Télé-relevé par liaison téléphonique RTC des appareils de comptage de type « Compteur Vert Electronique »

Identification : **Enedis-NOI-CPT_04E**
Version : **3**
Nb. de pages : **51**

Version	Date d'application	Nature de la modification	Annule et remplace
1	15/12/2007	Création	
2	01/03/2008	Prise en compte de l'identité visuelle d'ERDF	NOP-RES_105E
3	01/02/2017	Prise en compte de la nouvelle dénomination sociale d'Enedis	ERDF-NOI-CPT_04E

Résumé / Avertissement :

Cette spécification s'applique aux appareils de comptage électroniques de type « Compteur Vert Electronique » (CVE) assurant le comptage triphasé des énergies active et réactive et la quantification des dépassements de puissance.

Elle définit les informations qui sont administrées par cet appareil et auxquelles les utilisateurs des réseaux gérés par Enedis peuvent accéder moyennant certaines dispositions de réception de signaux et de traitement informatique appropriés (interfaces de communication et applications de télé-relevé non décrites dans ce document). Cet accès n'est possible que si le compteur est équipé d'un modem raccordé au réseau téléphonique commuté public (RTC). L'utilisateur du réseau ou son mandataire doit donc, en préalable, demander aux services d'Enedis la vérification du raccordement du compteur à un accès téléphonique et les informations sur les conditions de cet accès (code d'accès et horaire d'appel notamment).

Enedis - Tour Enedis
34 place des Corolles
92079 Paris La Défense Cedex
enedis.fr

SA à directoire et à conseil de surveillance
Capital de 270 037 000 euros
R.C.S. de Nanterre 444 608 442
Enedis est certifié ISO 14001 pour l'environnement

SOMMAIRE

1. Généralités

1.1. Domaine d'application

La présente spécification décrit les données gérées par un appareil de comptage de technologie électronique et qui peuvent être lues par les utilisateurs des Réseaux de Distribution gérés par Enedis au moyen d'une liaison téléphonique filaire (RTC). Les appareils de comptage concernés sont les compteurs électroniques de type « Compteur Vert Electronique » (CVE). Ils sont généralement utilisés dans le cas des utilisateurs du Réseau de Distribution de niveau de tension HTA, de puissance souscrite comprise entre 250 kW et 10 MW, et bénéficiant du tarif Vert A. Ils sont également utilisés dans le cas de certains utilisateurs du Réseau de Distribution de niveau de tension BT ou dont la puissance souscrite est inférieure à 250 kW ou qui bénéficient d'une autre tarification. Ces appareils permettent le comptage de l'énergie active et de l'énergie réactive, la quantification des dépassements de puissance, ainsi que la gestion d'autres grandeurs pouvant être utiles à la gestion contractuelle ou à la maîtrise des consommations (puissances atteintes, …).
L'appareil conserve dans sa mémoire un certain nombre d'informations sur une partie des mesures qu'il a effectuées. Ces informations sont accessibles localement. Elles peuvent être transmises par téléphone sur demande d'un système de traitement informatique si l'appareil est équipé d'un modem raccordé au réseau téléphonique commuté public (RTC).

Le compteur assure les différentes fonctions décrites ci-après :

- la mesure des énergies active et réactive et de la puissance active ;
- la gestion de la date et de l'heure de l'appareil (horloge interne) ;
- la gestion du tarif en cours par la sommation des énergies, les calculs des dépassements de puissance, et leur répartition par période tarifaire grâce à des tables internes définissant une structure « horosaisonnière » et, si nécessaire, des ordres de télécommande externe (cas d'un tarif à effacement) ;
- la mémorisation des grandeurs nécessaires à la gestion contractuelle et à la gestion des consommations en fonction du cycle de gestion prévu par le tarif : les données de consommation servant à la facturation sont mémorisées pour différentes périodes tarifaires successives ;
- la mémorisation des points de puissance moyenne, en puissance active par période de 10 minutes mesurées durant les 12 derniers jours ;
- la visualisation sur l'afficheur d'une partie des informations mesurées ou élaborées par l'appareil, ainsi que des paramètres de fonctionnement ;
- le relevé des informations à distance : chaque appareil est muni d'un modem qui doit, pour cela, être raccordé au réseau téléphonique commuté permettant, à l'initiative d'un système de traitement informatique central, de réaliser un échange de données bidirectionnel ;
- la programmation à distance d'une partie des paramètres de fonctionnement (même procédé que le relevé à distance) ;
- la mise à disposition d'informations sur un bornier réservé à l'utilisateur du réseau (contacts).

