reformate la ligne grâce à tous les renseignements recueillis lors de l'analyse (nombre de mots, leur longueur...).
On tient évidemment compte des éventuels mots restants de la ligne précédente, de même que certains mots sont reportés pour la ligne suivante.

8.4.3.2. Exemple.
-.-.-.-

_____ (1)

_____ (2)

_____. (3)

Supposons l'analysede la ligne (1) réalisée ; le type de la ligne est "début de paragraphe".

- a - On analyse la ligne (2) : type "ligne de paragraphe"
On traite alors la ligne (1) c'est-à-dire :

- on prend les X premiers mots pouvant être contenu dans le format du support de sortie
- on mémorise les autres

- on justifie la nouvelle ligne (1)
- on l'envoie au terminal.

-b - On analyse la ligne (3) : type "fin de paragraphe"
On traite la ligne (2) c'est-à-dire :

- on prend les mots restants de la ligne (1) plus les Y premiers mots de la ligne (2) pouvant être contenu dans le format du support de sortie.
- on réalise la même chose que pour la ligne (1). On recommence ceci jusqu'à ce qu'il n'y ait plus assez de caractères dans la ligne (2) pour faire une ligne complète.

- c - On fait la même chose qu'au point -b- pour la ligne (3). On obtient en sortie.

_____ (1)

_____ (2)

_____ (3)

_____ . (4)

La méthode pour reconnaître chaque type de ligne sera expliqué dans les paragraphes suivants (8.4.4.)

8.4.4. Les algorithmes.

8.4.4.1. Le coordinateur.

Le coordinateur permet d'analyser et de traiter séquentiellement toutes les lignes de texte mémorisées dans l'émulateur et ceci jusqu'à la réception de la primitive -fin de texte-.

8.4.4.1.1. Les entrées.

Les données d'entrée de ce module sont constitutées par les lignes de texte mémorisées dans la matrice ECRAN.

8.4.4.1.2. Le traitement.

+ Le module recopie la ligne à analyser dans un tableau.
+ Il analyse la ligne en appelant la routine adéquate (trt_analyse)
+ Il traite la ligne précédemment enregistrée et analysée grâce à la routine (trt_phrase).
+ On recommence ces trois étapes aussi longtemps qu'une des deux conditions suivantes est remplie
  - on est à la fin du texte émis,
  - il n'y a plus qu'une ligne en mémoire (pas de traitement possible) et on n 'est pas en fin de texte.

Dans le premier cas, on termine le traitement de la dernière ligne en supposant que la ligne suivante est blanche.
Dans le deuxième cas, on attend que la station VTP remplisse la matrice ECRAN.

N.B. : Le remplissage de la matrice ECRAN se fait modulo 48 c'est-à-dire que si on est sur la dernière ligne (ECRAN [47,X]) la ligne suivante est mémorisée à la place de la première (ECRAN [0,X])

Le remplissage de la matrice s'arrête lorsque le pointeur indiquant le numéro de la ligne analysée égale le pointeur indiquant le numéro de la dernière ligne introduite plus 1.

8.4.4.1.3. Les sorties.
****

8.4.4.1.4. Les algorithmes.
****

Voir annexe III.

8.4.4.2. L'analyse de la ligne.
-.-.-.-.-.-.-.-.-.-.-

8.4.4.2.1. Les entrées.
****

Les données d'entrée sont formées d'une ligne de texte contenue dans un tableau (TABLEAU_LIGNE [0...159]) et d'un paramètre (a), qui indique dans quelle ligne du tableau précédent il faut lire les informations.

8.4.4.2.2. Le traitement.
****

+ Le module lit la ligne jusqu'à ce qu' il rencontre un "carriage return" (CR) ou que le compteur de caractères est au maximum.

+ Si le caractère est un blanc, trois cas peuvent se produire :

1) Ce ou ces blancs se trouvent avant le premier mot de la ligne, et ils indiquent donc :

- un début de paragraphe
- un titre en retrait
- une ligne d'énumération
- une ligne blanche
- une entête en retrait.

Il est important de reproduire ce blanc en début de texte reformaté, il faut le mémoriser comme un mot de type différent d'un mot normal.

2) Ce ou ces blancs se trouvent entre deux mots et le nombre est inférieur à 5. Dans ce cas, on n'en mémorise qu'un, d'autres étant éventuellement ajoutés lors de la rejustification du texte.

3) Ces blancs se trouvent entre deux mots et leur nombre est supérieur ou égal à 5. Ils ne proviennent pas de la justification précédente du texte. Ils montrent la présence d'une phrase de type entête.
On enregistre un mot, sa position et on positionne une variable pour indiquer la rencontre de ce type de mot dans la ligne.

+ Si le caractère est différent du blanc, deux cas sont possibles :

1) Tous les caractères de la ligne sont identiques, on a affaire à une ligne de type souligné.
On mémorise le caractère servant à souligner et une variable indiquant la rencontre de ce type de ligne.

2) Les caractères sont différents, ce sont des mots.
On mémorise : - l'adresse du début de chaque mot,
- la longueur du mot,
- son type (mot,blanc,...)
dans une pile organisée FIFO.

+ Le module mémorise enfin :

1) le nombre total de caractères et de mots dans la ligne.

2) l'adresse et la valeur du dernier caractère non nul.

+ Il détermine avec tous les paramètres mémorisés, la nature présumée de la ligne qu'on vient d'analyser.

8.4.4.2.3. Les sorties
****

Les données de sortie sont constituée par :

- la phrase d'entrée dans le TABLEAU_LIGNE
- la pile FIFO
- différentes variables
- la nature présumée de la ligne.

8.4.4.2.4. Les algorithmes.
****

Voir annexe III

8.4.4.3. Le traitement de la ligne.
-.-.-.-.-.-.-.-.-.-.-.-.-.-.

8.4.4.3.1. Les entrées.
****

Les données d'entrée sont formées par :

- la file TAB

- les variables indiquant la nature des lignes analysées et à traiter.
- le tableau contenant la ligne à traiter et les restes de la précédente.

8.4.4.3.2. Le traitement.
****

Le module raffine le type de la ligne grâce à l'analyse de la suivante qui vient d'être faite.

Grâce à ce type découvert, on active la routine correspondante.

Chaque traitement de ce type est détaillé plus loin.

8.4.4.3.3. Les sorties.
****

Les sorties sont formées par la ligne justifié suivant le nouveau format.

8.4.4.3.4. Les algorithmes.
****

Voir annexe III

8.5. Le module "touches - fonctions".

Ce module permet à un terminal de programmer les touches de fonctions s'il ne peut le réaliser en standard.

Il est évident que ce module ne peut être activé que si le terminal dispose de ce genre de touche au clavier.

Ce module se divise en deux parties :

+ l'enregistrement du contenu des touches-fonctions.
+ la lecture du contenu des touches-fonctions.

8.5.1. L'enregistrement des touches-fonctions.

8.5.1.1. Les entrées.

Les données d'entrée sont constituées de la séquence de contrôle suivante :

ESC p PF1 ; PF2 ; ... PFn ;

Les caractéristiques de cette séquence sont :

+ PFi = ensemble de caractères formant le texte mémorisé pour la touche fonction i
+ Ce texte peut contenir :
  - 1 à 10 caractères
  - 0 caractère si on désire garder le texte précédemment enregistré pour cette touche.
  - "*" si ce caractère se trouve en première position et qu'il est seul dans PFi, il indique qu'on désire programmer la touche i avec la séquence standard. (ESC Ø i).
  - un caractère peut être n'importe quel caractère ASCII n°5 y compris un caractère de contrôle sauf le ";"
  - aucune zone PFi ne peut être omise.

8.5.1.2. Le traitement.
-.-.-.-.-.-.-

Il faut réaliser la validation de chaque PFi afin qu'il suive les règles énoncées au 8.5.1.1.

Cette validation va porter sur :

+ le nombre maximum de caractères dans la zone
+ le fait que s'il y a une "*" en première position elle doit être seule dans le PFi (entre deux ;)
+ le nombre maximum de fonctions PFi qu'on peut programmer suite à la négociation.

Il faut ensuite pour chaque PFi correct :

+ mémoriser le texte, ou
+ mémoriser la séquence de contrôle standard, ou
+ garder le texte précédent.

8.5.1.3. Les sorties.
-.-.-.-.-.-

Si les PFi sont corrects, les données de sortie sont composées du vecteur formé par l'enregistrement des différents PFi.

Si un des PFi est érroné, les données de sortie sont formées par un message d'erreur.

8.5.1.4. Algorithme.
-.-.-.-.-.

Voir annexe III

8.5.2. La lecture du contenu des touches-fonctions.

8.5.2.1. Les entrées.

Lorsqu'on enfonce une des touches-fonctions au clavier, on reçoit la séquence de contrôle standard (ESC Ø i)

Le i correspond au numéro de la touche enfoncée.

8.5.2.2. Le traitement.

On vérifie que le numéro de la touche enfoncée est plus petit que le nombre maximum de touches programmables.

S'il n'y a pas d'erreur, on lit le contenu de la touche fonction et on le transfère dans la station VTP.
Sinon, on ne fait rien.

8.5.2.3. Les sorties.

Les données de sortie sont formées par le contenu de la touche fonction si celle-ci pouvait être programmée.

Sinon, aucune donnée n'est transmise à la station VTP.

8.5.2.4. Algorithme

Voir annexe III

8.6. Le module "jeu de caractères".

Chaque langue nationale a un certain nombre de caractères particuliers comme le ê, é, è... en français.
Ceux-ci ont un codage dans la grille ASCII N°5 et ils remplacent certains caractères normalisés.

Par exemple en français le "é" est codé comme un "#" de la grille ASCII.

Si le terminal ne possède pas l'alphabet français, en recevant le "é", il va imprimer un caractère "#". Le but de ce module va être de convertir le "é" en un"e", afin de rendre le texte lisible malgré certaines fautes d'accent.

8.6.1. Les entrées.

Le module reçoit en entrée le caractère à convertir et la langue utilisée.

Ce caractère appartient à la grille ASCII N° 5.
Il doit ou non être converti.

8.6.2. Le traitement.

Grâce à des tables de conversion, on obtient le code correspondant à un caractère lisible.

8.6.3. Les sorties.

Le caractère converti constitue la donnée de sortie.
Il appartient à la grille ASCII N° 5 U.S.A.

8.6.4. L'algorithme.

Voir annexe III

8.7. Le module "zones formatées".

Ce module permet à un vidéo "scroll mode" ou "page mode" de devenir un vidéo "data entry mode" en lui ajoutant la possibilité d'analyser localement les caractères introduits à partir du clavier dans différentes zones définies sur l'écran.

Il y a deux problèmes à résoudre :

- enregistrer dans l'émulateur la position et les attributs de chaque zone ainsi que le texte qui doit apparaître à l'écran.
- réaliser dans l'émulateur l'analyse et la mémorisation des caractères introduits au clavier par l'utilisateur.

8.7.1. L'enregistrement des zones formatées.

Lorsque le convertisseur de code détecte la séquence de contrôle annonçant l'entrée dans le mode de formatage, il active ce sous-module enregistrement.

8.7.1.1. Les entrées.

Les données d'entrée sont constituées par des séquences de contrôle normalisées constituées de différentes parties :

- caractère de début de séquence (DC1)
- position en abscisse et ordonnée du début de la zone
- les attributs de la zone
- le texte (facultatif)
- caractère de fin de séquence (DC3)

D'après la définition du langage standard, aucune séquence de contrôle ne peut se trouver dans le texte à l'exception des séquences de visualisation.

8.7.1.2. Le traitement.
-.-.-.-.-.-.-..

Il faut commencer par initialiser la matrice utile à "nul", et la matrice attributs à la valeur "caractère normal" et vider l'écran.

Pour chaque zone formatée , on enregistre dans une file.

- l'adresse absolue de la fin de la zone,
- l'adresse absolue du début de la zone,
- les attributs de la zone.

Pour chaque zone de texte, il faut :

- analyser chaque caractère
- s'il existe une séquence de contrôle de visualisation, il faut alors remplir la matrice attribut,
- s'il existe une autre séquence de contrôle, on l'élimine,
- si le caractère est normal, on le mémorise dans la matrice utile,et sur l'écran si on se trouve dans la fenêtre,
- mémoriser la position du début de chaque zone non-protégée. Celle-ci sera utilisée lors de la manipulation de la touche tabulation -TAB-

Si une erreur se produit dans la forme de la séquence de contrôle, rendre la main au convertisseur de code et signaler une erreur à la station VTP.

8.7.1.3. Les sorties.
-.-.-.-.-.-.

Les données de sortie sont constituées par trois entitées :

- la matrice utile contenant le texte formaté,
- une file contenant le début, la fin et les attributs de chaque zone,
- une file contenant le début de chaque zone non-protégée.

N.B. : a) Ce sous-module rend la main au convertisseur de code lorsqu'il détecte la séquence de contrôle fin du formatage.

b) La file contenant les attributs de chaque zone, est initialisée avec les valeurs suivantes :

début zone : adresse 0
fin zone : adresse maxlig * maxcar
attribut : non protégé.

Ceci permet d'ajouter ou de retirer les éléments de la file et de traiter tous les cas comme le cas général.

c) Les types d'attributs :

8	7	6	5	4	3	2	1

1 : protégé
2 : affichable
3 : numérique
4 : alphabétique
5 : digit
6 : alphanumérique
7 : sensible au crayon lumineux
8 : remplissage complet

8.7.1.4. Algorithmes.
-.-.-.-.-.-.

Voir annexe III

8.7.2. Le traitement de caractère.
- - - - - - - - - - - - - - -

Après le formatage de l'écran, on entre dans le mode formaté. A partir de ce moment, il faut analyser tous les caractères introduits et verifier leur validité.

8.7.2.1. Les entrées.
-.-.-.-.-.-

Les données en entrée sont formées par des caractères différents des séquences de contrôle.

8.7.2.2. Les traitements.
-.-.-.-.-.-.-.-.-.

A la réception du caractère, on compare sa nature avec celle permise dans la zone où se trouve le curseur.

Si il y a correspondance, on mémorise le caractère dans la matrice ECRAN. Sinon, on attend le caractère suivant.

Si le caractère doit rester invisible, on met le huitième bit du byte à 1, ce qui le rend invisible à l'écran mais présent en mémoire.

On avance ensuite le curseur d'une position.

Deux cas peuvent se produire :

- le curseur reste dans la même zone, dans ce cas on ne fait rien.

- le curseur pénètre dans la zone suivante, si cette zone est protégée, on doit parcourir la file pour trouver la prochaine zone non-protégée et mettre à jour les caractèristiques de cette nouvelle zone.

8.7.2.3. Les sorties.
-.-.-.-.-. -

Les données de sortie sont formées par les caractères valides introduit dans le sous-module.

8.7.2.4. Algorithme
-.-.-.-.-.

Voir annexe III

8.7. 3. Remarques.
- - - - -

a) Certaines séquences de contrôle sont modifiées si nous nous trouvons dans le mode formaté.

Nous pouvons citer :

- la tabulation : ces fonctions travaillent sur les positions des zones non-protégées mémorisées dans TAB_PROTEGE.
- effacement : ces fonctions travaillent sur les zones non-protégées ou sur toute la page, suivant le mode.
- transmission : ces fonctions transmettent toutes les zones non-protégées ou toute la page, suivant le mode.

b) Pour réaliser une correction dans une zone, il suffit de mettre le curseur à l'endroit voulu, et de corriger le caractère érroné. Tous les contrôles sur le caractère sont à nouveau effectués.

8.8. Le convertisseur de code.

8.8.1. Remarques préliminaires sur les caractères et les séquences de contrôle.

a) L'émulateur et donc le convertisseur de code reçoit des caractères appartenant à la norme internationale ASCII N°5.

Notation.

- Les caractères sont repris dans une matrice (fig.8.8.1.) et peuvent être représentés de la manière suivante :

  X/Y avec X : la colonne de la matrice (entre 0 et 7)
  Y : la ligne (entre 0 et 15)

  Exemples :

  le caractère B = 4/2
  d = 6/4

- De plus, sur beaucoup de terminaux courants (compatibles télétypes), on obtient l'émission de caractères des colonnes 0 et 1 du tableau fig.8.8.1. en appuyant simultanément sur les touches CTRL et sur la touche correspondant au caractère de même rang dans les colonnes 4 et 5.

  Ainsi : SOH = CTRL A
  0/1 4/1

  DLE = CTRL P
  1/0 5/0

Classification.

Parmi tous les caractères définis, il existe un sous-ensemble de caractères dits imprimables comprenant les caractères - alphabétiques majuscules
minuscules

VERSION INTERNATIONALE

b7	b6	b5	b4		b3=0 b2=0 b1=0	b3=0 b2=0 b1=1	b3=0 b2=1 b1=0	b3=0 b2=1 b1=1	b3=1 b2=0 b1=0	b3=1 b2=0 b1=1	b3=1 b2=1 b1=0	b3=1 b2=1 b1=1
					0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	TC, (DLE)	SP	0	@	P	`	p
0	0	0	1	1	TC, (SOH)	DC,	!	1	A	Q	a	q
0	0	1	0	2	TC, (STX)	DC,	"	2	B	R	b	r
0	0	1	1	3	TC, (ETX)	DC,	#	3	C	S	c	s
0	1	0	0	4	TC, (EOT)	DC,	¤	4	D	T	d	t
0	1	0	1	5	TC, (ENQ)	TC, (NAK)	%	5	E	U	e	u
0	1	1	0	6	TC, (ACK)	TC, (SYN)	&	6	F	V	f	v
0	1	1	1	7	BEL	TC, (ETB)	'	7	G	W	g	w
1	0	0	0	8	FE, (BS)	CAN	(	8	H	X	h	x
1	0	0	1	9	FE, (HT)	EM	)	9	I	Y	i	y
1	0	1	0	10	FE, (LF)	SUB	*	:	J	Z	j	z
1	0	1	1	11	FE, (VT)	ESC	+	;	K	[	k	{
1	1	0	0	12	FE, (FF)	IS, (FS)	,	<	L	\	l	|
1	1	0	1	13	FE, (CR)	IS, (GS)	-	=	M	]	m	}
1	1	1	0	14	SO	IS, (RS)	.	>	N	^	n	~
1	1	1	1	15	SI	IS, (US)	/	?	O	_	o	DEL

CCITT. 5611

Fig : 8.8.1. : Code ASCII N°5.

- numériques
- de ponctuation
- auxiliaire (SP, # , $, %, ...)

par opposition aux autres caractères réservés aux fonctions de commande et appartenant justement essentiellement aux colonnes 0 et 1 du tableau fig 8.8.1. : les caractères de contrôle.

En plus, sur la plupart des terminaux, on a augmenté le nombre de commandes possibles par l'emploi de ce que nous appelons les séquences de caractères.

Pour exemple, voyons comment sont définies ces séquences suivant la norme américaine ANSI X3-64-1979 que nous avons choisie comme base de notre langage standard (voir chapitre II).

- La forme générale d'une séquence est la suivante :

CSI P..P I..I F

où * CSI est le caractère de début de séquence.

CSI : 9/11 : dans le cas d'une représentation en 8 bits

ESC : 1/11 + 5/11 : dans le cas d'une représentation en 7 bits, avec ASCII américain (5/11= [ )

* P...P est la chaîne de paramètres.
La longueur minimum de cette chaîne est 0 et la longueur maximum est définie par l'implémentation.

Cependant, les caractères admis sont compris entre 3/0 et 3/15.

* I...I sont des caractères intermédiaires utilisés pour étendre le répertoire des fonctions.

  Les caractères admis sont compris entre 2/0 et 2/15.

* F est le caractère définissant la fonction. Il peut être utilisé seul ou avec le(s) caractère(s) I pour établir quelle fonction est encodée.

  Les caractères admis sont compris entre 4/0 et 7/14.

- Les paramètres:

  Il existe deux types de paramètres :

  - les paramètres numériques utilisés pour passer des valeurs numériques.
    Dans certaines séquences, on a parfois besoin de plus d'un paramètre. Aussi, la chaîne de paramètres sera constituée de caractères compris entre 3/0 et 3/9 avec comme séparateur entre paramètre le caractère 3/11 (;).

  - les paramètres sélectifs pour sélectionner une entrée particulière dans une liste spécifiée.

b) L'analyse des codages de fonctions faite pour les terminaux de différents constructeurs (voir liste chapitre I) nous permet de dire :

- Toutes les fonctions des terminaux sont implémentées à l'aide :
  . d'une part, de caractères de contrôle, encore que, parfois, certains de ces derniers ne sont pas interprêtés comme la définition de la norme ASCII N°5 le spécifie,

. d'autre part, à l'aide des séquences de caractères et en ce qui concerne ces dernières, les constructeurs ont adoptés :

* des types de codages différents :
  - certains emploient le type de codage défini par la norme ANSI.
  - d'autres emploient un type de codage qui leur est propre.

* des types de codages identiques avec des différences quant aux caractères significatifs de la séquence d'une même fonction.

Exemples

Différence de type de codage pour une même fonction.

. ESC [Pn e
  - e : caractère significatif et terminal de la séquence
  - Pn : paramètre
  - [: Caractères de début de séquence

. ESC B Pn Ø
  - Ø : caractère terminal de séquence
  - Pn : paramètre
  - B : caractère significatif de la séquence
  - ESC : caractère de début de séquence

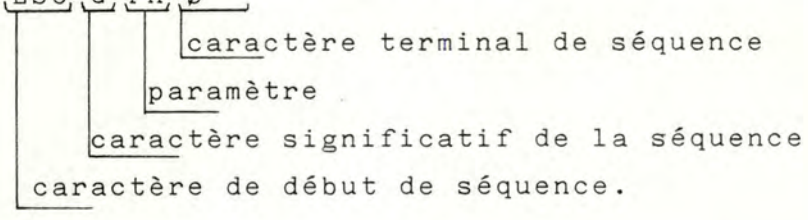
. ESC G Pn
  - il n'existe pas de caractère terminal de séquence
  - Pn : paramètre
  - G : caractère significatif de la séquence
  - ESC : caractère de début de séquence

Dans ce cas un peu particulier, il est évident que le nombre de paramètres est fixé et invariable pour la séquence.

Différence pour un même type de codage.

. ESC + Pn DEL

. ESC G Pn $



8.8.2. Le convertisseur de code : fonctionnement.

Dans notre solution, le convertisseur de code remplit deux rôles :

- Le premier consiste, dans le sens terminal-réseau, à reconnaître, filtrer et traduire éventuellement en langage standard les caractères de contrôle ou de séquence qui viennent du terminal et, dans le sens réseau-terminal, à reconnaître, traduire ou émuler les séquences standards issues de la station VTP.

- Le second consiste à être le moniteur de tous les autres modules en fonction de la séquence détectée et du type d'application dans laquelle on se trouve.

Pour remplir de manière efficace, les deux rôles qui lui sont donnés, nous avons doté le convertisseur de ce que nous avons appelé le vecteur de séquences, et les routines de traduction et d'émulation.

8.8.2.1. Principe général de fonctionnement.

Chaque caractère de la séquence, qui n'est pas paramètre, permet de progresser dans la structure que constitue le vecteur de séquences pour aboutir, lorsque le dernier caractère de la séquence est arrivé, à l'adresse d'une routine qui sera implémentée différemment suivant les cas :

. Dans le sens terminal-réseau.

La routine dite de TRADUCTION ;

+ consulte le vecteur de négociation
+ bloque la séquence ou envoie la séquence standard correspondante en fonction de ce que l'autre partenaire peut accepter.

Remarques :
Dans ce cas, le vecteur et les routines sont dépendantes du terminal :

- le vecteur reprend les séquences du terminal,
- les routines, à cause de l'exploitation des paramètres de ces mêmes séquences.

. Dans le sens réseau-terminal.

La routine dite D'EMULATION ;

+ gère éventuellement les modules de l'émulateur, et remplit ainsi le second rôle imparti au convertisseur
+ traduit ou, si nécessaire, émule la séquence standard reçue.

Remarques :

Dans ce cas, le vecteur est indépendant du terminal puisqu'il reçoit toujours des séquences standards.

Les routines sont en partie dépendantes du terminal, la partie "moniteur" étant identique quel que soit le terminal.

Il résulte de ce principe qu'il y a bien deux vecteurs de séquences par terminal. Ces derniers doivent être conçus de telle sorte :

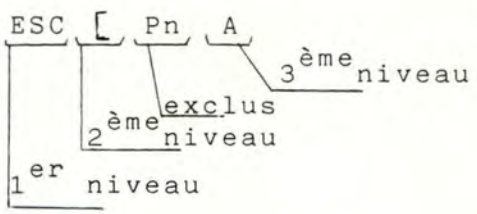
- qu'ils puissent être exploités par une et une seule routine de programme quel que soit le type de codage des fonctions (voir 8.1.1. remarque B)
- que le vecteur associé au sens terminal-réseau puisse être facilement modifié suivant les terminaux, et que cette modification n'interfère en aucun cas sur le reste du système.

8.8.2.2. Description du vecteur de séquences.

8.8.2.2.1. Description.

+ Le vecteur est divisé en niveaux, en fonction du nombre possible de caractères (non compris les paramètres) constituant une séquence de commande. (Séquence de caractères type ANSI ou d'autres types de caractères de contrôle dont la définition est particulière au constructeur).

Exemple :



+ Chaque niveau est divisé en zones : la zone est l'ensemble des caractères qui peuvent survenir étant donné que l'on a défini le caractère de séquence précédent.

  De par cette définition, le niveau 1 comprend une seule zone puisqu'il reprend l'ensemble des caractères de début de séquence de commande. (Les caractères de début de séquence peuvent définir la commande elle-même et être seuls : cas des caractères de contrôle).

  Les autres niveaux auront un ensemble de zones dépendant de l'implantation des séquences de commande propre au terminal.
  La fin de toute zone est marquée par un caractère unique "NUL".

+ Chaque zone est constituée d'éléments comprenant :

  . Le codage binaire du caractère de séquence.

  . Un pointeur sur le début de la zone qui contient le prochain caractère (non paramètre) de la séquence s'il existe.

  OU

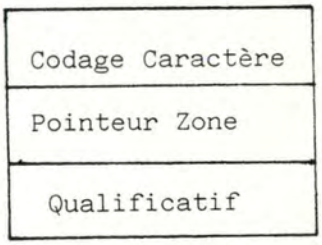
  Un index qui permettra de définir la routine à exécuter dans le cas où le caractère courant est le caractère de fin de séquence ou terminal de séquence.

  . Un qualificatif du type de caractère de séquence. En effet, un caractère de séquence peut être cerné par 4 qualificatifs :

- Le caractère peut être significatif, c'est-à-dire qu'il est celui qui définit la séquence.
- Le caractère peut être caractère de fin de séquence c'est-à-dire qu'il est le dernier caractère non paramètre de la séquence : les seuls caractères suivant celui-ci sont les paramètres de la séquence. Ces derniers sont obligatoires, en nombre fixe et invariable.
- Le caractère peut être paramètré, c'est-à-dire qu'il peut exister au moins un paramètre à la suite du caractère.
- Le caractère peut être terminal de séquence, c'est-à-dire que le caractère suivant n'appartient pas à la séquence.

Le caractère peut être une combinaison de ces quatre qualificatifs en vertu du tableau ci-après : (voir page 223)

+ Tous les éléments d'une zone ont ainsi la même taille.

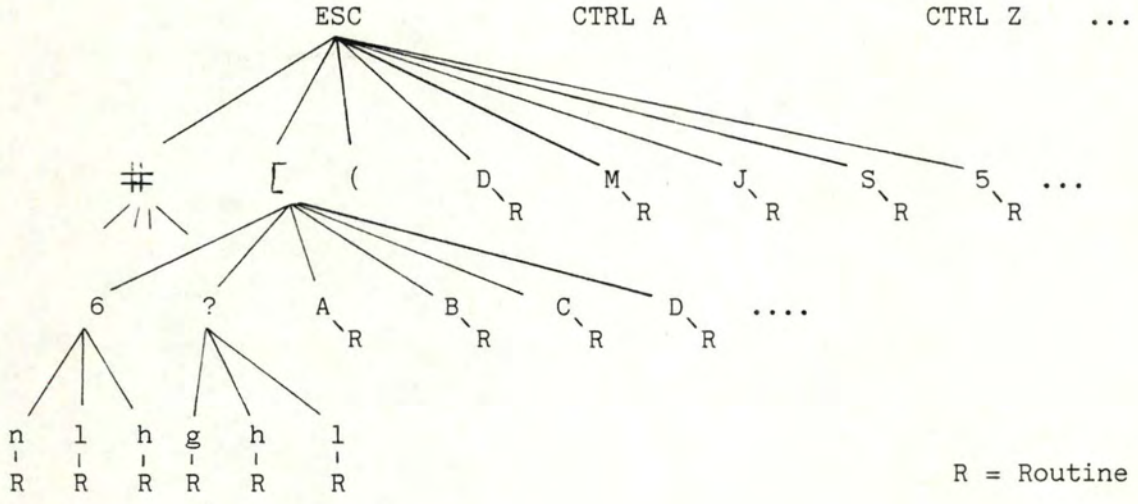


Qualif n°	Fin de Séquence	Terminal	Significatif	Paramètré	Exemples
0	0	0	0	0	ESC ....
1	0	0	0	1	ESC [Pn ..
2	0	0	1	0	
3	0	0	1	1	ESC G Pn ..
4	0	1	0	0	ESC G Pn $
5	0	1	0	1	
6	0	1	1	0	ESC [Pn A, CTRL A
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	ESC G Pn
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

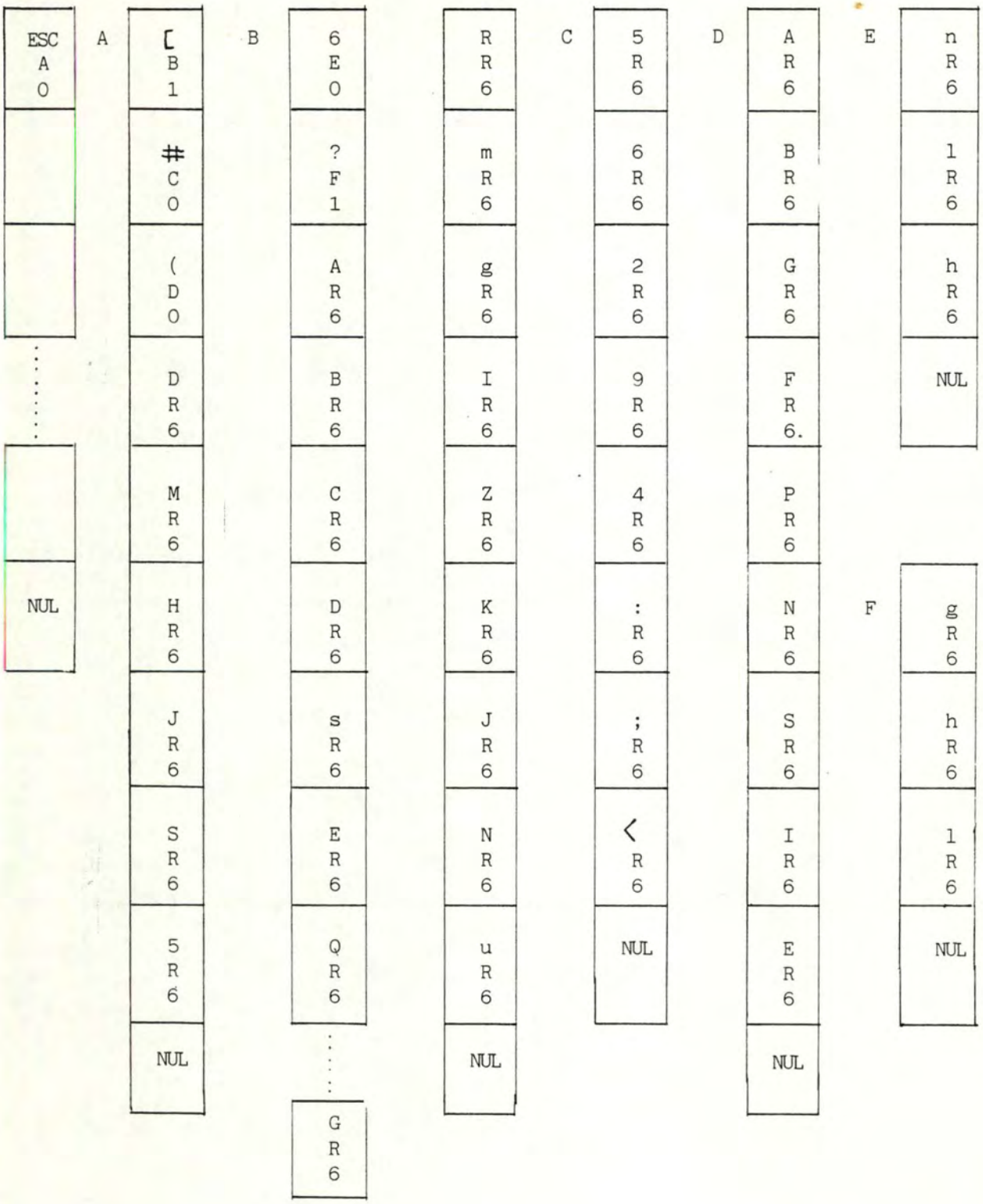
Tableau récapitulatif des qualificatifs pour les caractères de séquence.

8.8.2.2.2. Exemple : vecteur traitant de l'ANSI.
****

- Sous forme d'arborescence :



- Sous forme de vecteur : (voir page suivante)


ESC A 0
NUL
A
[B 1
C 0
(D 0
D R 6
M R 6
H R 6
J R 6
S R 6
5 R 6
NUL
B
6 E 0
? F 1
A R 6
B R 6
C R 6
D R 6
s R 6
E R 6
Q R 6
G R 6
C
R R 6
m R 6
g R 6
I R 6
Z R 6
K R 6
J R 6
N R 6
u R 6
NUL
D
5 R 6
6 R 6
2 R 6
9 R 6
4 R 6
: R 6
; R 6
< R 6
NUL
E
A R 6
B R 6
G R 6
F R 6.
P R 6
N R 6
S R 6
I R 6
E R 6
NUL
F
n R 6
l R 6
h R 6
NUL
g R 6
h R 6
l R 6
NUL

8.8.2.3. Mise en oeuvre du module convertisseur de code.

Le module convertisseur de code va être décrit en fonction des 3 parties qui le constituent :

- le sous-module qui se charge de l'exploitation des vecteurs de séquence
- les routines de traduction qui travaillent dans le sens terminal-réseau
- les routines d'émulation qui travaillent dans le sens réseau-terminal.

8.8.2.3.1. Exploitation des vecteurs de séquence.

C'est le sous-module "conversion" qui exploite le vecteur de séquence.

a. Les entrées.

Le sous module reçoit un caractère.

Celui-ci appartient à la grille ASCII N°5 et peut être :

1. un caractère n'appartenant pas à une séquence de caractère et n'étant pas un caractère de contrôle.
2. un caractère de contrôle dont la définition au niveau terminal est celle définie par ASCII N° 5 auquel cas il est considéré comme un caractère de type 1. (Il n'est pas repris dans le vecteur).

3. un caractère de contrôle dont la définition au niveau terminal n'est pas celle définie par ASCII N°5. Il est repris dans le vecteur en étant qualifié suivant le tableau page 223

4. un caractère appartenant à une séquence de caractères et qui n'est pas paramètre de cette dernière. Il est qualifié suivant le tableau page 223

5. un caractère appartenant à une séquence de caractères et qui est paramètre de cette dernière.

b. Le traitement.

Traitement_général.

Le traitement va devoir déterminer si le caractère courant appartient ou non à une séquence.

S'il n'appartient pas à une séquence, il doit être envoyé via une routine de traitement des caractères non de séquences.

S'il appartient à une séquence et s'il est repris dans les caractères du vecteur au niveau adéquat, un traitement est effectué en fonction du qualificatif de celui-ci (voir tableau P.223 et voir ci-dessous).

S'il appartient à une séquence et s'il n'est pas repris dans les caractères du vecteur au niveau adéquat : - il est paramètre et on l'envoie dans la pile des paramètres pour traitement ultérieur via la routine associée à la séquence définie,

- il n'est pas paramètre et on se trouve dans un cas d'erreur.

Traitement_à_effectuer_en fonction_des_qualificatifs_du_caractère_courant.

qualificatif = 0

. Progression à la zone adéquate du niveau suivant.

qualificatif = 1

. Indiquer que le caractère suivant pourrait être un paramètre de la séquence et initialiser la pile.

. Progression à la zone adéquate du niveau suivant.

qualificatif = 3

. Il est logique de penser que c'est le qualificatif "significatif" qui est important quant au choix de la routine à exécuter pour convertir ou émuler la séquence courante.
En effet, en tenant compte du cheminement effectué préalablement, ce caractère "significatif" n'est employé que par une et une seule séquence.

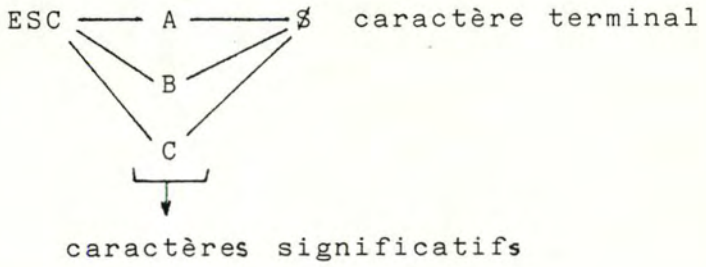
Exemple.

ESC [Pn A → significatif et donc unique par rapport aux autres séquences.

employé par plusieurs séquences

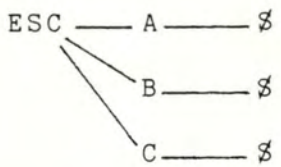
De plus, quand le caractère significatif est lui-même terminal, l'accès à la routine adéquate de traitement s'en trouve aisé.

En effet, si le caractère significatif n'est pas terminal, on peut représenter la progression interniveau comme suit pour 3 séquences différentes:

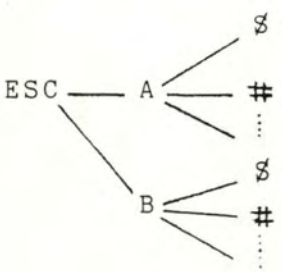


Cela suppose qu'au départ du caractère significatif on puisse trouver le caractère terminal (qui détermine la fin de la série éventuelle de paramètre) en plus d'obtenir l'index pour la routine d'émulation ou de traduction.
Traité comme tel, le cas est trop particulier. C'est pourquoi, on ramène le traitement du cas 3 au cas 6, on reporte implicitement le qualificatif significatif sur le caractère terminal.

Les séquences se représentent donc comme suit :



Cela impose un vecteur plus long mais simplifie grandement les traitements tout en permettant l'éventualité de cas plus généraux.



Le traitement du caractère qualifié 3.

- Indiquer que le caractère suivant pourrait être un paramètre de la séquence et initialiser la pile.
- Progression à la zone adéquate du niveau suivant.

qualificatif = 4

. Le caractère est terminal
. Indiquer que la séquence est terminée.
. On aboutit à la routine de traitement de la séquence aux vues de ce qui a été dit en 3.

qualificatif = 6.

. Idem 4

qualificatif = 10.

. Indiquer que la séquence est terminée.
. On aboutit àla routine de traitement qui doit se charger elle-même de l'acquisition des paramètres qui, rappelons le, sont dans ce cas en nombre fixe et invariable.

c. Les sorties.

Le sous module donne l'adresse de la routine à exécuter ou la détection qu'une erreur de séquence s'est produite et éventuellement les paramètres de la séquence.

8.8.2.3.2. Les routines de traduction.
****

Pour rappel :

- Ces routines sont accédées, via le vecteur de séquence, dans le sens terminal-réseau.
- Leur rôle est de transformer le code "terminal" des séquences acceptées par l'utilisateur en code "langage standard" et d'arrêter toute séquence non acceptée aux vues de la négociation.

Les routines de traduction, auront toutes la même structure. Celle -ci devra tenir compte du fait que l'on se trouve en mode conversationnel ou page. En effet, dans ce dernier cas :

- tout caractère envoyé par le terminal ne doit pas passer sur le réseau, mais doit susciter un écho.
- c'est à la seule pression de la touche d'émission que la page est envoyée entièrement ou partiellement en fonction des modes de transmission.
- c'est dans ce cas aussi qu'il est nécessaire de tenir compte des zones formatées.

La structure des routines de traduction à laquelle nous nous limitons est la suivante :

* Pour des fonctions :

```
si page alors écho
sinon filtre en fonction de la négociation
        si on peut envoyer alors
                envoyer la séquence standard
```

* Pour un caractère imprimable :

```
si page
    si zone formatée
        trt_zones_formatées (pour validation du
                             caractère introduit)
    sinon écho
sinon envoi du caractère sur réseau
```

8.8.2.3.3. Les routines d'émulation.

Pour rappel :

- Les routines sont accédées, via le vecteur de séquence dans le sens réseau-terminal.
- Outre la routine qui s'occupe du traitement d'un caractère imprimable ou d'un caractère de contrôle conrrectement interprété par le terminal, ces routines correspondent, chacune, à une fonction précise du terminal.
- Des paramètres représentés et structurés suivant ANSI (langage standard) doivent, éventuellement, être exploités.

Nous allons, par l'exemple de la gestion du curseur, montrer un genre de routines d'émulation.

Rappelons que seule la partie "émulation" est propre au terminal.

Exemple :

Considérons les déplacements classiques des curseurs :

haut	: qui correspond à la séquence ANSI ESC[Pn A
bas	: . . . . . ESC[Pn B
gauche	: . . . . . ESC[Pn D
droite	: . . . . . ESC[Pn C

Dans le cas des trois premiers types de déplacements, le curseur s'arrête aux limites, même si le paramètre en exige le dépassement.

Dans le cas du déplacement à droite, le déplacement, ou non, des limites dépend du mode autowrap.

a. Curseur haut et gauche.

Entrée_:

1 paramètre éventuel (Pn)

Traitement_:

```
Si "gestion_écran" de "status_émul" alors
     mise à jour _ curseur (X(pour gauche) ou Y(pour haut))
     N_écran
sinon
     mise à jour _curseur (depcur X(gauche) ou
                                   depcur Y (haut))
```

Emulation : envoi de la séquence de positionnement absolu du curseur avec les paramètres depcur X et depcur Y. La séquence est dans le langage compris par le terminal.
Si le terminal ne possède pas cette séquence, elle peut être émulée par combinaison d'autres séquences.

```
Mise à jour _curseur (pos_curs)
     si pile_param = vide alors Pn = 1
          sinon Pn = valeur dans la pile
     si pos_curs - Pn ⩾ 0  alors pos_curs = pos_curs-Pn
     sinon pos_curs = ∅
```

b. Curseur bas et droite.

Entrée:

1 paramètre éventuel (Pn)

Traitement :

```
si"gestion-écran" de "status-émul" alors
    mise à jour_curseur-2  (X (d)ouY (b)),(maxcar ou
                                                maxlig)
    N_écran
sinon
    mise à jour _curseur-2  (depcurX(d)ou depcurY(b)),
                            (maxcarT ou maxligT)
Emulation (cfr. exemple précédent)
```

```
mise à jour_curseur-2 (pos_curs, limite)
    si pile _ param = vide alors Pn = 1
                           sinon Pn = valeur dans la pile
    si pos_curs + Pn ≤ limite alors
        pos_curs = pos_curs + Pn
    sinon
        * cas "curs bas" ou pas autowrap dans le
          cas "curs droite"
              pos_curs = limite
        * cas autowrap avec curs droite
              pos_curs = (pos_curs + Pn) - limite
              pile_param = vide
              mise à jour_curseur-2 (depcurY,maxligT)
```

C O N C L U S I O N S.

C O N C L U S I O N S

En commençant ce travail, nous voulions montrer dans quelle mesure il était possible de rendre compatible deux terminaux aux caractéristiques différentes, ou d'employer sur un terminal de type A, une application initialement faite pour un terminal de type B et, ainsi, ramener la notion de protocole d'appareil virtuel, souvent traitée de manière théorique et trop abstraite, à un niveau plus concret.

En prenant cette orientation un peu originale, nous étions conscients d'une part, que l'étendue du sujet nous imposerait certaines limitations : ainsi,

- certains sujets comme les protocoles d'intercouches, les échanges libres, la gestion des tampons mémoires de l'adaptateur, le parallélisme des entrées/sorties n'ont pas été développés,
- l'étude s'est limitée à la compatibilité des terminaux classiques, couramment employés, étant exclus les terminaux graphiques et tous les périphériques des terminaux,
- certaines fonctions exclusives à certains terminaux n'ont pas été retenues, comme les attributs du curseur, les divers niveaux de brillance, la gestion de fenêtre dans un écran, etc...
- nous nous sommes limités aux applications les plus courantes (traitement de texte, questions-réponses, zones formatées, communications entre terminaux),

d'autre part, qu'en plus des limitations, nous serions amenés à poser des choix dont certains suscitent des réserves : ainsi,

- le choix de l'ANSI comme langage standard : ce choix est subjectif et devra être normalisé dans le futur,
- le choix de l'implantation de l'interface d'adaptation, comprenant l'émulateur et la station VTP, à l'extérieur du terminal,
- le choix de la représentation des paramètres associés aux primitives ; bien qu'on puisse intégrer ceux-ci dans les groupes de paramètres définis par la norme, ils n'en restent pas moins, eux aussi, purement subjectifs,
- le choix d'une émulation orientée application (notons que cette notion a été introduite par nous),
- le choix d'un schéma de négociation symétrique.

Toutes ces restrictions et tous ces choix étant établis, il est intéressant de citer ce que permet la solution telle qu'elle est décrite dans la seconde partie du travail.

Notre solution permet :

- d'éviter toute modification du terminal,
- une compatibilité des codes de gestion des terminaux (caractères et séquences de contrôle) grâce au langage standard,
- une compatibilité des formats d'écran grâce au système de "fenêtre",
- la possibilité pour un terminal d'accepter des applications qui n'ont pas été faites pour lui à l'origine,
- une meilleur lisibilité des textes formatés ou non, grâce à un reformateur (module traitement de texte) et un convertisseur de code ASCII pour les jeux de caractères de nationalité différente,

- une extensibilité toujours possible due à l'emploi du code ANSI et d'une structure software modulaire,
- de ne modifier, pour un nouveau terminal, que les vecteurs de négociation et une partie du module convertisseur de code,
- un traitement local de l'information.

Conclusions :

Aux vues de ce qui vient d'être dit, nous pouvons conclure que si notre travail répond en grande partie à l'objectif fixé, il n'en reste pas moins qu'il est encore au stade d'étude théorique avancée et ce pour différentes raisons parmi lesquelles nous citerons :

- l'ampleur du travail malgré les limites posées et citées ci-dessus,
- la difficulté de recueillir une documentation abondante, précise , variée au niveau de la gamme des terminaux, et représentative du marché,
- le faible degré d'approfondissement du support bibliographique qu'il nous a été donné de consulter sur le protocole d'appareil virtuel,
- la nécessité de poser de nombreux choix tout au long du travail en ce qui concerne des fonctions essentielles (langage standard, modules...).

B I B L I O G R A P H I E.

B I B L I O G R A P H I E

* "Téléinformatique" MACEBI - GUILBERT

* "The role and nature of a virtual terminal"
  D.L.A. BARBER - EIN/DLAB/VTP,INWG PROTOCOL 63.

* "A virtual terminal protocol for a Belgian University Network" S.A.R.T. 77/10/13,
  BAUWENS E. , MAGNEE F.

* "Projet de protocole terminal virtuel" (proposition I)
  BAUWENS E. , MAGNEE F.

* "Remarks on negociation mechanisms and attention handling"
  BAUWENS E. , MAGNEE F.

* "Projet d'avis S.61 : Répertoire de caractères et de jeux de caractères codés pour le service international télétex".

* "American National Standard Additional Controls for use with American National Standard Code for Information Interchange - ANSI X3.64 - 1979.

* Documentation sur les terminaux - I.B.M.
  - Honeywell-Bull
  - Télévidéo.

Année académique 1984 - 1985.

A N N E X E S.

Promoteur BRUNIN J.

ROTENS Etienne

TOURNAY Patrick

A N N E X E I.

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DT15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325
Cursor position report																						
1) host terminal		X	X	X	X	X	X	X		X	X					X	X					
2) terminal host		X	X	X	X	X	X	X		X	X					X	X	X				
Cursor up	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
Cursor down	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
Cursor forward	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
Cursor backward	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
Cursor next line		X						X		X	X											
Cursor previous line								X														
Index line		X						X		X	X									X	X	
Reverse index		X						X		X	X											
Relative horizontal movement								X														
Relative line positioning								X														
Absolute horizontal cursor adressing								X				X	X									
Absolute vertical positioning								X				X	X							X	X	X
Limited cursor addressing	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X				
Cursor hom e	X	X	X	X	X	X	X	X		X	X	X	X		X	X	X	X				
Line feed	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X			X	X	X
Reverse line feed	X			X	X	X	X							X								
Save cursor		X						X		X	X											
Restore cursor		X						X		X	X											
Next line		X								X	X											

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DM15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325
Line / Character attribute																						
1) single-with line 1H + 1L		X						X		X	X									X	X	X
2) 1 X H + 2 X L		X						X		X	X									X	X	X
3) 2 X H + 2 X L								X												X	X	X
4) Top half of double hight line		X						X		X	X											
5) Lowe half of double hight line		X						X		X	X											
6) Double height single with top										X												
7) botton										X												
Display visual attributes																						
1) Normal intensity		X		X	X	X	X	X		X	X	X	X	X	X	X						
2) half intensity												X	X			X						
3) Increase intensity		X						X		X	X											
4) Decrease intensity								X														
5) blonk				X	X	X	X	X				X	X	X		X						
6) Underline		X		X	X	X	X	X		X	X	X	X	X		X						
7) Blink		X		X	X	X	X	X		X	X	X	X	X		X						
8) Reverse video		X		X	X	X	X	X		X	X	X	X	X	X	X						
Contrast emphasis								X														
Screen background		X		X	X	X	X	X		X				X								
Screen intensity								X														
Line attribute mode								X														
Clear area attributes																						
1) position cursor								X														
2) cursor line								X														
3) all								X														
Set top and bottom margins		X								X	X									X	X	X

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DT15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325
Erose in page																						
1) Cursor fin	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X				
2) Début cursor		X						X		X	X											
3) Tout		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
Erose in line																						
1) cursor fin	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X				
2) début cursor		X						X		X	X											
3) tout								X		X	X											
Erose in field																						
1) cursor fin								X					X									
2) début cursor								X														
3) tout								X	X				X	X	X		X					

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DT15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325	
TAB																							
1) Next tab stop	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	
2) Next unprotected field			X	X	X	X	X	X	X				X	X	X		X						
TAB CLEAR																							
1) Cursor position		X	X	X	X	X	X	X		X	X			X									
2) tout		X	X	X	X	X	X	X		X	X			X	X					X	X		
3) tout + défaut valeur								X															
4) tab stop dans les 8 positions										X	\												
Set tab at current column		X	X	X	X	X	X	X		X	X			X	X					X	X	X	
Back tab																							
1) previous tab stop			X	X	X	X	X	X	X				X	X			X						
2) previous unprotected field			X	X	X	X	X	X	X				X	X			X						

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DT15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325
Modes																						
1) LF/NL		X								X	X										X	
3) Autowrap on		X						X		X	X											
4) 80/132		X						X		X	X											
5) Origin relatif/absolut		X						X		X	X											
8) Autopage On/off					X	X	X							X								
9) Monitor on/off			X	X	X	X	X					X	X	X		X	X	X				
10) Protect mode			X	X	X	X	X	X					X	X								
11) Write protct mode			X	X	X	X	X	X					X	X								
12) Conversation mode			X	X	X	X	X						X	X								
13) Block mode			X	X	X	X	X							X								
16) Set/reset mode		X									X											
17) clear display / LF										X												
18) Screen with change										X												
20) Right half of double with line erase										X												
21) Erase page extend mode										X												
22) Tab motion spaces/direct modes										X												
23) Transparent mode																						
24) Page mode													X	X								
25) Message mode													X									
28) X ON X OFF														X								
29) Format normal mode																						

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DT15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325	
SEND																							
1) ligne non protégée			X	X	X	X	X	X	X					X	X								
2) ligne protégée				X	X			X						X	X		X						
3) all the line			X	X	X	X	X	X	X						X								
4) unprotected page			X	X	X	X	X	X	X					X	X		X						
5) protected page				X	X			X							X								
6) all page			X	X	X	X	X	X	X				X	X	X		X						
7) unprotected message			X	X	X	X	X	X						X									
8) protected message				X	X			X															
9) all message			X	X	X	X	X	X					X	X									
10) un caractère	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
SPECIAL SEND																							
1) Terminal ID	X	X		X	X		X	X						X	X								
2) User·line				X	X		X	X						X	X								
3) Status line		X			X		X	X				X	X	X	X								
4) Request device attrib		X						X		X	X				X								
PROGGRAM SEND KEY																							

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DT15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325
Select characters set																						
1) UK		X						X		X	X									X	X	X
2) USA	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
3) german								X			X									X	X	X
4) french								X			X									X	X	X
5) spanish								X			X											X
6) portuguese								X			X											
7) norvegien								X			X									X	X	X
8) swedish								X			X									X	X	X
9) set line drowing	X	X								X	X											
10) set up defined											X											
11) graphique											X											X
12) alternate charact.		X								X										X		

LIBELLE	VT 52	VT 100	TV 910	TV 914	TV 924	TV 925	TV 950	TV 970	VISA 40	CIT 101	WS 584	REG. 100	REG. 200	VISU DT15	DKU 7005	TEC 500	TEC 510	BEE 1-2	BEE 100	LA 34	LA 120	FAC. 4325
Carriage Return	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Line Feed	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Bell		X	X	X	X	X	X	X				X	X							X	X	X

COMMENT PROTEGER UNE ZONE.

1) TV970

I) Form Build Mode (début zone)

II) Define Area Attribute

a) all input
b) no input
c) numeric
d) alphabetique
e) no input but transmission
f) data entry required
g) total fill
i) digits

III) Proctect Form Mode (fin de zone)

2) Regent 200

I) Form generation mode (ESC R)

II) Define area attribut

- non protégé/protégé
- attributs de visualisation
- print only
- constant (envoie même les zones protégées)

III) sortie zone

3) VISA 40

I) Set background

II) Data protégé ½ intensité 1 zone

III) Clear foreground

4) TV950 910 + VISUDT15

I) Write protect mode on

II) texte (½ intensité)

III) Write protect mode off

A N N E X E II.

Définition des fonctions standards.

Nous trouvons dans cette annexe la liste complète des fonctions adaptées comme standard.
Ce sont donc les seules qu'on pourra émettre sur le réseau et les seules que les émulateurs seront capables de décoder.

Quelques définitions :

Pn : est un paramètre que l'on retrouve dans beaucoup de fonctions de curseur.
Il est exprimé en ASCII c'est-à-dire que 108 est codé comme 3 caractères ASCII et non en entier.

Ligne courant : ligne où se trouve le curseur.

Colonne : colonne où se trouve le curseur.

N.B. : On travaille toujours sur un seul écran qui est équivalent à une page.

Les bords de l'écran correspondent au bord de la mémoire écran et pas nécessairement au bord du vidéo.

A) LE CURSEUR.

1) Curseur up (↑)

- code : ESC [Pn A
- Valeur de Pn par défaut : 1
- Définition : déplace la position du curseur vers le haut de Pn positions mais reste dans la colonne courante.
- Cas particulier : si le curseur atteint le haut de l'écran alors il s'arrête même si la valeur de Pn n'est pas satisfaite.

2) Curseur down (↓)

- Code : ESC [Pn B
- Valeur de Pn par défaut : 1
- Définition : déplace la position du curseur vers le bas de Pn positions mais reste dans la colonne courante.
- Cas particulier : si le curseur atteint le bas de l'écran alors il s'arrête même si la valeur de Pn n'est pas satisfaite.

3) Curseur Forward (→)

- Code : ESC [Pn C
- Valeur de Pn par défaut : 1
- Définition : déplace la position du curseur vers la droite de Pn positions dans la ligne courante.
- Cas particuliers :

  a) en mode normal : si le curseur atteint la droite de l'écran alors il s'arrête même si la valeur de Pn n'est pas satisfaite.

  b) en mode autowrap : si le curseur atteint la droite de l'écran alors il passe à la première position de de la ligne suivante et se déplace jusqu'à satisfaire Pn.

4) Curseur backward (←)

- Code : ESC [Pn D
- Valeur de Pn par défaut : 1
- Définition : déplace la position du curseur vers la gauche de Pn positions dans la ligne courante.
- Cas particuliers :

  a) en mode normal : si le curseur atteint la gauche de l'écran alors il s'arrête même si la valeur de Pn n'est pas satisfaite.

  b) en mode autowrap : idem mode normal.

5) Curseur Next line

- Code : ESC [Pn E
- Valeur de Pn par défaut : 1
- Définition : déplace la position du curseur à la première colonne de Pn lignes plus bas que la ligne courante.
- Cas particulier : si le curseur atteint le fond de l'écran alors il y a scroll up jusqu'à ce que Pn soit satisfait.

6) Curseur Previous line.

- Code : ESC [Pn F
- Valeur de Pn par défaut : 1
- Définition : déplace la position du curseur à la première colonne de Pn lignes plus haut que la ligne courante.
- Cas particulier : si le curseur atteint le haut de l'écran alors il y stoppe même si Pn n'est pas satisfait.

7) Index

- Code : ESC D
- Définition : déplace la position du curseur une ligne plus bas dans la colonne courante.
- Cas particulier : si le curseur est sur la dernière ligne de l'écran alors il y a scrolling up et le curseur reste dans la colonne courante.

8) Reverse index

- Code : ESC M
- Définition : déplace la position du curseur d'une ligne vers le haut dans la colonne courante.
- Cas particuliers : si le curseur est sur la première ligne de l'écran alors il y a scrolling down et le curseur reste dans la colonne courante.

9) Relative column movement

- Code : ESC [Pn a
- Valeur de Pn par défaut : 1
- Valeur limite de Pn : $0 \leq Pn \leq$ max. caractères
- Définition : déplace le curseur vers la droite d'un nombre donné de colonne relative à la position du curseur sans quitter la ligne présente.
- Cas particuliers : + une valeur trop grande déplace le curseur à l'extrémité droite de la ligne courante.
  + une valeur négative est ignoré.

10) Relative line positioning

- Code : ESC [Pn e
- Valeur de Pn par défaut : 1

- Définition : déplace le curseur vers le bas d'un nombre donné de ligne relative à la position du curseur sans changer de colonne.
- Cas particulier : une valeur trop grande déplace le curseur à la dernière ligne de l'écran.

11) Curseur addressing

- Code : ESC [Pe ; Pc s
- Valeurs de Pe et Pc par défaut : 1
- Définition : positionne le curseur à la Pe^{eme} ligne et à la Pc^{eme} colonne de l'écran.
- Cas particulier : la position home est obtenue en prenant Pc et Pe par défaut.

12) Absolute column movement

- Code : ESC [Pn G
- Valeurs de Pn par défaut : 1
- Définition : déplace le curseur à la colonne spécifiée par Pn dans la ligne où se trouve le curseur.
- Cas particulier : une valeur trop grande de Pn (> max caractères) déplace le curseur à la dernière colonne de la ligne courante.

13) Absolute Vertical Positioning

- Code : ESC [Pn d
- Valeurs de Pn par défaut : 1
- Définition : déplace le curseur à la ligne spécifiée par Pn sans changer de colonne.
- Cas particulier : une valeur trop grande de Pn (> Maxligne) déplace le curseur sur la dernière ligne dans la colonne courante.

14) Curseur Position Report

a) demande de l'ordinateur :

- Code : ESC [6 n
- Définition : l'ordinateur demande au terminal où se trouve le curseur.

b) réponse du terminal

- Code : ESC [Pe ; Pc R
- Définition : le terminal, après la réception d'une demande de rapport, envoie automatiquement la position du curseur.

B) LES ATTRIBUTS DE VISUALISATION.

1) La taille des caractères.

- Code :
  - +double largeur et double hauteur — ESC #2
  - +double largeur et double hauteur avec ligne supérieure — ESC #3
  - +double largeur et double hauteur avec ligne inférieure — ESC #4
  - +simple largeur et simple hauteur — ESC #5
  - +double largeur et simple hauteur — ESC #6
  - +double hauteur et simple largeur — ESC #7
  - +simple largeur et double hauteur avec ligne supérieure — ESC #8
  - +simple largeur et double hauteur avec ligne inférieure — ESC #9

- Définition : change la hauteur et la largeur des caractères dans toute une ligne où se trouve le curseur.
- Cas particuliers :
  - pour la double hauteur, certains terminaux effectuent cela en 2 lignes. Il faut donc se positionner correctement pour les parties hautes et basses du caractère.
  - aucun attribut n'est utilisable en mode protégé.
  - le nombre de caractères par ligne est divisé par 2 en double largeur.
  - Le nombre de lignes par page est aussi divisé par 2 en double hauteur.

2) les attributs d'un caractère

- Code : ESC [Ps;...;Ps m
- Valeurs de Ps :

normal	0
+ gras, + foncé	1
- gras, + clair	2
invisible	3
souligné	4
clignotant	5
inverse	7
italique	9

- Définition : les attributs visuels changent l'apparence de la donnée sur l'écran.
- Cas particuliers :
  - les attributs sont additifs.
  - quand on change les attributs, on utilise Ps = 0 à moins qu'on ne veuille combiner les attributs précédents aux nouveaux.
  - attribut valable jusqu'à nouvelle redéfinition.
  - il n'occupe pas une place mémoire.

C) LES TABULATIONS.

1) Set horizontal tabstop.

- Code : ESC H
- Définition : met un tabstop à la position courante du curseur.
- Cas particulier : cette commande est ignorée en mode protégé.

2) Set vertical tabstop.

- Code : ESC J
- Définition : met un tabstop à la ligne courante.

3) Clear tabstop horizontal.

- Code : ESC [Ps g
- Valeurs de Ps :

  0 : efface le tabstop à la position du curseur

  3 : efface tous les tabstops

  ?0 : efface tous les tabstops et remet ceux par défaut.

- Définition : efface des tabstops suivant la valeur de Ps
- N.B. : cette commande est inactive en mode protégé.

4) Clear vertical tabstop.

- Code : ESC [Ps g
- Valeurs de Ps

  1 : efface le tabstop de la ligne courante.

  4 : efface tous les tabstops.

5) Horizontal tab

- Code : ESC [Pn I
- Valeur par défaut de Pn : 1
- Définition : déplace le curseur Pn tabstops plus vers la droite.
- Cas particuliers :
  - a - en mode protégé : déplace le curseur de Pn zones non protégées vers l'avant. Si le nombre Pn est plus grand que la dernière zone alors on s'arrête sur elle.
  - b - en mode autowrap : si le Pn est plus grand que le dernier tabstop, on passe à la ligne suivante pour satisfaire Pn.
  - c - en mode normal : si Pn est plus grand que le dernier tabstop, on s'arrête au dernier.

6) Horizontal backtab.

- Code : ESC [Pn Z
- Valeur par défaut de Pn : 1
- Définition : déplace le curseur de Pn tabstop plus vers la gauche.
- Cas particuliers :
  - a - en mode protégé ; n'a pas d'action.
  - b - en mode autowrap : n'a pas d'action.
  - c - en mode normal : si Pn fait revenir avant le premier tabstop, on s'arrête à la première position de la ligne.

7) Vertical tab

- Code : VT
- Définition : déplace le curseur jusqu'au tabstop ligne suivant .

8) Marge gauche et droite.

- Code : ESC [n_1 ; n_2 H
- Définition : permet de définir une marge à gauche et à droite du texte imprimé et cela jusqu'à redéfinition d'une nouvelle marge à gauche et à droite.
- Cas particuliers :
  - uniquement utilisable sur des imprimantes ou des télétypes.
  - n_1 et n_2 correspondent au numéro de la colonne par rapport au début de la ligne qui correspond à la colonne 0.
  - n_1 et n_2 doivent être plus petits que la largeur maximum du formulaire.

9) Marge en haut et en bas.

- Code : ESC [n_1 ; n_2 r
- définition : permet de définir une marge en haut et en bas de la page imprimée, c'est-à-dire le nombre de ligne que l'on passe au début et en fin de page. Ceci se reproduit sur toutes les pages jusqu'à une nouvelle redéfinition.
- Cas particuliers :
  - uniquement utilisable sur des imprimantes ou des télétypes.
  - n_1 et n_2 correspondent au numéro de la ligne par rapport au début de la page.
  - n_1 et n_2 doivent être plus petits que la longueur maximum du formulaire.

10) Indice.

- Code : ESC K
- Définition : permet d'imprimer le caractère qui suit en indice c'est-à-dire une demi-ligne plus bas.

11) Exposant.

- Code : ESC L
- Définition : permet d'imprimer le caractère qui suit en exposant c'est-à-dire une demi-ligne plus haut.

D) L'EFFACEMENT.

1) Effacement dans une ligne.

- Code : ESC[Ps K
- Valeur de Ps par défaut : 0
- Valeurs de Ps :

  0 : efface de et incluant le curseur jusqu'à la fin de la ligne courante.

  1 : efface du début de la ligne courante jusqu'à et incluant la position du curseur.

  2 : efface toute la ligne courante et déplace le curseur à la première position de cette même ligne.

- N.B. : la commande efface les caractères et les attributs de la ligne.

2) Effacement dans une page.

- Code : ESC [Ps J
- Valeur par défaut de Ps : 0
- Valeurs de Ps. :

  0 : efface de et incluant le curseur jusqu'à la fin de la page.

  1 : efface le début de la page jusqu'à la position du curseur incluse.

  2 : efface toute la page, change tous les attributs des lignes et déplace le curseur au début de la page (home).

N.B. : - en mode protégé ; cette commande efface toutes les zones non-protégées de la page et déplace le curseur jusqu'au début du premier champ non protégé.

- aprés effacement, les attributs sont simple hauteur et largeur. Ceci ne concerne pas la ligne du curseur dans le cas ou Ps = 0 ou 1.

3) effacement dans une zone. .

- Code : ESC [Ps N .
- Valeur par défaut de Ps : 0
- Valeurs de Ps :

  0 : efface du curseur jusqu'au tabstop suivant non-inclus ou jusqu'à la fin de la zone en mode protégé.

  1 : efface du tabstop ou de début de zone jusqu'au curseur inclus.

  2 : efface la totalité de la zone tabstop et replace le curseur au tabstop courant ou au début de la ligne.

- N.B. : - en mode protégé : zone tabstop est égale à la zone entre tabstop précédent ou le début de la ligne et le tabstop suivant ou la fin de la ligne.
  - en mode protégé : la zone tabstop est une zone non-protégée.

E) LES JEUX DE CARACTERES.

- Codes : ESC (Pn ou ESC) Pn

- Définition : - permet de choisir parmi la liste celui des alphabets que l'on désire utiliser.

  - reste valable tant qu'on ne réintroduit pas une nouvelle commande ESC (Pn ou ESC) Pn.

- Valeurs de Pn :

1. A English
2. B USA (ASCII)
3. E Spanish
4. F French
5. G German
6. I Italian
7. N Norvegian / Danish
8. P Portuguese
9. S Swedish
10. H Finish

F) LES TOUCHES DE FONCTIONS PROGRAMMABLES.

1) Programmation.

- Code : ESC P X_1, X_2, X_3..., X_n,
- Définition : permet de programmer les touches de fonctions du terminal.
  La valeur X_i correspond à une suite de caractères éventuellement vides qui seront envoyés à chaque pression de la touche PFi.
- Caractéristiques :
  a) la zone Xi peut contenir de 0 à 10 caractères.
  b) tous les caractères de contrôle sont permis.
  c) chaque zone X_i est séparée de la suivante par une virgule ainsi que la dernière zone X_i.
  d) aucun X_i ne peut être oublié.
  e) le caractère *, s'il se trouve en première position dans X_i et qu'il est seul dans X_i est considéré comme la programmation d'une séquence standard correspondant à cette touche (voir 2).

2) Touche PFi

- Code : ESC O Pn
- Valeurs de Pn

@	touche PF1
A	PF2
B	PF3
C	PFA

- Définition : si on a programmé la séquence standard alors on envoie lorsqu'on enfonce une touche PFi le code correspondant ESC O i

G) LES MODES.

1) Autowrap mode.

- Codes : ESC [? 7 l OFF
  ESC [? 7 h ON
- Définition :
  A) ON
  - en mode normal : en fin de ligne, il y a un saut automatique à la ligne suivante à la première position.
  - en mode protégé : en fin de ligne, il y a un saut automatique au champ non-protégé de la ligne suivante.

  B) OFF
  - en mode normal : le dernier caractère introduit est surimprimé lorsqu'on a atteint le bout droit de l'écran dans la ligne courante.
  - en mode protégé : le dernier caractère introduit est surimprimé lorsqu'on atteint la dernière position non protégée de la ligne courante.

2) Line Feed / New Line mode.

- Codes : ESC [20 l LF
  ESC [20 h NL
- Définition :
  A) LF : si on appuie sur Return alors on envoie seulement un CR. Si on reçoit un LF alors seulement on passe à la ligne -(LF).
  B) NL : si on appuie sur Return alors on envoie un CR+LF. Si on reçoit un LF, on accomplit un CR+LF.

3) Erasure mode.

- Code : ESC [6 1 1)
  ESC [6 h 2)
- Définition : uniquement en mode protégé.
  1) efface toutes les données de la page - les attributs visuels, sort du mode protégé.
  2) efface toutes les données dans les zones non protégées.

4) Form Build mode.

a) Code : ESC [? 17h

Définition : permet d'entrer dans le mode pour formater l'écran.

b) Code : ESC [? I71

Définition : permet de sortir du mode qui sert à formater l'écran.

5) Protect forms mode.

a) Code : ESC [? Oh

Définition : une fois l'écran formaté, si on désire travailler dans cet écran formaté alors on active ce mode.

b) Code : ESC [? Ol

Définition : on sort de ce mode d'écran formaté et protégé.

6) Autotab mode.

a) - Code : ESC [? 151
   - Définition : si on tente d'entrer dans une zone protégée, alors le caractère ou le curseur va sauter jusqu'à la première position de la zone non-protégée qui suit.

b) - Code : ESC [? 15h

- Définition : si on tente d'entrer dans une zone protégée alors le caractère où le curseur reste dans la zone non protégée précédente si c'est un caractère, il y a surimpression.

7) Block/Conversation mode.

a) Block

- Code : ESC [? 10 h
- Définition : les caractères introduits au clavier ne sont pas envoyés à l'ordinateur tant qu'il n'y a pas une demande du clavier ou de l'ordinateur.
- N.B. : les données tapées au clavier sont envoyées au vidéo.

B) Conversation

- Code : ESC [? 101
- Définition : les caractères sont envoyés à l'ordinateur aussitôt qu' ils sont tapés au clavier.
- N.B. : les données n'apparaissent à l'écran que si l'ordinateur les renvoie .

H) LES TRANSMISSIONS.

N.B. : Il faut toujours se trouver en block mode pour utiliser les modes et les commandes qui suivent. Celles-ci vont déterminer quelle partie de l'écran on envoie à l'ordinateur.

1) Transfer Execution.

a) Immediate transmission.

- Code : ESC [? 14 h
- Définition : si on enfonce la touche send, on envoie directement la partie du texte sélectionnée par les autres commandes.

b) Defer transmission.

- Code : ESC [? 14 l
- Definition : si on enfonce la touche send, on envoie la séquence disant à l'ordinateur qu'on est prêt à émettre (ESC S) et l'ordinateur invite à émettre en envoyant la séquence ESC 5.

c) Set transmit state séquence.

- Code : ESC S
- Définition : le terminal dit à l'ordi teur qu'il est prêt à transmettre.

d) Transmit séquence.

- Code : ESC 5
- Définition : l'ordinateur fait une demande de transmission d'un bloc de données

2) Transfer termination.

a) Transfer partiel page.

- Code : ESC [16 l
- Définition : transfère la partie de la page comprise entre la position "home" et la position courante du curseur incluse.

3) Page transfer.

a) Transfer page

- Code : ESC [? 13 l
- Définition : transfère la totalité de la page.

b) Transfer display

- Code : ESC [? 13 h
- Définition : transfère un écran.

4) Line transfer.

a) Transfer display

- Code : ESC [? 11 l
- Définition : cette commande permet la commande H-2 qui spécifie si on transmet toute la page ou la partie avant le curseur.

b) Transfer line.

- Code : ESC [? 11 h
- Définition : cause la transmission de données du début de la ligne courante jusqu'à et incluse la position du curseur, sans envoyer les codes de contrôle de fin de ligne.

5) Multiple Area Transfer.

a) Transfer one field.

- Code : ESC [15 l
- Définition : en mode protégé, transfère la zone non-protégée contenant le curseur.

b) Transfer all fields.

- Code : ESC [15 h
- Définition : en mode protégé, transfère toutes les zones non-protégées de l'écran.

6) Guarded Area Transfer.

a) Transfer only unguarded areas.

- Code : ESC [1 l
- Définition : transfère toutes les zones non protégées et seulement elles.
- N.B. : les zones non protégées sont séparées par un code puisque les zones protégées ne sont pas émises.

b) Transfer all areas.

- Code : ESC [1 h
- Définition : envoie tous les caractères des zones protégées ou non.

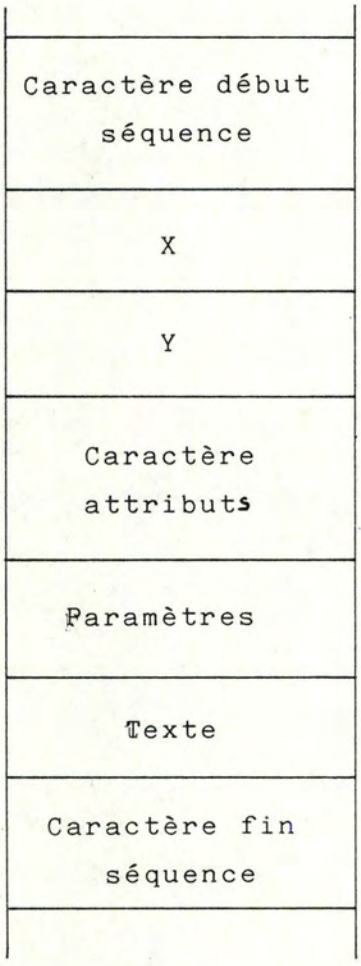
2. b suite de la page

Transfer entre page

- Code : ESC [16 l
- Définition : transfère la totalité de la page.

I) ECRAN FORMATE

1) Le format de la séquence de formatage d'une zone.



2) Définition de chaque partie.

a) Caractère début séquence.

- Code : DC 1
- Définition : le caractère représente le début de la séquence qui définitle début d'une zone sur l'écran uniquement si on est en mode formaté.

b)

c) X, Y

- Définition : représente la position sur l'écran par rapport à la position home, où va commencer la zone définie dans cette séquence.

- X est le nombre de colonne à sauter.
- Y est le nombre de ligne à sauter.

- Code : - en bytes
  - en entier
- N.B. : Les parties a, b, c, peuvent être omises si la nouvelle zone définie est contiguë à celle qu'on vient de terminer. Il n'est pas nécessaire de donner X, Y puisqu'on les connaît déjà.

  Le fait de les redéfinir n'entraîne aucune erreur.

  Si une séquence commence par le caractère-attribut alors on suppose qu'elle commence à la fin de la précédente.

d) Caractère attributs.

- Code : DLE
- Définition : caractère de contrôle qui annonce les paramètres qui seront pris comme les attributs de la zone.

e) Paramètres.

- Code : - sur un byte
  - chaque bit est un interrupteur.

7	6	5	4	3	2	1	0
h	g	f	e	d	c	b	a

- Définition de chaque bit.

  -a- Zone protégée :

  - Code : 0 : non protégée
    1 : protégée

- Définition : Toutes les données qui se trouvent dans une zone protégée ne sont pas modifiables à partir du clavier et il est impossible d'introduire une donnée dans ces zones si on est en mode protégé.

  Toutes les données introduites dans une zone non protégée sont valables, si elles répondent à la condition de types c, d, e, ou f.

-b- Zone affichable.

- Code : 0 : non affichable
  1 : affichable
- Définition : si on se trouve dans une zone non protégée et que le caractère respecte le type c, d, e, ou f alors il n'y apparaît pas à l'écran mais est envoyé à l'ordinateur.

-c- Zone à accès numérique.

- Code : 0 : non
  1 : oui
- Définition : uniquement de zone non protégée, le caractère introduit au clavier doit appartenir à l'ensemble $\{,;.;-;+;*;/;(;);0;1;...9\}$ pour être accepté.

-d- Zone à accès alphabétique.

- Code : 0 : non
  1 : oui
- Définition : uniquement dans zone non protégée, le caractère introduit au clavier doit appartenir à l'ensemble $\{a,b,...,z\}$ pour être accepté.

-e- Zone à accès digits.

- Code : 0 : non
  1 : oui
- Définition : uniquement dans zone non protégée, le caractère introduit au clavier doit appartenir à l'ensemble $\{0,1,2,\ldots,9\}$ pour être accepté.

-f- Zone à accès alphanumérique.

- Code : 0 : non
  1 : oui
- Définition : uniquement dans zone non protégée, le caractère introduit au clavier peut être quelconque.
- N.B. : - Les attributs c, d, e, f, sont exclusifs donc on ne peut trouver deux bits à 1.
  - Dans le cas où cela arrive, on prend alors l'attribut f par défaut.
  - Si ces bits sont positionnés dans une zone protégée, il n'en est pas tenu compte.

-g- Sélectionnable par crayon lumineux.

- Code : 0 : non
  1 : oui
- Définition : uniquement dans une zone non protégée, le crayon lumineux sélectionne un endroit de l'écran qui est mémorisé.

-h- Remplissage.

- Code : 0 : incomplet
  1 : complet
- Définition : uniquement dans une zone non protégée en plus des types c, d, e, f, on spécifie si la zone allouée doit être complètement remplie ou non.

F) Texte.

- Définition : - contient le texte à imprimer dans les zones protégées.
  - est vide ou contient une suite de blanc égal à la longueur de la zone si on est dans une zone non protégée.
- N.B. : - La longueur de cette zone n'a comme limite que la fin de l'écran pour le maximum et peut être vide pour le minimum.
  - ne peut pas contenir de séquence de contrôle autre que celle définissant un attribut de visualisation.

G) Caractère fin de séquence.

- Code : DC 3
- Définition : caractère qui indique la fin de la séquence de définition d'une zone.
- N.B. : Une zone se termine - 1 - au début de la suivante si rien ne permet d'en voir la fin.
  - 2 - à la fin du texte se trouvant dans la partie f. Si une nouvelle zone ne commence pas à cette endroit, alors on en rajoute une avec aucune protection.

3) Comment formater un écran.

a) entrer dans un mode formaté, en envoyant la séquence
ESC [? I7 h

b) définir chaque zone au moyen de la séquence définie en 1 et 2.
On donne donc - début de la zone en X, Y
- les attributs
- le texte s'il existe

c) faire - b - jusqu'à ce que tout l'écran soit formaté.

d) sortir du mode formaté en envoyant la séquence ESC [? 17 l

4) Comment travailler avec l'écran formaté.

Il existe une commande - pour travailler en mode formaté
ESC [? 0 h

- pour travailler en mode non formaté
donc sortir du mode formaté ESC [? 0 l

A N N E X E III.

DEFINITION DE VARIABLES GENERALES.

Nous donnons dans ce chapitre la définition et les caractèristiques des variables générales utilisées lors de l'étude de l'adaptateur du terminal (l'émulateur et la station VTP).

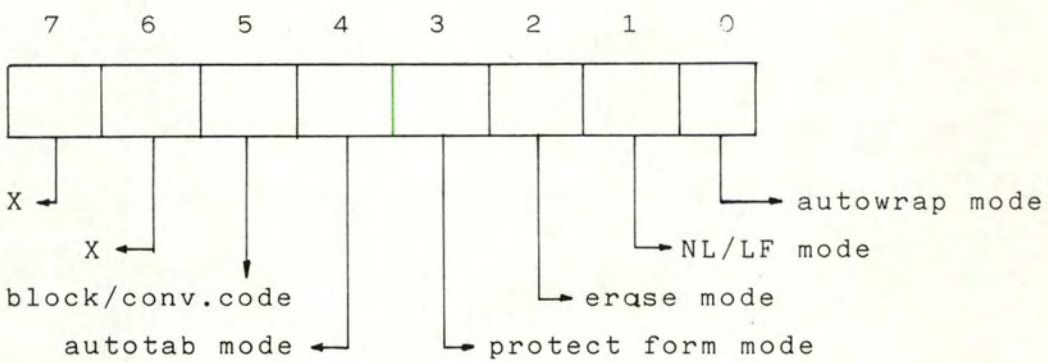
1. Les mots d'état.

1.1. STATUS-MODE :

1.1.1. Définition : variable qui représente le mot d'état reprenant à tout instant les différents modes utilisés dans le terminal virtuel. Ceux-ci sont définis dans l'annexe II au point 7.

1.1.2. Caractèristiques :

+ codé sur un byte
+ chaque bit correspond à un mode :



bit 0 : valeur 1 : mode ON
0 : mode OFF

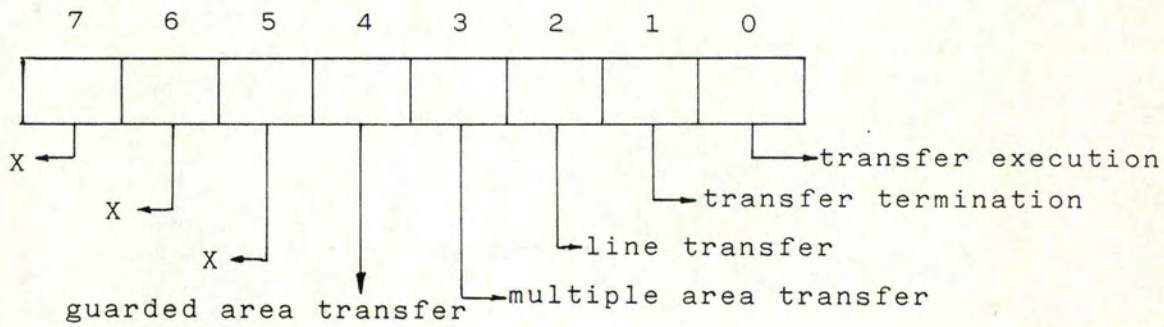
bit 2 : valeur 1 : non protégé
0 : tout
bit 3 : valeur 1 : mode ON
0 : mode OFF
bit 4 ; valeur 1 : curseur reste dans la zone
0 : curseur sauteà la zone suivante
bit 5 : valeur 1 : block mode
0 : conversation mode
bit 6, 7 : valeur indéfinie.

1.2. STATUS-TRANS:

1.2.1. Définition : variable qui représente le mot d'état reprenant à tout instant les différents modes de transmission utilisés dans le terminal virtuel. Ceux-ci sont définis dans l'annexe I au point B.

1.2.2. Caractéristiques :

+ Codée sur un byte
+ chaque bit correspond à un mode



bit 0 : valeur 1 : transfert immédiat
0 : transfert différé
bit 1 : valeur 1 : transfere toute la page
0 : transfert partiel de la page
bit 2 : valeur 1 : transfert d'une ligne
0 : transfert d'une page

bit 3 : valeur 1 : transfere la zone du curseur

0 : transfere toutes les zones non-protégées

bit 4 : valeur 1 : transfere seulement les zones non protégées

0 : transfere toutes les zones

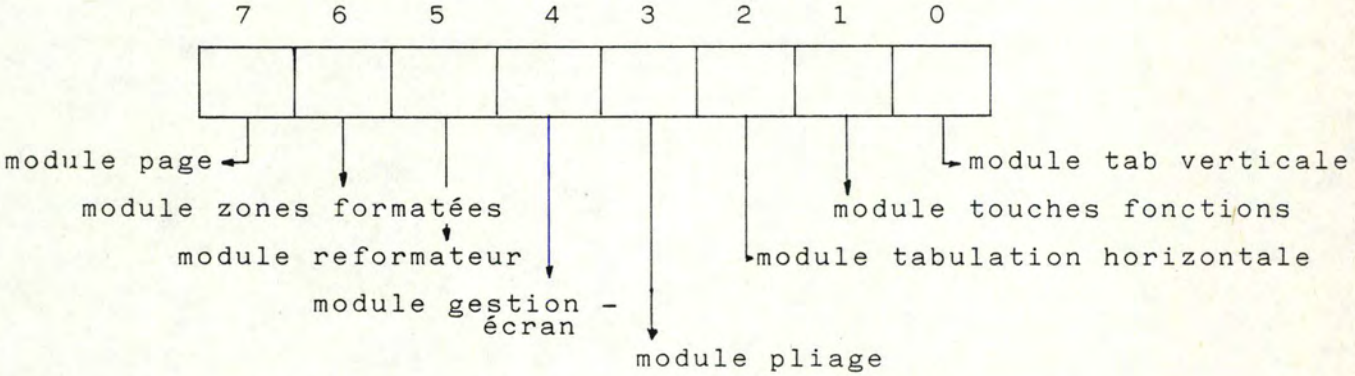
bit 5, 6, 7 : valeur indéfinie.

1.3. STATUS-EMUL :

1.3.1. Définition : variable qui représente le mot d'état reprenant à tout instant les différents modules de l'émulateur dont on a besoin pour réaliser le dialogue.

1.3.2. Caractéristiques :

+ Codée sur un byte
+ Chaque bit correspond à un module.



bit 0 - 7 : valeur 1 : module utilisé

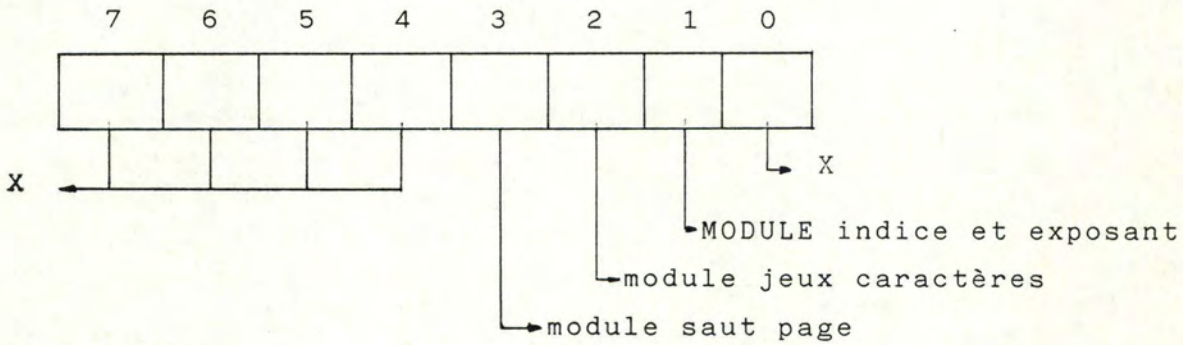
0 : module non - utilisé

1.3' STATUS-REMUL :

1.3'1. Définition : variable qui représente le mot d'état reprenant à tout instant certains modules de l'émulateur dont on a besoin pour réaliser le dialogue.

1.3'1. Caractéristiques :

- codée sur un byte
- chaque bit correspond à un module.



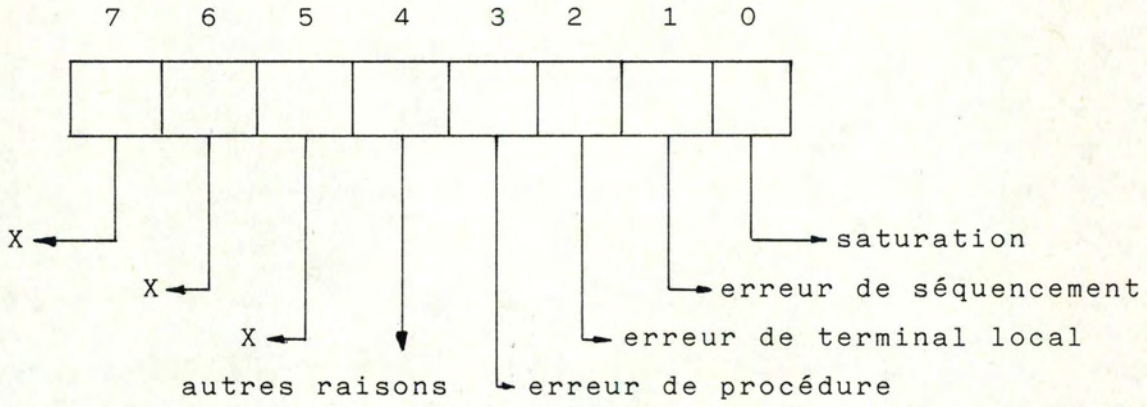
- bit : valeur 1 : module utilisé
  0 : module non utilisé

1.4. STATUS-ERREUR :

1.4.1. Définition : variable qui représente le mot d'état reprenant à tout instant les différentes erreurs s'étant produites dans la station VTP lors du traitement d'une page de données.

1.4.2. Caractéristiques :

- Codée sur un byte
- chaque bit correspond à un mode.



bit 0 - 4 : valeur 1 : une erreur s'est produite
0 : une erreur ne s'est pas produite

bit 5, 6, 7 : · valeur indéfinie
· peuvent être utilisé dans le futur
pour distinguer d'autres types d'erreurs.

2. Variables de l'écran.

2.1. ECRAN:

2.1.1. Définition : tableau qui contient la matrice permettant de mémoriser n'importe quel écran d'un terminal créateur, et qui est utilisé par le module gestion écran.

2.1.2. Caractéristiques :

+ ECRAN[0...159, 0..47]
+ un élément du tableau contient 1 caractère
+ les caractères utilisables sont ASCII (0/0 → 7/15)
+ la matrice est mise à NUL (0/0) à l'initialisation et lors de tout effacement.

2.2. MAXCAR:

2.2.1. Définition : variable contenant le nombre maximum de caractères par ligne utilisable dans la matrice ECRAN.

2.2.2. Caractéristiques :

+ valeur en entier
+ codée sur un byte
+ valeurs extrêmes entre 0 et 159
+ obtenue lors de la négociation;

2.3. MAXLIG:

2.3.1. Définition : variable contenant le nombre maximun de lignes par page utilisable dans la matrice ECRAN.

2.3.2. Caractéristiques :

+ valeur en entier
+ codée sur un byte
+ valeurs extrêmes comprises entre 0 et 47
+ obtenue lors de la négociation.

2.4. MAXCART ;

2.4.1. Définition : variable contenant le nombre maximum de caractères par ligne imprimable sur le terminal réel.

2.4.2. Caractéristiques :

+ valeur en entier
+ codée sur un byte
+ valeurs extrêmes comprises entre 0 et 159.

2.5. MAXLIGT:

2.5.1. Définition : variable contenant le nombre maximum de lignes par page imprimable sur le terminal réel.

2.5.2. Caractéristiques :

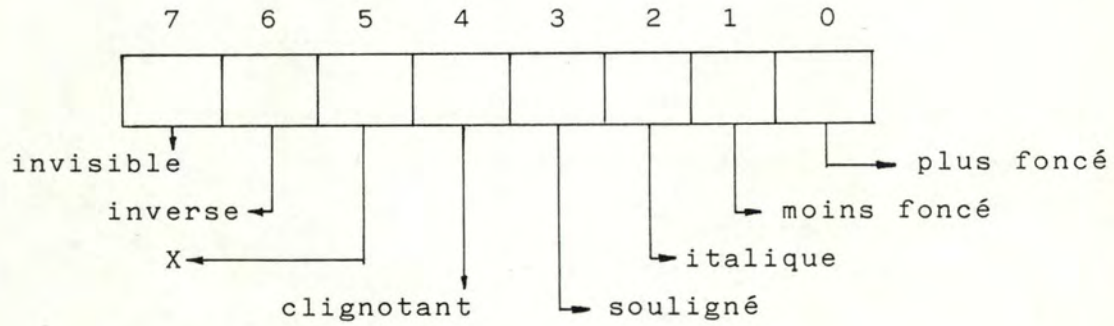
+ valeur en entier
+ codée sur un byte
+ valeurs extrêmes comprises entre 0 et 47

2.6. ATTRIB :

2.6.1. Définition : tableau qui contient la matrice permettant de mémoriser les attributs de visualisation des caractères de la matrice utile (ECRAN)

2.6.2. Caractéristiques :

+ ATTRIB [0..159;0...47]
+ un élément contient un byte qui contient les attributs de visualisation
+ codage



+ si le byte est à 0,on a un caractère normal
+ à l'initialisation la matrice est mise à 0

2.7. ATTRIBUT-CURSEUR :

2.7.1. Définition : variable mémorisant les attributs de visualisation des caractères pointés par le curseur.

2.7.2. Caractéristiques :

+ un byte
+ codage voir 2.6.2
+ à l'initialisation, ce byte est à zéro.

3. Curseur.

3.1. X et Y:

3.1.1. Définition : variables contenant la position du curseur dans la matrice ECRAN. Ce curseur est appelé curseur fictif.

3.1.2. Caractéristiques :

X : + valeur en entier
+ codée sur un byte
+ valeurs extrêmes comprises entre 0 et MAXCAR.

Y : + valeur en entier
+ codée sur un byte
+ Valeurs extrêmes comprises entre 0 et MAXLIG.

3.2. DEPCURX et DEPCURY:

3.2.1. Définition : variables contenant la position du curseur sur l'écran réel du terminal.

3.2.2. Caractéristiques :

DEPCURX : + valeur en entier
+ codée sur un byte
+ valeurs extrêmes comprises entre 0 et le plus grand nombre de caractères par ligne disponible sur le terminal.

DEPCURY : + valeur en entier
+ codé sur un byte
+ valeurs extrêmes comprises entre 0 et le plus grand nombre de lignes par page disponible sur le terminal.

4. Les marges.

4.1. MVG :

4.1.1. Définition : variable contenant la position absolue de la marge verticale gauche.

4.1.2. Caractéristiques :

+ valeur en entier
+ codée sur un byte
+ $0 \leqslant MVG \leqslant$ MAXCAR ou MAXCART

4.2. MVD :

4.2.1. Définition : variable contenant la position absolue de la marge verticale droite.

4.2.2. Caractéristiques :

+ valeur en entier
+ codée sur un byte
+ $0 < MVD \leqslant$ MAXCAR ou MAXCART
+ $MVD < MVG$

4.3. MHS :

4.3.1. Définition : variable contenant la position absolue de la marge horizontale supérieure.

4.3.2. Caractéristiques :

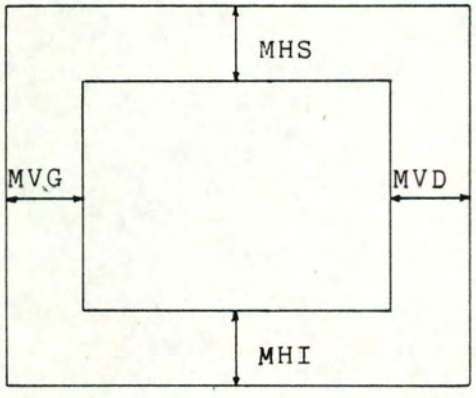
+ valeur en entier
+ codée sur un byte
+ $0 \leqslant MHS \leqslant$ MAXLIG ou MAXLIGT

4.4. MHI :

4.4.1. Définition : variable contenant la position absolue de la marge horizontale inférieure.

4.4.2. Caractéristiques :

+ valeur en entier
+ codée sur un byte
+ $0 \leq MHI \leq MAXLIG$ ou MAXLIGT
+ $MHI < MHS$



5. La tabulation.

5.1. TAB-AUX-H et TAB-AUX-V:

5.1.1. Définition : tableaux mémorisant les différentes positions des tabulations stops horizontales et verticales.

5.1.2. Caractéristiques :

+ TAB-AUX-H [0...159]
+ chaque élément du tableau sera
  - en entier
  - 0 ⩽ valeur ⩽ MAXCAR ou MAXCART
  - en ordre croissant
  - codé sur un byte

+ TAB-AUX-V [0...47]
+ chaque élément du tableau sera
  - un entier
  - 0 ⩽ valeur ⩽ MAXLIG ou MAXLIGT
  - en ordre croissant
  - codé sur un byte.

5.2. TAB-DEF-H et TAB-DEF-V:

5.2.1. Définition : tableaux mémorisant les différentes positions des tabulations stops horizontales (H) et verticales (V) prises par défaut.

5.2.2. Caractéristiques :

+ les caractéristiques sont identiques à TAB-AUX-H et TAB-AUX-V.

5.3. TAB-PROTEGE:

5.3.1. Définition : tableau mémorisant le début de chaque zone non protégée.

5.3.2. Caractéristiques :

+ TAB-PROTEGE [0...100]
+ chaque élément du tableau sera
  - un entier sur deux bytes
  - $0 \leqslant$ valeur $\leqslant 160 * 48$
  - en ordre croissant.

6. Divers.

6.1. ZONE:

6.1.1. Définition : ensemble de variables contenant toutes les informations sur une zone de l'écran. Elle est composée de trois parties.

6.1.2. Caractéristiques.:

ZONE-POSMIN : variable contenant la première position de la zone.

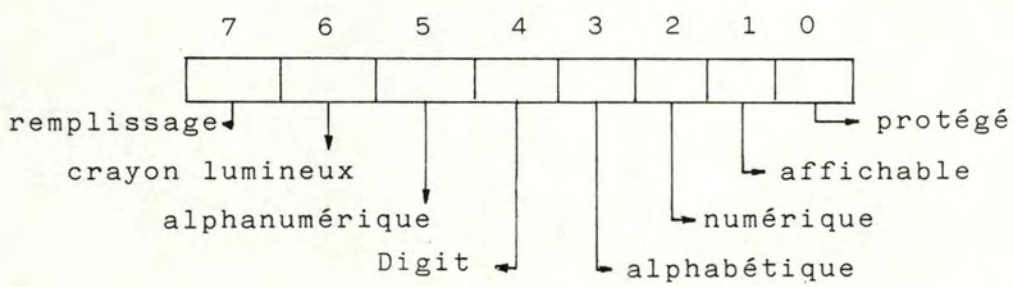
- en entier
- codée sur deux bytes
- adresse absolue
- 0 ⩽ POSMIN ⩽ MAXCAR * MAXLIG ou MAXCART * MAXLIGT

ZONE-POSMAX : variable contenant la dernière position de la zone.

- en entier
- codée sur deux bytes
- adresse absolue
- 0 ⩽ POSMAX ⩽ MAXCAR * MAXLIG ou MAXCART * MAXLIGT

ZONE-TYPE : variable contenant les attributs de la zone comprise entre ZONE-POSMIN et ZONE-POSMAX.

- en entier
- codée sur un byte



Bits 0 - 7 : valeur 1 : attribut utilisé
0 : non utilisé

6.2. LANG :

6.2.1. Définition : variable indiquant quel alphabet est utilisé sur le terminal à l'instant courant.

6.2.2. Caractéristiques :

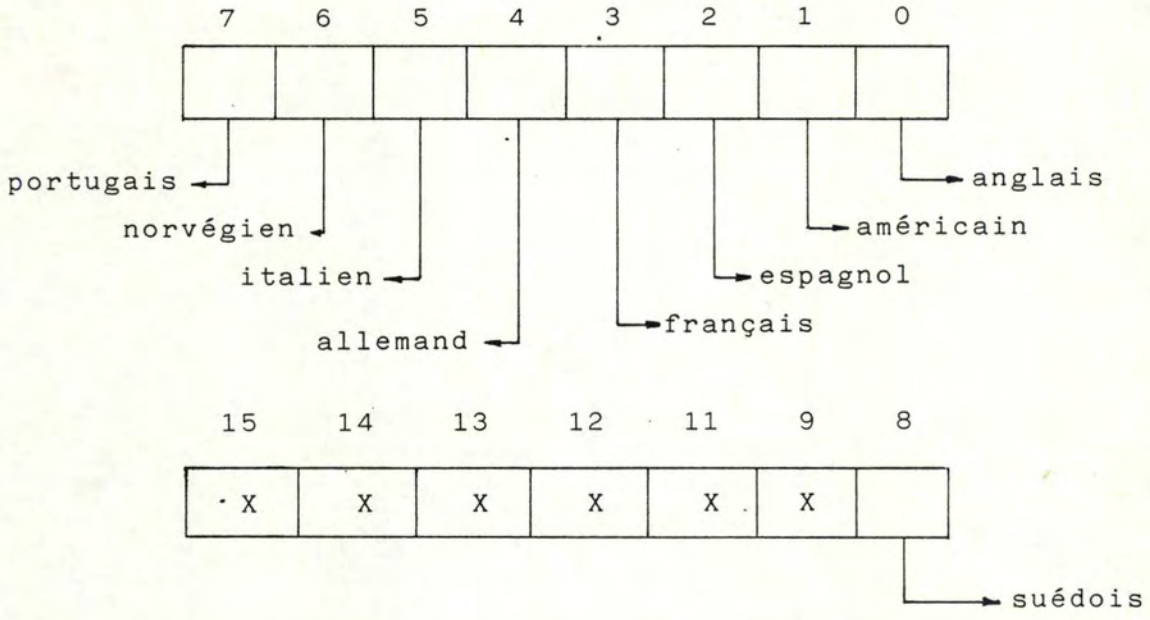
+ valeur en entier
+ codée sur un byte
+ codage
  - 0 : anglais
  - 1 : américain
  - 2 : espagnol
  - 4 : français
  - 8 : allemand
  - 16 : italien
  - 32 : danois/norvégien
  - 64 : suédois/finlandais
  - 128 : portugais
+ ces valeurs sont les seules permises si une erreur se produit, on prendra la valeur "0" comme correction.

6.3. JEU :

6.3.1. Définition : tableau contenant les alphabets à transformer s'ils sont utilisés car il n'existe pas sur le terminal.

6.3.2. Caractéristiques :

+ valeur en entier
+ codée sur deux bytes



+ bits 0 - 8 : valeur 1 : doivent être transformé
0 : ne doivent pas être transformé.
+ bits 9 - 15 : valeur indéfinie.

6.4. FONCT :

6.4.1. Définition : tableau contenant le programme de chaque touche de fonctions programmables.

6.4.2. Caractéristiques :

+ FONCT 0 ..703
+ 11 positions sont réservées par touche :

position 1 : pour un compteur indiquant le nombre réel de caractères mémorisés.

2 - 11 : pour les caractères mémorisés.

+ nombre maximum de touches égal 64
+ Caractères enregistrés sont - en ASCII
- de 0/0 → 7/15

7. VECTEUR DE NEGOCIATION-RESULTAT.

7.1. NEGOS :

7.1.1. Définition : tableau mémorisant le vecteur résultat de la négociation.

7.1.2. Caractéristiques :

+ NEGOS [1 ..35]
+ les éléments sont - sous forme d'entier
  - sur un byte
+ les éléments d'indice i correspondent à :

1 : le type d'application, la mode et le type de terminal
2 : le nombre de caractères par ligne
3 : le nombre de lignes par page
4-5 : les mots d'état des tabulations horizontales et verticales
6 : le mot d'état pour les indices et les exposants
7-10 : les caractères de contrôle de IA5
11-27 : les séquences de contrôle standards
28-29 : les jeux de caractères
30 : les touches de fonctions
31-32 : les attributs des zones
33-35 : les séquences de contrôle de transmission

N.B. : Tous ces éléments sont décrits en détails au chapitre VII.

A L G O R I T H M E S D E S P R I M I T I V E S.

1.1. Algorithme gérant la primitive CDCL.

1.1.1. Le programme principal.

* Définition :

Il réalise l'analyse de la primitive de négociation, zone par zone et il permet l'obtention du vecteur résultat. Il envoie ensuite une réponse positive ou négative à son correspondant suivant le résultat de l'analyse.

* Variables globales.

negoR : tableau contenant le vecteur utilisateur. Tous les bytes constitutifs sont décrits au chapitre VII.

negoE : tableau contenant le vecteur créateur. Tous les bytes constitutifs sont décrits au chapitre VII.

negoS : tableau contenant le vecteur résultat. Il est décrit dans la définition des variables générales.

carE :
carR : dernier caractère lu ou écrit dans les vecteurs
carS : respectifs.

sol, sol1 , sol2 , sol3 , sol4 : variables intermédiaires.

nature : indique si on est dans le terminal utilisateur ou chez le créateur.

nature = 0 chez le créateur
nature = 255 chez l'utilisateur

PointE, pointR : variables indiquant le début de la partie à analyser respectivement dans le vecteur negoE et negoR.

param : variable permettant au programme principal de savoir ou en est l'analyse des zones du vecteur :

Param = 0 : on analyse
1 : les deux terminaux sont égaux et l'analyse est terminée.
2 : une erreur s'est produite lors de l'analyse et celle-ci doit se terminer.
3 : on a terminé l'analyse.

cause : variable indiquant la cause de l'erreur lors de l'analyse d'une zone ou une partie.

Cause : 1 : incompatibilité entre les deux dialoguants.
2 : il existe une erreur de paramètre.

type : variable indiquant le type de l'appareil dans le vecteur résultat.

type = 0 : vidéo
1 : télétype
2 : imprimante.

type d'applic : variable indiquant le type d'application que vont traiter les deux partenaires.

type-applic 1 : traitement de texte
2 : questions-réponses
3 : zones formatées
4 : texte libre 1
5 : texte libre 2

mode : variable indiquant le mode du terminal repris dans le vecteur résultat.

mode : 1 : SCROLL
2 ; PAGE
4 : DATA ENTRY

maxcar , maxlig, maxcarT,
maxligT, status_emul,
status_remul, jeu,
tab_aux_H, tab_aux_ V
tab_def_H, tab_def_ V

ces variables sont définies au paragraphe "définitions et variables générales".

K : variable indiquant le numéro de la zone qu'on va analyser

w, x, y, : variables qui pointent l'élément suivant à analyser respectivement dans negoE, negoR et negoS.

paramètre C : tableau en mémoire de la station VTP contenant le vecteur créateur.

paramètre V : tableau en mémoire de la station VTP contenant le vecteur utilisateur.

paramètre R : tableau contenant les paramètres de la primitive CDCL reçue de l'autre.

* Algorithme : après réception d'un CDCL.

```
si nature = 0  alors [envoyer (CDCL + paramètre C)
                      negoE = paramètre C
                      negoR = paramètre R

                sinon [envoyer (CDCL + paramètre V)
                      negoE = paramètre R
                      negoR = paramètre V

k = 1
param = 0
tant que param = 0

        [ si k =
          [
            1 : analyse identification
            2 : analyse format
            3 : analyse caractères IA5
            4.: analyse séquences
            5 : analyse jeux caractéres
            6 : analyse touches
            7 : analyse zones
            8 : analyse transmission

si param =

     [1 : envoyer (RDCLP + negoE)
     2 : envoyer (RDCLN + cause)
     3 : envoyer (RDCLP + negoS)

wait
```

pour l'émission d'un CDCL.

```
si nature =  0 alors  envoyer (CDCL + paramètre C)
               sinon  envoyer (CDCL + paramètre V)

wait
```

1.1.2. Etude de routines d'intérêt général.

1.1.2.1. Routine LIRE

- Définition : elle permet de lire dans les vecteurs negoE et negoR le caractère suivant et de l'assigner respectivement aux variables carE et carR.

- Variable locale :

- Algorithme

LIRE

```
carE = negoE [w]
w = w + 1
carR = negoR [x]
x = x + 1
retour.
```

1.1.2.2. Routine EXTRAIRE (i,j,car)

- Définition : elle permet d'extraire les bits compris entre les positions i et j exprimées en bit dans le byte "car".

- Variables locales : m : indice de boucle.
  scar : variable intermédiaire permettant la formation de fonction.
  i,j : variables indiquant la position des bits à extraire du byte.
  car : variable contenant la valeur sur laquelle va se faire l'extraction.

```
EXTRAIRE (i, j, car)

┌ pour m = 1 ⟶ i
│        [car = car  div  2
│ scar = 0
│ pour m = 0 ⟶ (j - i )
│        [scar = scar * 2 + 1
│ extraire = scar ET car
└ retour
```

1.1.2.3. Routine METTRE (i, j, p, car)
****

- Définition : elle permet de déposer la valeur "p" dans le byte "car" entre les positions i et j exprimées en bit; (i est le bit de poids le plus faible).

- Variables locales : m : indice de boucle.
  i, j : variables indiquant la position des bits à mettre dans les bytes.
  p : variable contenant la valeur à déposer dans le byte.
  car : variable contenant le byte sur lequel on va mettre ces nouveaux bits.

- Algorithme.

```
METTRE (i, j, p, car)

┌ pour m = 1⟶ j
│        [p = p * 2 + 1
│ pour m = 0⟶ (j-1)
│        [p = p * 2
```

```
pour m = 1 → i
        [p = p * 2 + 1
car = p ET car
pour m = 1 → i
        [p = p * 2
car = p OU car
retour
```

1.1.2.4. Routine DEB_ZONE.
****

- Définition : au début de chaque partie et pour les vecteurs negoR et negoE, elle met à jour la variable qui pointe le début de la partie suivante. Elle positionne un pointeur sur le premier élément à lire.

- Variables locales. : pointE : pointeur indiquant dans le vecteur negoE le début de la partie suivante à analyser.
  pointR : idem pointE mais dans le vecteur negoR.

- Algorithme.

DEB _ ZONE

```
w = pointE + 1
pointE = pointE + negoE [pointE]
x = pointR + 1
pointR = pointR + negoR [pointR]
retour
```

1.1.2.5. Routine TRT_OU (i,j)
****

- Définition : elle permet de réaliser un OU logique entre les bits compris entre la position i et j des bytes carE et carR et mettre la solution dans carS.

- Variables locales : i, j : variables indiquant la position des bits à traiter.

- Algorithme.

TRT __ OU (i,j)

```
sol = extraire (i,j,carE) OU extraire (i,j,carR)
mettre (i,j,sol,carS)
retour
```

1.1.2.6. Routine TRT __ ET (i,j)
****

- Définition : elle permet de réaliser un ET logique entre les bits compris entre la position i et j des bytes carE et carR et mettre la solution dans carS.

- Variables locales : i et j : variables indiquant la position des bits à traiter.

- Algorithme.

TRT __ ET (i,j)

```
sol = extraire (i,j,carE) ET extraire (i,j,carR)
mettre (i,j,sol,carS)
retour
```

1.1.2.7. Routine VALID1 (car, ens)
****

- Définition : elle teste si le caractère "car" appartient à l' ensemble défini par "ens".

- Variable locale : néant

- Algorithme.

VALID1 (car, ens)

```
si car ∈ens alors VALID1 = vrai
              sinon VALID1 = faux

retour
```

1.1.2.8. Routine VALID2 (i)
****

- Définition : elle vérifie que la longueur de la partie que nous allons analyser est bien de longueur "i" imposée par la définition du vecteur de négociation.
- Variable locale : i : variable donnant la longueur normalisée de la partie à tester.
- Algorithme.

VALID2 (i)

```
VALID2 = faux
si i ≠ negoE [w-1] alors VALID2 = vrai
si i ≠ negoR [x-1] alors VALID2 = vrai
retour
```

1.1.2.9. Routine VALID3 (i) (VALID4 (i))
****

- Définition : elle vérifie que la longueur de la partie que nous allons analyser est plus petite ou égale (plus grande ou égale) à la longueur "i" imposée par la définition du vecteur de négociation.
- Variable locale : i : variable donnant la longueur normalisée de la partie à tester.
- Algorithme.

VALID3 (i)

```
si i< (>) negoE [w-1] alors VALID3 = faux (VALID4=faux)
si i< (>) negoR [x-1] alors VALID3 = vrai (VALID4=VRAI)
retour
```

1.1.2.10 Routine TEST_ SOL
****

- Définition : elle réalise un test permettant de savoir s'il faut utiliser un module de l'émulateur suivant qu'on se trouve chez l'utilisateur ou le créateur.

- Variable locale : néant

- Algorithme.

TEST __ SOL

```
si (sol1 = 0 ET nature = 0) OU (sol2 = 0 ET nature ≠ 0 )

        alors test_sol = vrai

        sinon test_sol = faux

retour
```

1.1.3. Routines d'analyse des zones.

1.1.3.1. Routine ANALYSE _ IDENTIFICATION.

- Définition : elle réalise l'analyse de la zone d'identification en suivant les règles énoncées au chapitre VII et en appelant des sous-routines effectuant l'analyse des parties : nom, numéro, type d'application.

- Variable globale : bool : variable booléenne indiquant s'il y a égalité entre les deux noms ou numéros des terminaux.

- Algorithme.

ANALYSE _ IDENTIFICATION

```
w=1 ; x=1 ; y=1 ; pointE = 1 ; pointR = 1
deb_zone
si valid3 (40) alors [param = 2
                      cause = 1
                      retour

bool = vrai
analyse_nom
si bool= vrai alors  [deb_zone
                      si valid3 (40) alors [param = 2
                                            cause = 1
                                            retour

                      analyse_nom

si bool= vrai alors [negoS = negoE
                     param = 1
                     retour

deb_zone
```

```
si valid2 (2) alors [param = 2
                     cause = 1
                     retour

lire
si valid1(carE,ens) OU valid1(carR,ens) alors [param = 1
                                               cause = 1
                                               retour

sol1 = extraire (6,7,carE)
sol2 = extraire (0,2,carE)
sol3 = extraire (6,7,carR)
sol4 = extraire (0,2,carR)
si (sol1 ≠ 0 ET sol2≤ 1)OU(sol3 ≠ 0 ET sol4≥ 1) alors [param = 2
                                                       cause = 1
                                                       retour

sol = extraire (3,5,carE)
switch sol=

      [1: applic_TT
      2: applic_QR
      3: applic_ZF
      4: applic_TL1
      5: applic_TL2

k = k + 1
retour
```

1.1.3.1.1. Routine ANALYSE_NOM.

+ Définition : elle compare deux chaînes de caractères des vecteurs negoE et negoR et elle vérifie que les chaînes suivent les règles énoncées au chapitre VII

+ Variables locales : longE : variable indiquant la longueur de la chaîne de caractères du vecteur negoE.

longR : variable indiquant la longueur de la chaîne de caractères du vecteur negoR.

+ Algorithme.

ANALYSE_NOM

```
longE = negoE [w-1] -1
longR = negoR [x-1] -1
si longE= longR alors tant que longE ≠ 0 OU bool = vrai
                        lire
                        si carE ≠ carR alors bool = faux
                        si carE= blanc OU "." alors bool = faux
                        si carR= blanc OU "." alors bool = faux
                        longE = longE - 1
retour
```

1.1.3.1.2. Routine APPLIC_TT

+ Définition : elle analyse le type et le mode de l'appareil pour une application "traitement de texte".

+ Variable locale : néant

+ Algorithme.

APPLIC_TT

```
APPLIC_TT

mettre (3,5,sol,carS)
sol = extraire (6,7,carE)
mettre (6,7,sol,carS)
type = sol
mettre (0,2,1,carS)
mode = 1
negoS (y) = carS
y = y + 1
type_applic = 1
si nature ≠ 0 alors envoyer (ESC [ ? 101)
retour
```

1.1.3.1.3. Routine APPLIC_QR

+ Définition : elle analyse le type et le mode de l'appareil pour une application "questions-réponses".

+ Variable locale : /

+ Algorithme.

```
APPLIC_QR

Mettre (3,5,sol,carS)
sol2 = extraire (6,7,carR)
si sol2 ≠ 0 alors [param = 2
                   cause = 2
                   Retour
mettre (6,7,0,carS)
type = 0
sol1 = extraire (0,2,carE)
sol2 = extraire (0,2,carR)
si sol1 =
```

```
    1 : mettre (0,2,1,carS)
        si (sol2≠ 1) ET (nature = Ø) alors envoyer (ESC [ ? 101)
        mode = 1

    2 : mettre (0,2,2,carS)
        mode = 2
        si (sol2 = 1 ET nature ≠ 0) alors  mettre (7,7,1,status_emul)
                                           envoyer (ESC [ ? 10h)
                                    sinon  si(nature ≠ 0) alors
                                           envoyer (ESC [ ? 10h)

    4 : mettre (0,2,4,carS)
        mode = 4
        si (sol2 ≠ 4 ET nature ≠ 0) alors  mettre (6,6,1,status_emul)
        si sol2 ≠ 1 alors envoyer (ESC [ ? I0h)

negoS [y] = carS
y = y + 1
type_applic = 2
retour
```

1.1.3.1.4. Routine APPLIC_ZF.

+ Définition : elle analyse le type et le mode de l'appareil pour une application "zones-formatées".

+ Variable locale : /

+ Algorithme.

APPLIC _ ZF

```
mettre (3,5,sol,carS)
sol1 = extraire (6,7,carE)
sol2 = extraire (0,2,carE)
```

```
si sol1 ≠ 0 ET sol2 = 1 alors [param = 2
                               cause = 2
                               retour

sol3 = extraire (6,7,carR)
sol4 = extraire (0,2,carR)
si Sol3≥ 1 alors [param = 2
                  cause
                  retour
mettre (0,2,sol2,carS)
mode = sol2
mettre (6,7,0,carS)
type = 0
negoS [y] = carS
y = y + 1
type_applic = 3
si (sol4 ≠ 4 ET sol2 ≠ 4 ET nature ≠ 0) alors mettre (6,6,1,status_émul)
si (sol4= 1 ET sol2= 2 ET nature ≠ 0) alors mettre (7,7,1,status_émul
si nature ≠ 0 alors envoyer (ESC [ ? 10h)
retour
```

1.1.3.1.5. Routine APPLIC _ TL1.

+ Définition : elle analyse le type et le mode de l'appareil pour une application "texte libre 1".

+ Variable locale : /

+ Algorithme.

APPLIC - TL1

```
mettre (3,5,sol,carS)
sol1 = extraire (6,7,carE)
sol2 = extraire (6,7,carR)
mettre (6,7,sol1,carS)
```

```
type = sol1
sol3 = extraire (0,2,carE)
sol4 = extraire (0,2,carR)
switch sol3
    1 :   mettre (0,2,1,carS)
          si nature ≠ 0 alors  envoyer (ESC [? 10 1)
          mode = 1

    2 :   si (sol4= 1 ET nature ≠ 0) alors mettre(7,7,1,status_emul
          mettre (0,2,2,carS)
          si nature ≠ 0 alors  envoyer (ESC [? 10h)
          mode = 2

    4 :   si (sol4 ≠ 4 ET nature ≠ 0) alors mettre
                                          (6,6,1,status_émul)
          mettre (0,2,4,carS)
          si nature ≠ 0 alors envoyer (ESC [? 10h)
          mode = 4

negoS [y] = carS
y = y + 1
type _ applic = 4
retour
```

1.1.3.1.6. Routine APPLIC _ TL2

+ Définition : elle analyse le type et le mode de l'appareil pour une application "texte libre 2".

+ Variable locale : /

+ Algorithme.

APPLIC _ TL2

```
APPLIC_TL2
┌
mettre (3,5,sol,carS)
sol1 = extraire (6,7,carE)
sol2 = extraire (6,7,carR)

si sol2 = 2 alors ┌ mettre (6,7,sol2,carS)
                  └ type = sol2

           sinon  ┌ mettre (6,7,sol1,carS)
                  └ type = sol1

sol3 = extraire (0,2,carE)
sol4 = extraire (0,2,carR)
si sol3 =
      ┌
      1 : si sol4 =
            ┌
            1 : ┌ mettre (0,2,sol4,carS)
                └ mode = sol4

            2 : ┌ si sol1 ≠ 0 alors ┌ param = 2
                │                    │ cause = 2
                │                    └ retour
                │
                │ mettre (0,2,sol4,carS)
                │ si nature = 0 alors
                │      mettre   (7,7,1, status-émul)
                └ mode = 2

            4 : ┌ si sol1 ≠ 0 alors ┌ param = 2
                │                    │ cause = 2
                │                    └ retour
                │
                │ mettre (0,2,sol4,carS)
                │ si nature = o alors
                │      mettre (6,6,1,status-émul)
                └ mode = 4
```

```
2 : si sol2 =
        1,2 : param = 2
              cause = 2
              retour
        0   : si sol4 =
              1 : mettre (0,2,2,carS)
                  mode = 2
                  si nature ≠ 0 mettre (7,7,1,status_emul)
              2 : mettre (0,2,2,carS)
                  mode = 2
              4 : mettre (0,2,4,carS)
                  mode = 4
                  si nature ≠ 0 mettre (6,6,1,status_emul)
4 : si sol2 =
        0   : mettre (0,2,4,carS)
              mode = 4
              si (sol4 ≠ 4) ET (nature ≠ 0) alors mettre
                                              (7,7,1,status_emul)
        1,2 : param = 2
              cause = 2
              retour
negoS [y] = carS
y = y + 1
type_applic = 5
retour.
```

1.1.3.2. Routine ANALYSE_FORMAT.
****

- Définition : elle analyse la zone "format" dans les vecteurs de négociation en suivant les règles énoncées au chapitre VII.

- Variables locales :

  bool1, bool2 : variables booléennes indiquant si on a trouvé un format adéquat.

  last : variable mémorisant le dernier format disponible dans le vecteur de l'utilisateur.

  long : longueur de la zone contenant les formats de l'utilisateur.

  flag : variable indiquant si on utilise les tabulations horizontales ou verticales.

- Algorithme.

  ANALYSE_FORMAT.

```
deb_zone
si valid4 (2) alors [param = 2
                     cause = 2
                     retour

bool1 = vrai
analyse_ligne (bool1)
si bool1 ET type > 0 alors [deb_zone
                            valid4(1) alors [param = 2
                                             cause = 1
                                             retour

                            long = negoR [ x]
                            x = x+1
                            deb_zone
                            si valid4(1)  alors [param = 2
                                                 cause = 1
                                                 retour
```

```
                                trt_impr (bool1)

si not(bool1) ET nature ≠ 0 alors commuter (last)
                                  si carE < last alors
                                          mettre (3;3,1,status_émul

si bool1 ET nature ≠ 0 alors
   si type_applic =
      1   : mettre (5,5,1,status_emul)
      2,3 : si type = 0 alors mettre (4,4,1,status_émul)
                        sinon param = 2
                              cause = 2
                              retour
      4,5 : si type = 0 alors mettre (4,4,1,status_émul)
                        sinon mettre (3,3,1,status_émul)
   commuter (last)
maxcar = carE ; maxcarT = last
bool2 = vrai
deb_zone
si valid4(2) alors param = 2
                   cause = 2
                   retour
analyse_ligne (bool2)

si bool2 ET type > 0 alors deb_zone
                           si valid4(1) alors param = 2
                                              cause = 1
                                              retour
                           long = negoR [x]
                           x = x + 1
                           deb_zone
                           si valid4(1) alors param = 2
                                              cause = 1
                                              retour
                           trt_impr (bool2)
```

```
si bool2 = faux ET nature ≠ 0 alors [commuter (last)
                                    [si carE < last alors
                                        mettre (3;3,1,status_émul)

si bool2 = faux ET nature ≠ 0 alors

   [si type applic=
      [1    : [si extraire (5,5,status_émul) ≠ 1 alors
               [                              mettre (3,3,1,status_remul)
      [2,3  : [si type = 0 alors [mettre (4,4,1,status_émul)
               [                 sinon [param = 2
               [                       [cause = 2
               [                       [retour
      [4,5  : [si type = 0 alors [mettre (4,4,1,status_émul)
               [                 sinon [mettre (3,3,1,status_remul)
   [commuter (last)

maxlig = carE ; maxligT = last
deb_zone
si valid4(2) OU valid3 (maxcar) alors [param = 2
                                      [cause = 1
                                      [retour
flag =  0
trt_tabulat(2)
si flag = 1 alors retour
flag = 0
deb_zone
si valid4(2) OU valid3 (maxlig) alors [param = 2
                                      [cause = 1
                                      [retour
```

```
trt_tabulat (0)
si flag = 1 alors retour
deb_zone
si valid2(2) alors [param = 2
                    cause = 1
                    retour
trt_indexp
k = k + 1
retour
```

1.1.3.2.1. Routine ANALYSE_LIGNE (Z)

- Définition : elle recherche dans la liste des formats celui qui pourrait convenir. Le paramètre indique la réussite dans la recherche.

- Variables locales :

  longR : variable indiquant la longueur de la zone à lire dans le vecteur utilisateur.

  i : indice de boucle

  z : booléen indiquant qu'on a trouvé un format utilisable.

- Algorithme.

  ANALYSE_LIGNE (Z)

```
carE = negoE [w]
w = w + 1
longR = negoR [x-1] -1
pour i = 1 → longR
     [carR = negoR [x]
     last = carR
     x = x + 1
```

```
si carR ⩾ carE alors [ i = longR
                       z = faux
                       negoS [ y ] = carE
                       y = y + 1
retour
```

1.1.3.2.2. Routine TRT_IMPR (Z)

- Définition : elle permet, pour une imprimante ou un télétype, de trouver un format compatible en changeant l'interligne et l'espacement entre caractères.

- Variables locales :

  longR : variable indiquant la longueur de la partie à lire dans le vecteur utilisateur.

  i : indice de boucle

  z : booléen indiquant qu'on a trouvé un format utilisable.

  taille : exprime un nombre de caractères par ligne à partir de la largeur du formulaire et du nombre de caractères par inch (idem pour l'autre sens).

- Algorithme.

  TRT_IMPR (Z)

```
longR = negoR [ x - 1] -1
pour i = 1 → longR
        carR = negoR [x]
        x = x + 1
        taille = long *10  div  carR
```

```
si carE ⩾ taille alors [ i = lαngR
                       | z = faux
                       | negoS [y] = carE
                       | y = y + 1
                       | lαst = taille
retour
```

1.1.3.2.3 routine TRT_TABULAT (C)

- Définition : elle permet le traitement de la partie "tabulation horizontale et verticale" dans les vecteurs de négociation.

- Variables locales :

  c : paramètre indiquant le bit à positionner dans le mot d'état status_émul.

  i : indice de boucle.

  avant : variable mémorisant l'avant dernière tabulation stop pour pouvoir la comparer à la dernière lue et vérifier la croissance entre elles.

- Algorithme.

  TRT_TABULAT(C)

```
lire
trt_prog(0)
trt_prog(1)
si extraire (C,C,status-émul) = 1 alors [ avant = -1
                                        | pour i=1 → negoE [w-1] -1
                                        |      carE = negoE [w]
                                        |      w = w + 1
```

```
si avant ≥ carE          alors [flag = 1
                               param = 2
                               cause = 1
                               retour]

si c = 2          alors
                        [tab_aux_H [ i ] = carE
                        tab_def_H [ i ] = carE

                  sinon
                        [tab_aux_V [ i ] = carE
                        tab_def_V [ i ] = carE

avant = carE            avant = carE

retour
```

1.1.3.2.4. Routine TRT_PROG (a)

- Définition : elle vérifie qu'il y a une fonction tabulation et qu'elle est programmable.
- Variable locale : a : variable indiquant le bit à traiter dans le mot d'état des tabulations.
- Algorithme.

TRT_PROG (a)

```
sol1 = extraire (a,a,carE)
sol2 = extraire (a,a,carR)

si type_applic ≠ 5 alors [mettre (a,a,sol1,carS)
                          si sol2 =  0 ET nature ≠ 0
                            alors mettre (c,c,1, status_emul)

                sinon [sol=sol1 OU sol2
                       mettre (a,a,sol,carS)
                       si test_sol alors mettre (c,c,1,status_
                                                 emul)

retour
```

1.1.3.2.5. Routine TRT_INDEXP
==========

- Définition ; elle traite le problème des indices et des exposants.

- Variable locale :

- Algorithme :

TRT_INDEXP

```
lire
trt_OU (1,1)
si sol= 1 ET test_sol alors mettre (1,1,1,status_remul)
trt_OU (0,0)
si sol=  1 ET test_sol alors mettre (I,1,1,status_remul)
negoS [ y ] = carS
y = y + 1
retour
```

1.1.3.3. Routine ANALYSE_CARACTERES _ IA5.
****

- Définition : elle analyse la zone "caractères de contrôle" en suivant les règles énoncées au chapitre VII.

- Variable locale : i : indice de boucle

- Algorithme :

ANALYSE_CARACTERES_IA5

```
deb_zone
si valid2(5) alors [param = 2
                    cause = 1
                    retour
pour i = 1 → 4
     [lire
     negoS [ y]= carE ET carR
```

```
        y = y + 1
k = k + 1
retour.
```

1.1.3.4. Routine ANALYSE_SEQUENCES.
****

- Définition : elle analyse la zone "séquences de contrôle" en suivant les règles énoncées au chapitre VII.

- Variable locale : /

- Algorithme :

ANALYSE_SEQUENCES.

```
deb_zone
si valid2 (18) alors [param = 2
                      cause = 1
                      retour
trt_fonction (17)
k = k + 1
retour
```

1.1.3.4.1. Routine TRT_FONCT (j).
==========

- Définition : elle analyse j bytes dans les vecteurs negoE et negoR en suivant les règles énoncées précédemment et crée le vecteur negoS.

- Variables locales :

i,m : indices de boucles
l : position des bits traitée dans un byte
j : variable indiquant le nombre de bytes à analyser.

- Algorithme :

TRT_FONCT(j)

```
pour i = 1 → j
    lire
    l = 0
    pour m = 1 → 4
        sol1 = extraire (l,l+1, carE)
        sol2 = extraire (l,l+1,carR)
        si sol1 =
            0 : mettre (l,l+1,sol1,carS)
            1 : si sol2 ≤ 2  alors mettre (l,l+1,0,carS)
                             sinon mettre (l,l+1,1,carS)
            2 : si sol2 ≤ 2  alors mettre (l,l+1,0,carS)
                             sinon mettre (l,l+1,2,carS)
            3 : mettre (l,l+1,sol2,carS)
        l=l+2
    negoS [y] = carS
    y = y + 1
retour
```

1.1.3.5. Routine ANALYSE_TRANSMISSION.
****

- Définition : elle analyse la zone de transmission dans les vecteurs de négociation en suivant les règles énoncées au chapitre VII

- Variable locale :

- Algorithme :

```
ANALYSE_TRANSMISSION
┌─ deb_zone
│  si valid2(4) alors ┌param = 2
│                     │cause = 1
│                     └retour
│
│  trt_fonct(3)
│  k = k + 1
│  param = 3
└─ retour
```

1.1.3.6. Routine ANALYSE_JEUX_CARACTERES.
****

- Définition : elle analyse la zone "jeux de caractères" dans les vecteurs de négociation en suivant les règles énoncées au chapitre VII

- Variables locales : i : indice de boucle
  u : variable intermédiaire.

- Algorithme.

ANALYSE_JEUX_CARACTERES.

```
┌─
│  deb_zone
│  si valid2(3)  alors ┌param = 2
│                      │cause = 1
│                      └retour
│
│  pour i = 1 ⟶ 2
│       ┌─
│       │ lire
│       │ negoS [y] = carE OU carR
│       │ y = y + 1
│       │ si nature = 0 ┌alors u = carE
│       │               │
│       │               └sinon u = carR
│       │
│       │ jeu [i]= negoS [y - 1] XOR u
│       │ si jeu [i]≠ 0 alors mettre (2,2,1,status_remul)
│       └─
│  k = k + 1
└─ retour.
```

1.1.3.7. Routine ANALYSE_TOUCHES.

- Définition : elle analyse la zone "touches-fonctions" dans les vecteurs de négociation en suivant les règles énoncées au chapitre VII

- Variable locale : /

- Algorithme.

ANALYSE_TOUCHES

```
deb-zone
si valid(2) alors [param = 2
                   cause = 1
                   retour

lire
sol1 = extraire (7,7,carE)
sol2=  extraire (7,7,carR)
sol = sol1 ET sol2
si type_applic ≠ 5 ET sol1 =  1 ET sol2= 0 alors [param = 2
                                                  cause = 2
                                                  retour

mettre (7,7,sol,carS)
trt_OU (6,6)
si sol1= 0 ET sol2= 0 alors retour
si test_sol alors mettre (1,1,1,status_emul)
sol1 = extraire (0,5,carE)
sol2 = extraire (0,5,carR)
si type_applic ≠ 5 [alors si sol1 > sol2 [alors [param = 2
                                                Cause = 2
                                                retour
                                           sinon  mettre (0,5,sol1,carS
                    sinon si sol1 > sol2 [alors  mettre(0,5,sol2,carS)
                                          sinon  mettre(0,5,sol1,carS)
```

```
k = k + 1
negoS [y] = carS
y = y + 1
retour
```

1.1.3.8. Routine ANALYSE_ZONES.
****

- Définition : elle analyse la zone "zone-protégée" dans les vecteurs de négociation en suivant les règles énoncées au chapitre VII

- variable locale : /

- Algorithme.

ANALYSE_ZONES

```
deb_zone
si valid2(3) alors [param = 2
                    cause = 1
                    retour

si mode = 1 alors  [param = 2
                    cause = 1
                    retour
lire
trt_OU (0,0)
trt_OU (1,1)
negoS [y] = carS
y = y + 1
lire
trt_OU (0,6)
trt _ET (7,7)
negoS (y) = carS
y = y + 1
si type_applic ≠ 5 ET sol = 1 alors [param = 2
                                    cause = 2
                                    retour

k = k + 1
retour.
```

1.2. Algorithme gérant la primitive RDCLP.

* Définition :

Il réalise l'analyse des paramètres reçus et ceux présents dans la station. En fonction du résultat, il active les primitives adéquates.

* Variables globales.

i : indice de boucle
erreur : variable booléenne indiquant une erreur.
carE, carR : derniers caractères lus dans les vecteurs respectifs negoE et negoR
compt_renv : variable indiquant le nombre de tentatives de négociation ayant échoué.
prim : variable indiquant au moniteur de la station VTP le numéro de la primitive à envoyer ou du traitement à réaliser.
paramètreR : tableau contenant les paramètres de la primitives RDCLP reçue.

* Algorithme.

```
negoE = negoS
negoR = paramètreR
i = 0
erreur = vrai
tant que i < 35 OU erreur = faux
    [ lire
    | si carE ≠ carR alors erreur = faux
    | i = i + 1
si erreur = faux alors si compt_renv > 2 alors [prim=8C
                                                sinon [prim=80
                                                       compt_renv = +1
retour
```

1.3. Algorithme gérant la primitive RDCLN.

* Définition :

Il réalise l'analyse du paramètre reçu et, en fonction de cela, il réalise un traitement.

* Variables globales :

cause : cette variable indique la cause de l'erreur lors de l'analyse du vecteur de négociation.

compt_renv : variable indiquant le nombre de tentatives de négociation ayant échoué.

prim : variable indiquant au moniteur de la station VTP le numéro de la primitive à envoyer ou du traitement à réaliser.

paramètreR : tableau contenant le paramètre de la primitive RDCLN reçue.

* Algorithme.

```
cause = paramètreR
si cause=
        1 : si compt_renv > 2 alors [ prim = 8C
                              sinon [ prim = 80
                                    [ compt_renv = + 1
        2 : prim= 8C
        0,3 → 255 : prim= 8C
retour
```

ALGORITHMES DES MODULES DE L'EMULATEUR.

1. ALGORITHME DU MODULE "PLIAGE".

A) Liste des variables.

B) Algorithmes

```
lire (car)
depcur X := + 1
si depcur X > MVD - MVG alors

      envoyer (CR,LF)
      depcur X = 0
      depcur Y = + 1
      Si status_remul = saut_page
         alors  trt_saut-de-page
         sinon  si (depcurY = maxligT et block mode)
                                alors depcur Y = 0

                si (depcurY = maxligT et conv mode)
                                alors depcur Y = maxligT
retour
```

2. ALGORITHME DU MODULE "SAUT DE PAGE".

A) Liste des variables.

B) Algorithme.

trt_saut_de_page

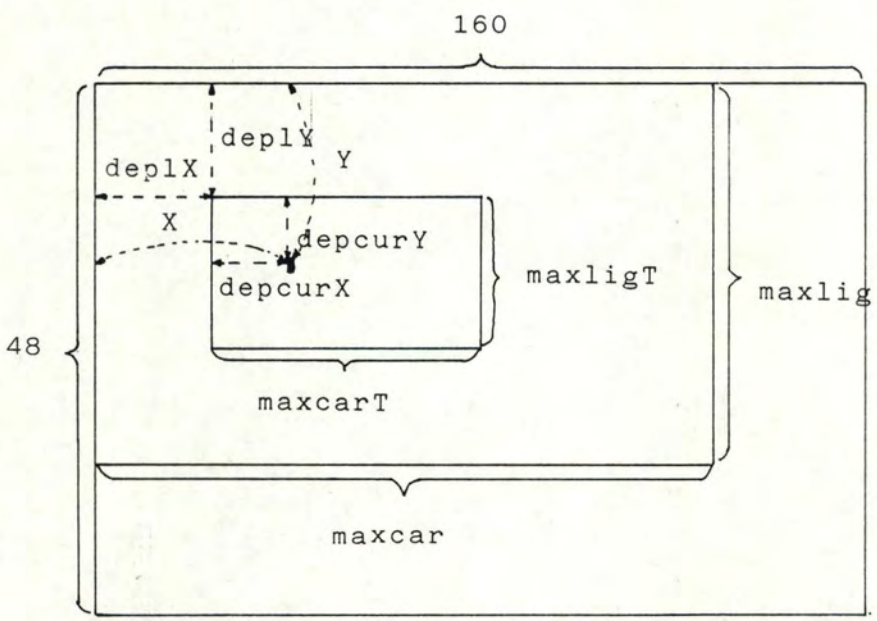
```
si depcur y > MHS-MHI  alors  envoyer (FF)
                              depcur  X = 0
                              depcur  Y = 0

retour
```

3. ALGORITHME DU MODULE "GESTION-ECRAN".

3.1. Interprétation des touches de commande.

liste des variables.

deplY } variables contenant la position du coin supérieur
deplX } gauche de l'écran réel par rapport à la position [0,0] de la matrice utile.



Algorithme.

```
trt_touches_commande

si touche =
    ctrl A : si deplY > 0  alors [ deplY = -1
                                   TRT_ affichage

    ctrl B : si deplY < maxlig alors [ deplY = +1
                                       TRT_affichage
```

```
        ctrl C : si deplX > 0  alors [deplX = -1
                                      TRT_ affichage

        ctrl D : si deplX < maxcar alors [deplX = +1
                                          TRT_ affichage
retour
```

3.2. Affichage de la fenêtre.

Liste des variables.

nbre_lig, nbre_car : variables indiquant le nombre de lignes (caractères) courant.

Algorithme.

trt_affichage

```
pour nbre_lig = 0 → maxligT
    pour nbre_car = 0 → maxcarT
        envoyer (écran [deplY + nbre_lig, deplX + nbre_car])
        trt_seq_attribut
retour
```

3.3. Mémorisation d'une page.

Liste des variables.

i,j : variables indiquant la position du caractère courant.
car : variable contenant le caractère à traiter.

Algorithme.

trt_mémoire_page

```
blocage clavier
pour i = 0 → maxlig
    pour j = 0 → maxcar
        lire (car)
        ECRAN (i,j)
déblocage clavier
retour.
```

3.4. Mise à jour du curseur fictif.

Liste des variables.

Algorithme.

```
trt_Nécran (X,Y)
si X > maxcar alors si STATUS_MODE=autowrap alors X = -maxcar + X
                                                  Y = Y + 1
                                            sinon X = maxcar
si Y > maxlig alors  Y = maxlig
si X < 0 alors X = 0
si Y < 0 alors Y = 0
si X > maxcarT + deplX alors deplX = X-maxcarT-5
                             depcurX = maxligT- 5
si Y > maxligT + DeplY  alors deplY = Y - maxligT
                              depcur Y = maxligT
si X < deplX  alors deplX = X
                    depcur X = 0
si Y < deplY  alors deplY = Y
                    depcur Y = 0
envoyer (ESC,depcurX,depcurY)
trt _ afffichage
retour
```

Routine.

trt_seq_attribut : envoie la séquence de contrôle contenant l'attribut de visualisation des prochains caractères.

trt_seq_attribut_

```
couleur = ATTRIB [X,Y]
si ATTRIBUT_CURSEUR = Couleur alors [retour
ATTRIBUT_CURSEUR = couleur
envoyer (ESC, [ , 0)
pour i = 0→7
     [bool = extraire (i,i,couleur)
     si bool = 1 alors [si i =
                          [0 : envoyer (;,0)
                          1 ; envoyer (;,1)
                          2 : envoyer (;,2)
                          3 : envoyer (;,3)
                          4 : envoyer (;,4)
                          5 : envoyer (;,5)
                          6 : envoyer (;,7)
                          7 ; envoyer (;,9)
envoyer (m)
retour
```

4. ALGORITHMES DU MODULE "TRAITEMENT DE TEXTE".

4.1. Le coordinateur.

Liste des variables.

pligne : variable indiquant le numéro de la ligne à analyser dans la matrice ECRAN

dligne : variable indiquant le numéro de la dernière ligne mémorisée dans ECRAN.

Ces deux variables travaillent en modulo 48.

résultat : variable reprenant la différence entre la ligne analysée et la dernière introduite.

i, u : variables désignant le numéro de la ligne du tableau qui sert lors de l'analyse et du traitement du du texte.

Algorithme

```
i = 0
pligne = 0
u = 1
trt_analyse (i)
tant que status_ligne ≠ 0
    si dligne < pligne alors résultat = dligne + 48 - pligne
                       sinon résultat = dligne - pligne
    si résultat < 2 alors WAIT
    trt_analyse (u)
    trt_phrase (i)
    si i = 1 alors u = 1
                   i = 0
             sinon i = 1
                   u = 0
    pligne = (pligne + 1) mod (48)
retour
```

4.2. L'analyse de la ligne.

Liste des variables.

car : dernier caractère lu dans TABLEAU-LIGNE.

nbre_car : nombre de caractères déjà lu dans la ligne.

nbre_blanc : nombre de blancs consécutifs.

pcar : variable mémorisant le caractère précédent celui qu'on vient de lire.

dcar : variable donnant le dernier caractère différent d'un blanc dans la ligne.

adresscar : variable mémorisant l'adresse du dernier caractère non blanc.

carmot : variable donnant le nombre de caractères dans le mot courant.

tab : variable composée :

- car/mot : donne le nombre de caractères dans le mot.
- adresse : donne l'adresse du début du mot dans la ligne.
- type : donne la nature du mot :
  - 0 : dernier élément de la file,
  - 1 : caractère blanc,
  - 2 : caractère différent d'un blanc.

nbre_mot : variable donnant le nombre de mots dans une ligne.

gr_blanc : variable indiquant qu 'il y a un grand blanc à l'intérieur de la ligne.

longblanc : variable donnant la longueur du blanc en début de ligne.

souligne : variable désignant que le type de la ligne est "souligné"

carsoulig : variable donnant le code du caractère souligné.

a : variable indiquant le numéro de la ligne dans TABLEAU LIGNE et dans toutes les variables indicées.

j : variable indiquant qu'on lit le premier blanc dans une ligne ou qu'on se trouve dans un grand blanc au milieu de la ligne.

Algorithme.

trt_analyse (a)

```
init
tant que car ≠ CR OU nbre_car ≠ maxcar
    si car = blanc  alors  si nbre_mot = 0 alors  nbre_blanc =+1
                                                  j = 1
                           sinon                  trt_blanc
                   sinon   si pcar = blanc alors  trt_grand_blanc
                           sinon                  trt_normal
                           dcar [a] = car
                           adresscar [a] = nbre_car
                           carmot = +1
    nbre car   = +1
    pcar = car
    lire (car)
si pcar = blanc alors  tab_car/mot = carmot
                       poser-pile (a,tab)
trt_type
retour
```

Routine.

- init

```
car = 0
pcar = 0
nbre_car = 0
nbre_mot = 0
nbre_blanc = 0
carmot = 0
j = 0
gr_blanc [a] = 0
lire (car) ; retour
```

- trt_blanc

```
si pcar = blanc  alors nbre_blanc = +1
                       si nbre_blanc ⩾ 5  alors j = 1

                  sinon tab_car/mot = carmot +1
                        tab_type = 2
                        poser file (a,tab)
                        nbre_blanc = 0
                        carmot = 0

retour
```

- trt_grand_blanc

```
si j = 1 alors tab_car/mot = nbre_blanc
               tab_adresse = nbre_car - nbre_blanc
               tab_type = 1
               poser file (a,tab)
               j = 0
               si nbre_mot [a] = 0 alors long_blanc [a] =nbreblanc
                                   sinon grblanc [a] = 1
tab_adresse = nbre_car
nbre_mot [a] = + 1
souligne [a] = 0
retour
```

-trt-normal-

```
si pcar = car  alors [car_soulig [a] = car
                     [souligne [a] = 1
               sinon [souligne [a] = 0
retour
```

- lire (car)

```
car = TABLEAU_LIGNE [a,nbrecar]
retour
```

-trt_type (a)

```
si nbre_mot [a] = 0  alors  suiV [a] = 0;ligne blanche
si grblanc [a] = 1   alors  suiV [a] = 1; entête
si souligne [a] = 1  alors  suiV [a] = 2; souligné
si dcar [a] = ":"    alors  suiV [a] = 4; début énumération
si genre [o] ≠ normal  alors suiV [à] = 8; ligne dans une énu-
                                          mération
                       sinon suiV [a] = 16; ligne normale
si nbrecar ≠ maxcar alors index = 1
retour
```

4.3. Le traitement de la ligne.

liste des variables.

index : variable indiquant qu'on est en fin de paragraphe ou en fin de titre ou d'énumération.

longueur : variable donnant la longueur utile de la ligne sur le support de sortie.

suiV : variable indiquant la nature de la ligne analysée ou traitée.

nature : variable indiquant si on se trouve dans une ligne normale ou une ligne dans une énumération.

nbre_mot : variable donnant le nombre de mots dans une ligne.

TAB : variable composée (voir 8.5.4.2.4.)

TBL, TABL, TABL_blanc : idem TAB

i, r, s, : pointeur indiquant dans quelle file on travaille.

a : indique le n° de la ligne dans tableau-ligne utilisé pour la ligne traitée.

u : idem "a" mais pour la ligne analysée.

tot_car : variable indiquant le nombre de caractère dans la partie droite d'une entête.

max : variable mémorisant le tot_car maximum rencontré pour cette entête.

A : complément de "max" par rapport à 'longueur'.

niV : variable indiquant le niveau d'encastrement des énumérations.

flag : variable indiquant qu'on doit mettre à jour la longueur du blanc du début de ligne.

blanc_long [ ] : variable mémorisant pour chaque niV, la grandeur du blanc du début de ligne.

carsoulig : variable donnant le code du caractère souligné.

adresse-car [ ] : variable mémorisant l'adresse du dernier caractère non-blanc.

compteur : variable donnant le nombre de caractères dans la ligne traitée.

FAC : facteur de réduction de la ligne égal à

$$\frac{\text{MAXCART}}{\text{MAXCAR}}$$

genre : variable indiquant si un niV[i] est normal ou une énumération.

p,q : variables indiquant le nombre de blanc à ajouter entre les mots :
- p depuis le début de la ligne → h
- q depuis h → la fin de la ligne.

h : variable indiquant l'endroit dans la ligne où il faut passer de p→q

pair : variable indiquant si on traite une ligne paire ou impaire.

ajoutblanc : variable indiquant le nombre de blancs total à ajouter dans la phrase pour réaliser la justification.

incr : position courante dans la ligne permettant de voir s'il faut passer de p→q

algorithmes

trt_phrase.

```
longueur = maxcarT - ((MVG) + (MVD) * FAC)
si suiV [ u ] = 0 OU long_blanc [ u ] ≠ 0
                alors index = 1
                sinon index = 0
si suiV [ u ] = 2  alors [suiV [ a ] = 32
si suiV [ a ] =
    0 : trt_ligne_blanche
    1 : trt_entête
    4 : trt_début_énumération
    8 : trt_énumération
   16 : trt_paragraphe
   32 : trt_titre_souligné
```

```
    2 : trt_souligné
retour
```

les routines.

* trt_ligne_blanche

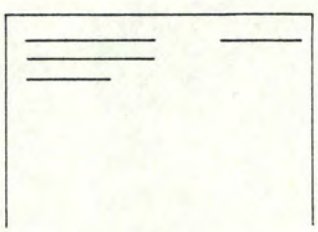
Cette routine réalise le traitement d'une ligne blanche.

```
si nature = énumération alors [trt_fin_énumération
si genre [ o ] = normal alors [nature = normal
envoyer (CR,LF)
retour
```

* trt_entête

Cette routine procède de la manière suivante :

- envoie les mots à gauche du grand blanc,
- mémorise dans une file les mots à droite du grand blanc,
- si la ligne suivante est différente d'une ligne d'entête, on vide la file en envoyant toutes les parties de droites des lignes précédemment enregistrées.



```
nbre_mot = 0
tirer_file (i,TAB)
tant que non (nbre_mot ≠ 0 ET TAB_type = 1)
```

```
trt_envoie
    pour k = 0 → TAB_car/mot - 1
                [envoyer (TABLEAU_LIGNE [a, TAB_adresse +k ] )
    envoyer (blanc)
    tirer_file (i,TAB)
    nbre_mot = +1

tirer_file (i,TAB)
TBL_type = 255
TBL_adresse = 0
TBL_car/mot = 0
poser_file (r,TBL)
tant que TAB_type ≠ 0
    [poser_file (r,TBL)
    tot_car = tot_car + TAB_car/mot + 1
    tirer_file (i,TAB)

si max < tot_car alors [max= tot_car
si suiV [u] ≠ entête alors
                        A = longueur-max
                        tant que TBL_type ≠ 0
                            [pour i = 0 → A - 1
                                [envoyer (blanc)
                            tirer file (r,TBL)
                            tant que TBL_type ≠ 255
                                [trt_envoie
                            envoyer (CR,LF)

retour
```

* trt_début_énumération

Cette routine traite une ligne se terminant par ":"
comme suit : - traite la ligne comme une ligne normale,

- met à jour un certain nombre de variables pour indiquer qu'on est dans une énumération et le niveau d'encastrement.

```
trt_paragraphe
genre [niV] = énumération
nature = énumération
flag = 1
niV = niV + 1
retour
```

* trt_énumération

Cette routine travaille comme suit :

- vérifie qu'on ne change pas de niveau d'énumération, c'est-à-dire une énumération dans une autre
- justifie la ligne
- si on est en fin de paragraphe, vérifie qu'il faut ou non justifier ce qui reste
- vérifie qu'il faut descendre d'un niveau si la ligne suivante est blanche et que le blanc en début de ligne est différent du paragraphe précédent.

```
tirer (i,TAB)
si flag = 1 alors [ compteur = + TAB__car/mot * FAC
                    blanclong [niV] = TAB__car/mot
                    TABL_blanc = TAB
                    poser-file (s,TABL_blanc)
                    flag = 0
                  ]
```

```
trt_lecture
tant que TAB_type ≠ 0
    [ trt_rangement
    [
trt_fin_paragraphe
trt_fin_énumération
retour
```

* trt_paragraphe

Cette routine traite toutes les lignes de type différents d'entête ligne-blanche, énumération et titre souligné.

```
trt_une_ligne
trt_fin_paragraphe
retour
```

* trt_titre_souligné

Cette routine travaille comme suit :

- envoie d'une ligne de titre de longueur "longueur"
- souligne le titre
- recommence jusqu'à la fin du texte.

```
trt_lecture
tant que TAB_type ≠ 0
    [ trt_rangement
    [ si TAB_type ≠ 0 alors [ pour j = 0 → longblanc [ i ]
                            [     [ envoyer (blanc)
                            [
                            [ pour j = longblanc → longueur
                            [     [ envoyer (carsoulig)
                            [
```

```
si index = 1 alors [trt_fin_paragraphe
                    pour i = 0 → adresse-car [ i ]
                        [envoyer (carsoulig)
retour
```

* trt_lecture

Cette routine met à jour le nombre de caractères déjà lu dans la ligne courante.

```
tirer (i,TAB)
si TAB_type = 1 alors compteur = + TAB__car/mot * FAC
si TAB_type = 2 alors compteur = + TAB__car/mot
retour
```

* trt_fin_énumération

Cette routine vérifie et réalise le passage d'un niveau au précédent dans une énumération.

```
si long_blanc [ u ] ≠ blanc_long [niV]
     alors [tant que long_blanc [ u ] ≠ blanc_long [niV]
                    [genre [niV] = normal
                    [niV = niV - 1
            flag = 1
si genre [ 0 ] = normal  alors [nature = normal
retour
```

* trt_souligné

```
[retour
```

* trt_rangement

Cette routine prend dans la file TAB suffisamment de mots pour reformer une ligne sur le support de réception et les range dans une nouvelle file.

Pour une ligne complète, elle active différentes routines permettant de reformater et justifier le texte.

```
tant que compteur ≤ longueur OU TAB_Type ≠ 0
    poser_file (s,TABL)
    nbre_mot = + 1
    trt_lecture
si TAB_type ≠ 0 alors  trt_calcul
                       trt_émission
                       compteur = 0
                       si nature = énumération alors
                                   TABL = TABL _blanc
                                   poser (s, TABL)
                                   compteur = TAB-car/mot * FAC
retour
```

* trt-fin-paragraphe

Cette routine vérifie qu'il faut ou non justifier le reste de la ligne.

```
si index = 1 alors  si nbre_car > o,7 * longueur
                           alors  trt_calcul
```

```
                        sinon [ p = 1
                              [ q = 1
                              [ h = longueur/2

                trt_émission
                compteur = 0

retour
```

* trt_une ligne

Cette routine réalise le reformatage de la ligne en partant du bout de ligne précédente et ce jusqu'à ne plus pouvoir créer une ligne entière avec les mots restants.

```
trt_lecture
tant que TAB_type ≠ 0

     [trt_rangement

retour
```

* trt_calcul

Cette routine calcule le nombre de blancs à ajouter entre les mots pour la justification. Il distingue les lignes paires et impaires pour éviter de toujours ajouter, en début de lignes, les plus grands blancs entre les mots.

Pour les lignes paires p > q

Pour les lignes impaires q > p

```
ajoutblanc = maxcarT - compteur
si pair = 1  alors [ pair = 0
                   [ q = ajout_blanc div (nbre_mot-1)
                   [ h = ajout_blanc + 1 - q *(nbre_mot - 1)
                   [ p = q + 1
```

```
                sinon [pair = 1
                       p = ajoutblanc div (nbre_mot - 1)
                       h = p * (nbre_mot - 1) + (nbre_mot-ajoutblanc)
                       q = p + 1
retour
```

* trt émission (s, p, q, h,)

```
tirer (s,TABL)
incr = 0
tant que TABL_type ≠ 0
    si TABL_type ≠ 1 alors [pour k = 0 ⟶ TABL_car/mot - 1
                            envoyer (TABLEAU LIGNE [a,TABL-adresse
                                                    +k ]
                          sinon [pour k = 0 ⟶ (FAC * TABL-car/mot) -1
                                 envoyer (blanc)
    si incr < p * (h-1) alors [pour k = 0 ⟶ p
                                   [envoyer (blanc)
                               incr = + p
                          sinon [pour k = 0 ⟶ q
                                   [envoyer (blanc)
                               incr = + q
retour
```

* tirer_file

  Cette routine retire un élément d'une file et la supprime de la chaîne.

* poser_file

  routine inverse de la précédente.

* envoyer

  routine qui envoie les caractères passés en paramètres.

5. ALGORITHMES DU MODULE "TOUCHES-FONCTIONS".

5.1. L'enregistrement du contenu des touches-fonctions.

liste des variables.

nfct : numéro de touche fonction courante.

ncar : nombre de caractères courant dans la zone PFi.

point : variable indiquant le début de la zone réservée à la touche fonction courante.

k : variable pointant la première position réservée à la touche fonction courante et mémorisant le nombre de caractère utile dans la zone.

car : caractère courant.

erreur1, erreur2 : variable booléenne indiquant une fin prématurée dans l'analyse d'une touche PFi.

maxnfct: variable obtenue lors de la négociation et qui indique le nombre de touche utilisable.

let [1...64] : tableau contenant les 64 lettres à mettre dans la séquence de contrôle par défaut.

Algorithme.

trt_enregistrement_touches-fonctions

```
nfct = 0
ncar = 0
point = nfct * 11 + 1
k = point - 1
erreur1 = erreur2 = vrai
tant que nfct < maxnfct OU erreur1 = vrai
```

```
    lire (car)
    tant que car ≠ ";" OU erreur1 = vrai
        ncar = + 1
        point = point + 1
        FONCT [point] = car
        si car = "*" ET ncar = 1
                    alors  FONCT [ 1] = "ESC"
                           FONCT [ 2] = "0"
                           FONCT [ 3] = let [nfct]
                           ncar = 3
                           erreur2 = faux
        si ncar > 10 alors [erreur1 = faux
        lire (car)
        si  erreur2 = faux ET car ≠ ";" alors [erreur1 = faux
                                       sinon [erreur2 = vrai
    si ncar ≠ 0 alors FONCT  [ k] = ncar
    nfct = + 1
    ncar = 0
    point = nfct * 11 + 1
    k = point - 1
retour
```

5.2. La lecture du contenu des touches-fonctions.

liste des variables

nb : variable indiquant le numéro de la touche enfoncée.

index : variable servant d'indice pour le tableau F et qui pointe vers l'élément à transférer.

car : variable contenant la lettre correspondant à la touche enfoncée.

let [1...64] : tableau contenant les 64 lettres à mettre dans la séquence de contrôle par défaut.

maxfct : variable obtenue lors de la négociation et qui indique le nombre de touches fonctions utilisables.

Algorithme.

trt_lecture_touches_fonctions (car)

```
[ nb = 0
  tant que car ≠ let [nb]                    ou nb = car - 65d
        [ nb = nb + 1 ]                            |||
                                                   A en ASCII
  si nb> maxfct alors [retour
  index = nb * 11
  pour i = index + 1 ⟶ index + FONCT [index]
        [ envoyer (FONCT [i]) ]
  retour ]
```

6. ALGORITHME DU MODULE "JEU DE CARACTERES".

liste des variables

JEU1, JEU 2... JEU9 : tableaux contenant tous les caractères ASCII N°5 converti en ASCII U.S.A. qui se trouve sur tous les terminaux en standard.

car : variable contenant le code du caractère à convertir. Il va servir d'index pour la conversion.

Algorithme.

trt_jeu-caractères (car,LANG)

```
si LANG =
      1 : envoyer (JEU1 [car] )
      2 : envoyer (JEU2 [car] )
      4 : envoyer (JEU3 [car] )
      8 : si car = 126 alors envoyer (s)
          envoyer (JEU4 [car] )
     16 :        ¦
                 ¦
     32 : si car = 91  alors envoyer (A)
          si car = 123 alors envoyer (a)
          envoyer (JEU6 [car] )
     64 :        ¦
                 ¦
                 ¦
    128 : envoyer (JEU8 [car] )
retour
```

N.B. : Voici la liste pour chaque langue des codes à modifier et leur conversion.

JEU 1 English

caractère	code	caractère	code
£	35		35

JEU2 Espagnol

caractère	code	caractère	code

JEU3 Français

caractère	code	caractère	code
à	64	a	97
â	91	a	97
ç	92	c	99
ê	93	e	101
î	94	i	105
ô	96	o	111
é	123	e	101
ù	124	u	117
è	125	e	101
û	126	u	117

JEU4 Allemand.

caractère	code	caractère	code
Ä	91	A	65
Ö	92	O	79
Ü	93	U	85
ä	123	a	97
ö	124	o	111
ü	125	u	117
ß	126	ss	115

JEU5 Italien

caractère	code	caractère	code

JEU6 . Danois/Norvégien

caractère	code	caractère	code
Ä	64	A	65
AE	91	AE	65,69
Ø	92	O	79
Å	93	A	65
ü	94	U	85
ä	96	a	97
ae	123	ae	97,101
ø	124	o	111
å	125	a	97
ü	126	u	117

JEU7 Suédois/Finlandais

caractère	code	caractère	code
É	64	E	69
Ä	91	A	65
Ö	92	O	79
Å	93	A	65
Ü	94	U	85
é	96	e	101
ä	123	a	97
ö	124	o	111
å	125	a	97
ü	126	u	117

7. ALGORITHMES DU MODULE "ZONES FORMATEES".

7.1. L'enregistrement des zones formatées.

Liste des variables.

pos : variable indiquant la position courante dans la zone. Cette position correspond à une adresse absolue.

car : variable contenant le dernier caractère introduit.

i,j : indice de boucle.

set_alpha : ensemble contenant toutes les lettres de l'alphabet : majuscules et minuscules qui servent dans les séquences de contrôle.

attribut : variable mémorisant la valeur courante de l'attribut de visualisation lorsqu'on est en train de la modifier.

Algorithme

trt_enregistrement-zone

```
envoyer (ESC, [,2,J)
pour i = 0 ⟶ maxcar - 1
    pour j = 0 ⟶ maxlig - 1
        ECRAN [j,i] = 0
        ATTRIB [j,i] = 0
ZONE_POSMIN = 0
ZONE_POSMAX = 160*48
ZONE_TYPE = 32 + 1 (non-protégé et alphanumérique)
pos = - 1
i = 0
lire (car)
```

```
tant que car ≠ ESC
    [ si car ≠ DC1 alors [ZONE_POSMIN = pos + 1
                  sinon [ lire (A)
                          lire (B)
                          ZONE_POSMIN = A + B*160
                          si A ≤ maxcarT ET B ≤ maxligT
                             alors [envoyer (ESC, [,A,;,B,s )
      lire (car)
      si car ≠ DLE  alors [ZONE_TYPE = 32 + 1
                     sinon [lire (ZONE_TYPE)
      si ZONE_TYPE ≠ protégé alors [TAB_PROTEGE [i] = ZONE_POSMIN
                                    i = i + 1
      lire (car)
      tant que car ≠ DC3
              [ trt_ESC
                ECRAN [X,Y] = CAR
                ATTRIB [X,Y] = attribcurseur
                trt_avance-curseur
                si X ≤ maxcarT ET Y ≤ maxligT alors [envoyer (car)
                lire (car)
      ZONE_POSMAX = pos
      poser-file (zone)
    ]
test (5,[ ? 17 1 )
retour
```

Routines

+ trt_ESC

Cette routine analyse tous les caractères et elle supprime toutes les séquences de contrôle à l'exception de celles de visualisation et les caractères de contrôle.

```
si car =
   o...31 : lire (car)
   27(ESC) : lire (car)
             si car =
                [ : lire (car)
                    tant que car ∉ set_alpha
                                si car ≠ ";" alors mettre-pile (car
                                lire (car)
                    si car = m alors trt_visu
                # : lire (car)
                    ECRAN [159,Y] = car
                ( : lire (car)
                ) : lire (car)
             lire (car)
retour
```

+ trt_avance-curseur

Cette routine met à jour la position du curseur fictif.

```
si X + 1 > maxcar  alors  X = 0
                          Y = Y + 1
                          poscur = Y * 160

                   sinon  X = X + 1
                          poscur = X + Y * 160

si Y > maxlig alors  Y = 0
                     X = 0
                     poscur = 0

retour
```

+ Poser-file

cette routine permet de mettre une variable dans une file en réalisant le chaînage avec l'élément précédent et l'élément suivant.

+ Test

Cette routine permet de comparer les caractères passer en paramètres et ceux retirés d'un buffer.

+ Mettre-pile

Cette routine permet de déposer dans une pile le caractère passer en paramètre.

+ trt-visu

Cette routine permet de mettre à jour l'attribut de visualisation du curseur et par le fait même, changer l'attribut de visualisation de tous les caractères qui seront pointé par le curseur.

trt-visu

```
attribut = attribut_curseur
tant que pile ≠ vide
    tirer-pile (car)
    si car = 0 alors attribut = 0
    mettre (car, car, 1, attribut)
attribut curseur = attribut
```

+ tirer-pile

cette routine permet de retirer un élément de la pile et l'assigner au paramètre.

7.2. Le traitement de caractère.

Liste des variables

set-alphabet : ensemble contenant toutes les lettres de l'alphabet minuscule et majuscule.

set_numérique : ensemble contenant les caractères suivants: 0,1,2,3,4,5,6,7,8,9,+,-,*,/,.,(,),",'

set_digit : ensemble contenant les caractères suivants : 0,1,2,3,4,5,6,7,8,9

set_alphanum : ensemble contenant tous les caractères ASCII moins les caractères de contrôle (0→31)

poscur : variable donnant la position du curseur en absolu.

valable : variable booléenne indiquant qu'un caractère est valable (=vrai) ou non (=faux).

Algorithme

trt-zones-formatées

```
poscur = X + 160 * Y
valable  = faux
si poscur < ZONE_POSMIN OU poscur > ZONE_POSMAX
          alors trt_Saut-zone (poscur)
si ZONE_TYPE =
   num : si car ∈ set_numerique alors
                    alors si ZONE_TYPE = non-visible
                             alors car = car AND 80 H
                          1
                          valable = vrai
                          ECRAN [X,Y] = car
                          ATTRIB [X,Y] = attribut-curseur
   alpha_num : si car ∈ set_alphnum alors  1
   digit : si car ∈ set_digit  alors  1
   alphabet : si car ∈ set_alphabet alors  1
si valable = vrai alors si poscur + 1 ≤ ZONE_POSMAX
                        alors trt_avance-curseur (poscur)
                        sinon trt_saut-zone-non-protégée
                                                  (poscur+1)
trt_Necran
ret cur
```

Routines

+ trt-saut-zone

Cette routine permet, si le curseur sort d'une zone non protégée ou s'il est placé dans une zone protégée, d'accéder à la zone suivante non-protégée.

```
si poscur < ZONE_POSMIN
   alors [tant que poscur < ZONE_POSMIN
                  [tirer - preced - file (zone)
   sinon [tant que poscur > ZONE_POSMAX
                  [tirer - suiv - file (zone)
si ZONE_TYPE = protégé
   alors [tant que ZONE_TYPE = protégé
                  [tirer-suiv-file (zone)
Y = ZONE_POSMIN div 160
X = ZONE_POSMIN mod 160
poscur = ZONE_POSMIN
retour
```

+ tirer-précéd-file
+ tirer-suiv-file

Ces routines servent à extraire d'une file une variable : soit celle qui précéde, soit celle qui suit.

+ trt-avance-curseur

  voir 8.9.1.4.C

+ trt-Necran

  voir 8.4.5.4.B

+ trt-saut zone non-protégée

```
si status_mode = auto tab  alors
    tant que ZONE_TYPE = protégé
            tirer_suiv_file (zone)
    Y = ZONE_POSMIN div  160
    X = ZONE_POSMIN mod  160
    poscur = ZONE_POSMIN
retour
```

Liste des variables :

PTEUR : pointeur qui parcourt le vecteur.
séquence : booléen qui indique si on se trouve ou pas dans une séquence.
param : booléen qui indique que le caractère suivant peut être paramètre.
début_vect : pointeur sur le début du vecteur émetteur-récepteur.
début_zone : pointeur sur le début de la zone ou s'effectue la recherche.
trouve : booléen qui indique le caractère n'est pas dans la zone courante

Algorithme :

```
Exploit_vect_séq.
initialisation
tant que   des caractères faire
        si séquence  = faire alors  I = 1
                                    début_zone = début_vect

        recherche_car_dans_niveau (I)
        si trouve = truc alors
            si I = 1 alors séquence = truc
            trt_qualif
        sinon
            si séquence = faire alors excécute_rout (index)
            sinon
                si param = false alors trt_erreur
                sinon trt_param
```

```
Initialisation

   pile à vide
   séquence = false
   param = false
   en fonction de l'état du terminal (récepteur-émetteur),
   positionner début_vect sur le vecteur émetteur ou récepteur

          sélection des buffers origines des caractères

   lire caractère

recherche_car_dabs niveau (I)

   PTEUR = début_zone
   trouve = false
   tant que trouve = false et (PTEUR)   fin_zone faire

         si caract_courant = (PTEUR) alors  trouve = truc
                                            param = false

         sinon
              avancer PTEUR sur caractère suivant

trt_qualif

   case qualif_car

        o : progression
      1,3 : param = truc
            progression
    4,6,10 : séquence = false
             execute_rout (index) (caractère de séquence)
             lire caractère
```

```
trt_erreur

   ⌈mise à jour de STATUS_ERREUR
   |purge
   ⌊

trt_param

   ⌈caract dans pile
   |modification des pointeurs de pile
   |lire caract
   ⌊

progression

   ⌈I = I + 1
   |début_zone = index
   |lire caract
   ⌊
```

A N N E X E IV.

Les caractères suivants sont affectés aux positions optionnelles du tableau 2/V.3:

#	Signe numéro	2/3
¤	Signe monétaire	2/4
@	a commercial	4/0
[	Crochet gauche	5/11
\	Barre de fraction renversée	5/12
]	Crochet droit	5/13
{	Accolade gauche	7/11
|	Barre verticale	7/12
}	Accolade droite	7/13

CCITT-4929

Il est à noter qu'aucune substitution n'est permise lors de l'emploi de la version internationale de référence.

6.3 *Versions nationales*

6.3.1 Les organismes nationaux de normalisation ont la responsabilité de la définition des versions nationales. Ces organismes doivent arrêter les choix qui sont ouverts en fonction des besoins.

6.3.2 En cas de nécessité, un pays peut définir plusieurs versions nationales. Ces différentes versions doivent être identifiées séparément. En particulier, si pour une position à usage national donnée, par exemple 5/12 ou 6/0, deux caractères sont nécessaires, deux versions différentes doivent être identifiées, même si elles ne diffèrent que par ce seul caractère.

6.3.3 Si un pays n'a pas besoin de caractère spécifique, il est vivement recommandé que les caractères de la version internationale de référence soient affectés respectivement aux mêmes positions à usage national.

6.4 *Versions d'application particulière*

Au niveau d'entreprises industrielles nationales ou internationales, d'organisations ou de groupes professionnels, il est possible d'utiliser des versions pour des applications particulières. Ces versions nécessitent un accord précis entre les parties intéressées pour arrêter les choix qui sont ouverts en fonction des besoins.

7. *Caractéristiques fonctionnelles des caractères de commande*

Certaines définitions du présent paragraphe sont exprimées en termes généraux et des définitions d'emploi plus précises peuvent être nécessaires pour des applications particulières des tableaux de codes sur des supports d'enregistrement ou sur des voies de transmission. Ces définitions plus précises font l'objet de publications de l'ISO.

7.1 *Dénominations générales des caractères de commande*

Les désignations générales des caractères de commande comportent une dénomination générique suivie d'un indice.

Ils sont définis comme suit:

TC – *Caractère de commande de transmission*

Caractère de commande destiné à commander ou à faciliter la transmission d'informations sur les réseaux de télécommunication.

L'utilisation des caractères TC sur les réseaux généraux de télécommunication fait l'objet de publications ISO.

Les caractères de commande de transmission sont:

ACK, DLE, ENQ, EOT, ETB, ETX, NAK, SOH, STX et SYN.

FE – *Commande de mise en page*

Caractère de commande qui a principalement pour objet de commander la disposition ou la mise en page de l'information sur une imprimante ou un récepteur visuel. Toute référence à une imprimante dans une définition de commande spécifique de mise en page doit être considérée comme applicable à un récepteur visuel. Les définitions de commande de mise en page emploient le concept suivant:

a) une page est composée d'un nombre défini de lignes de caractères;

b) les caractères formant une ligne occupent un nombre défini de positions appelées positions de caractère;

c) la position active est la position de caractère dans laquelle le caractère, sur le point d'être traité, apparaîtrait s'il était à imprimer. Normalement, la position active se déplace d'une position de caractère à la fois.

Les caractères de commande de mise en page sont:

BS, CR, FF, HT, LF et VT (voir également la remarque 1 relative au tableau 1/V.3).

DC – *Commande d'organe périphérique*

Caractères de commande destinés à la commande d'un ou plusieurs organes périphériques situés sur place ou éloignés et reliés à un système de traitement des données ou de télécommunication. Ces caractères de commande ne sont pas prévus pour commander des systèmes de télécommunication; ceci doit se faire par l'intermédiaire des TC.

Certains emplois préférentiels de DC particuliers sont donnés dans le paragraphes 7.2.

IS – *Séparateurs d'information*

Caractères de commande employés pour séparer et qualifier logiquement des données. Il en existe quatre. Ils peuvent être utilisés dans un ordre hiérarchique supérieur ou non hiérarchique. Dans le second cas, leur signification spécifique dépend de leur application.

S'ils sont utilisés hiérarchiquement, l'ordre croissant est:

US, RS, GS, FS.

Dans ce cas, les données, normalement délimitées par un séparateur particulier, ne peuvent être divisées par un séparateur d'un ordre hiérarchique supérieur mais seront considérées comme délimitées par un séparateur d'un ordre hiérarchique supérieur.

7.2 *Caractères de commande particuliers*

On désigne parfois des membres particuliers de classes de commande par le nom abrégé de la classe affecté d'un indice (par exemple, TC_5) ou encore par une dénomination particulière qui en indique l'emploi (par exemple, ENQ).

Des significations différentes mais apparentées peuvent être associées à certains caractères de commande, mais ceci exige normalement un accord entre l'émetteur des données et leur destinataire.

ACK – *Accusé de réception positif*

Caractère de commande de transmission transmis par un récepteur comme réponse affirmative à l'émetteur.

BEL – *Sonnerie*

Caractère utilisé lorsqu'il est nécessaire d'attirer l'attention; il peut commander des dispositifs d'appel ou d'avertissement.

BS – *Retour arrière*

Commande de mise en page qui ramène la position active en arrière d'une position de caractère sur la même ligne.

CAN – *Annulation*

Caractère ou premier caractère d'une suite de caractères indiquant que les données le précédant sont erronées et que ces données doivent être ignorées. Le sens spécifique de ce caractère doit être défini pour chaque application et parfois faire l'objet d'un accord entre l'émetteur des données et leur destinataire.

CR – *Retour du chariot*

Commande de mise en page qui déplace la position active à la première position de caractère de la même ligne.

Commandes d'appareil auxiliaire

DC1 – Caractère de commande d'appareil auxiliaire principalement destiné à enclencher ou à mettre en marche un appareil auxiliaire. Si on n'en a pas besoin pour cette fonction, il peut être utilisé pour rétablir dans un appareil le mode principal de fonctionnement (voir aussi DC2 et DC3) ou pour toute autre fonction de commande d'appareil auxiliaire non prévue par les autres DC.

DC2 – Caractère de commande d'appareil auxiliaire principalement destiné à enclencher ou à mettre en marche un appareil auxiliaire. Si on n'en a pas besoin pour cette fonction, il peut être utilisé afin que l'appareil fonctionne d'après un mode spécial (dans ce cas, DC1 sera utilisé pour ramener l'appareil au mode principal de fonctionnement) ou pour toute autre fonction de commande d'appareil auxiliaire non prévue par les autres DC.

DC3 – Caractère de commande d'appareil auxiliaire principalement destiné à déclencher ou à arrêter l'appareil auxiliaire. Cette fonction peut être un arrêt de niveau secondaire, par exemple attente, pause, mise en réserve ou halte (dans ce cas, DC1 est utilisé pour rétablir l'opération normale). Si on n'en a pas besoin pour cette fonction, il peut être utilisé pour toute autre fonction de commande d'appareil non prévue par les autres DC.

DC4 – Caractère de commande d'appareil auxiliaire principalement destiné à déclencher, arrêter ou interrompre un appareil auxiliaire. Si on n'en a pas besoin pour cette fonction, il peut être utilisé pour toute autre fonction de commande d'appareil auxiliaire non prévue par les autres DC.

Exemples d'usage de commandes d'appareil auxiliaire:

1) Une connexion

	marche – DC2	arrêt – DC4

2) Deux connexions indépendantes

Première connexion	marche – DC2	arrêt – DC4
Seconde connexion	marche – DC1	arrêt – DC3

3) Deux connexions dépendantes

Général	marche – DC2	arrêt – DC4
Particulier	marche – DC1	arrêt – DC3

4) Connexion de l'entrée et de la sortie

Sortie	marche – DC2	arrêt – DC4
Entrée	marche – DC1	arrêt – DC3

DEL – *Oblitération*

Caractère employé principalement pour effacer ou oblitérer les caractères erronés ou indésirables sur une bande perforée. Les caractères DEL peuvent également servir comme caractères de remplissage de temps ou de support d'information. Ils peuvent être insérés dans une suite de caractères ou en être retirés sans que le contenu d'information de cette suite soit affecté; mais, dans ce cas, l'insertion ou la suppression de ces caractères peut affecter la disposition des informations ou la commande des équipements.

DLE – *Echappement transmission*

Caractère de commande de transmission qui change la signification d'un nombre limité de caractères successifs qui le suivent. Ce caractère est utilisé exclusivement pour fournir des commandes supplémentaires de transmission. Seuls, des caractères graphiques et des caractères de commande de transmission peuvent être utilisés dans les séquences DLE.

EM – *Fin de support*

Caractère de commande qui peut être utilisé pour identifier la fin matérielle du support, ou la fin de la partie utilisée du support ou la fin de la partie désirée des informations enregistrées sur un support. La position de ce caractère ne correspond pas nécessairement à la fin matérielle du support.

ENQ – *Demande*

Caractère de commande de transmission employé comme demande de réponse d'une station éloignée – la réponse peut inclure l'identification de la station ou l'état de la station ou les deux. Lorsqu'un contrôle d'identité *Qui est là?* est exigé sur un réseau général de transmission avec commutation, la première utilisation du caractère ENQ après l'établissement de la liaison aura le sens *Qui est là?* (identification de la station). Une nouvelle utilisation du caractère ENQ peut ou non inclure la fonction *Qui est là?*, selon accord préalable.

EOT – *Fin de transmission*

Caractère de commande de transmission utilisé pour indiquer la fin de la transmission d'un ou de plusieurs textes.

ESC – *Echappement*

Caractère de commande employé pour fournir des fonctions de commande supplémentaires. Il modifie la signification d'un nombre limité de combinaisons d'éléments successifs qui le suivent et constituent la séquence d'échappement.
Les séquences d'échappement sont utilisées pour obtenir des fonctions de commande supplémentaires qui peuvent, entre autres, fournir des jeux de caractères graphiques en dehors du jeu normalisé. Ces commandes supplémentaires ne doivent pas être utilisées comme commandes additionnelles de transmission.
L'emploi du caractère ESC et des séquences d'échappement dans la mise en œuvre des techniques d'extension de code fait l'objet d'une norme ISO.

ETB – *Fin de transmission de bloc*

Caractère de commande de transmission utilisé pour indiquer la fin d'un bloc de données lorsque ces données sont divisées en bloc en vue de leur transmission.

EXT – *Fin de texte*

Caractère de commande de transmission utilisé pour terminer un texte.

FF – *Page suivante*

Commande de mise en page qui déplace la position active jusqu'à la position de caractère correspondante sur une ligne prédéterminée d'un imprimé ou d'une page suivante.

HT – *Tabulation horizontale*

Commande de mise en page qui déplace la position active jusqu'à la position de caractère prédéterminée suivante sur la même ligne.

Séparateurs d'information

IS_1 (US) – Caractère de commande employé pour séparer et qualifier des données dans un sens logique; sa signification spécifique doit être déterminée pour chaque application. Si ce caractère est employé dans l'ordre hiérarchique indiqué dans la définition générale de IS, il délimite un ensemble de données appelé SOUS-ARTICLE.

IS_2 (RS) – Caractère de commande employé pour séparer et qualifier des données dans un sens logique; sa signification spécifique doit être déterminée pour chaque application. Si ce caractère est employé dans l'ordre hiérarchique indiqué dans la définition générale de IS, il détermine un ensemble de données appelé ARTICLE.

IS_3 (GS) – Caractère de commande employé pour séparer et qualifier des données dans un sens logique; sa signification spécifique doit être déterminée pour chaque application. Si ce caractère est employé dans l'ordre hiérarchique indiqué dans la définition générale de IS, il détermine un ensemble de données appelé GROUPE.

IS_4 (FS) – Caractère de commande employé pour séparer et qualifier des données dans un sens logique; sa signification spécifique doit être déterminée pour chaque application. Si ce caractère est employé dans l'ordre hiérarchique indiqué dans la définition générale de IS, il délimite un ensemble de données appelé FICHIER.

LF – *Interligne*

Commande de mise en page qui déplace la position active jusqu'à la position de caractère correspondante sur la ligne suivante.

NAK – *Accusé de réception négatif*

Caractère de commande de transmission transmis par un récepteur comme réponse négative à l'émetteur.

NUL – *Nul*

Caractère de commande destiné au remplissage de temps ou de support d'information. Les caractères NUL peuvent être insérés dans une suite de caractères ou en être retirés sans que le contenu d'information de cette suite en soit affecté; mais, dans ce cas, l'adjonction ou la suppression de ces caractères peut modifier la disposition des informations et/ou la commande des équipements.

SI – *En code*

Caractère de commande qui est employé en combinaison avec les caractères HORS CODE et ECHAPPEMENT pour étendre le jeu de caractères graphiques du code. Il peut rétablir la signification normalisée des combinaisons d'éléments qui le suivent. L'effet de ce caractère dans la mise en œuvre de techniques d'extension de code fait l'objet d'une norme ISO.

SO – *Hors code*

Caractère de commande qui est employé en combinaison avec les caractères EN CODE et ECHAPPEMENT pour étendre le jeu de caractères graphiques du code. Il peut modifier la signification de combinaisons d'éléments des colonnes 2 à 7 qui le suivent jusqu'au caractère EN CODE. Néanmoins, les caractères ESPACE (2/0) et OBLITÉRATION (7/15) ne sont pas modifiés par le caractère HORS CODE. L'effet de ce caractère dans la mise en œuvre de techniques d'extension de code fait l'objet d'une norme ISO.

SOH – *Début d'en-tête*

Caractère de commande de transmission employé comme premier caractère d'un en-tête de message d'information.

SP – *Espace*

Caractère qui déplace la position active d'une position de caractère en avant sur la même ligne. Ce caractère est considéré comme un caractère graphique non imprimé.

STX – *Début de texte*

Caractère de commande de transmission précédant un texte et employé pour terminer un en-tête.

SUB – *Caractère de substitution*

Caractère de commande employé pour remplacer un caractère reconnu non valide ou erroné. Le caractère SUB est introduit par le système de traitement.

SYN – *Synchronisation*

Caractère de commande de transmission utilisé par un système de transmission synchrone en l'absence de tout autre caractère (situation inactive) pour produire un signal à partir duquel le synchronisme peut être obtenu ou maintenu entre équipements terminaux de données.

VT – *Tabulation verticale*

Caractère de mise en page qui déplace la position active jusqu'à la position de caractère correspondante sur la ligne suivante prédéterminée.

Avis V.4

STRUCTURE GÉNÉRALE DES SIGNAUX DU CODE POUR L'ALPHABET INTERNATIONAL N° 5 DESTINÉ À LA TRANSMISSION DE DONNÉES SUR LE RÉSEAU TÉLÉPHONIQUE PUBLIC [3]

(Mar del Plata, 1968, modifié à Genève, 1976)

Le CCITT,

1. *considérant en premier lieu*

l'accord réalisé entre l'Organisation internationale de normalisation (ISO) et le CCITT sur les principales caractéristiques d'un alphabet à sept moments d'information (Alphabet international n° 5) utilisable pour les transmissions de données et pour les besoins des télécommunications que ne peut satisfaire l'Alphabet télégraphique international n° 2 actuel à cinq moments;

l'intérêt que présente aussi bien pour les usagers que pour les services de télécommunications un accord sur l'ordre chronologique de transmission des bits dans le mode «série»,

émet l'avis

que le numéro conventionnel du rang du moment dans le tableau alphabétique des combinaisons corresponde à l'ordre chronologique de transmission dans le mode «série» sur les voies de télécommunications;

que, lorsque ce rang dans la combinaison représente le poids du bit en numérotation binaire, la transmission des bits soit effectuée dans le mode «série» par ordre des poids croissants;

[3] Voir Avis X.4 pour la transmission de données sur le réseau public pour données.

· Analyse excellente du problème.

|d'abord| ~~tant du pt~~ de leur caract. de la catég. attributives. → définir les paramè.

|ensuite| ~~que de~~ la mise en oeuvre · négociat. ~~des échanges~~ p. rendre ~~et esthan~~ ~~l'adaptat~~ — à exploiter p. tenter de les rendre compatibles

la compatibilité possible par simulation située au niveau le + adéq.

|enfin|

↓

La conception des algorithmes et routines qui vont permettre ~~d'envol~~ de réaliser une compatibilité satisfaisante

① de ~~P.~~ Format départ et/ou à l'arrivée

② Alg et R. non incrémentées

- tentative.

- examen de la possibilité et des difficultés rencontrées
  - lacunes. — Quelles sont-elles ?

- programmation m̂ élément: — sur ~~appareil~~ (négociation, —)
  - sur l'enchaînem des Routines ~~de la~~ en ~~réalisation~~ permettant, ~~des~~ fonction de canalisation — de mesurer l'efficacité

① T com / ~~ce test~~ signif. / le + courants / chez les constructeurs les + sollicités sans modification hardware qui paraît ~~liminaire~~ et ~~à tenter~~ possible mais difficulté c'est choix des modules à transformer et comment.