Le présent document traite uniquement du relevé de certaines grandeurs accessibles à distance par la liaison téléphonique RTC sous réserves de présentation d'un code d'accès appelé aussi « clé client ». Dans le cas du présent compteur, ce code d'accès est aussi appelé « identifiant esclave ».

Le document est structuré en 2 parties principales :

- la partie 1 (chapitre 2) décrit d'abord les groupes de données disponibles, puis les détails des données mises à disposition par le compteur pour tout système de traitement informatique (non décrit dans ce document) qui est relié au modem du compteur par le réseau de téléphonie RTC ;
- la partie 2 (chapitre) décrit les protocoles de communication régissant les transferts de données du compteur vers le système de traitement informatique. Tout logiciel de traitement (non décrit dans ce document) du système de traitement informatique doit se conformer à ces protocoles.

1.2. Références normatives

Les normes internationales ou françaises citées en référence contiennent des dispositions qui, par suite de la référence qui en est faite, constituent des dispositions valables pour la présente spécification.
Les normes citées sont sujettes à révision.

NF EN ISO/CEI 7498-1 : Technologies de l'information - Modèle de référence de base pour l'interconnexion de système ouverts (OSI) - Partie 1 : le modèle de base (décembre 1995).

NF ISO 7498-2 : Systèmes de traitement de l'information - Interconnexion de systèmes ouverts - Modèle de référence de base - Partie 2 : architecture de sécurité (septembre 1990).

NF ISO 7498-3 : Systèmes de traitement de l'information - Interconnexion de systèmes ouverts - Modèle de référence de base - Partie 3 : dénomination et adressage (juillet 1989).

UIT-T V23 : Modem à 600/1200 bauds normalisé pour usage sur le réseau téléphonique général avec commutation.

UIT-T V24 : Liste des définitions des circuits de jonction à l'interface entre l'équipement terminal de traitement de données (ETTD) et l'équipement de terminaison du circuit de données (ETCD).

UIT-T V25 : Equipement de réponse automatique et/ou équipement d'appel automatique en mode parallèle sur le réseau téléphonique général avec commutation, y compris les procédures de neutralisation des dispositifs de protection contre l'écho lorsque les appels sont établis aussi bien entre postes à fonctionnement manuel qu'entre postes à fonctionnement automatique.

UIT-T V25bis : Equipement d'appel et/ou de réponse automatique sur le réseau téléphonique général avec commutation, utilisant les circuits de liaison de la série 100.

UIT-T V28 : Caractéristiques électriques des circuits de jonction dissymétriques pour transmission par double courant.

UIT-T X25 : Interface entre équipement terminal de traitement de données (ETTD) et équipement de transmission du circuit de données (ETCD) pour terminaux fonctionnant en mode-paquet et raccordés à des réseaux publics pour données par circuit spécialisé.

ISO 2110 : Technologies de l'information - Communication de données - Connecteur d'interface ETTD/ETCD à 25 pôles et affectation des numéros de contacts (1989).

Rappel : l'association UIT-T a remplacé le CCITT depuis 1993.

2. PARTIE 1 - Données télé-relevables par la liaison téléphonique

2.1. Définitions concernant le comptage et le télé-relevé

2.1.1. Principes généraux sur le télé-relevé des compteurs

L'utilisateur souhaitant exploiter des données par télé-relevé doit disposer d'un système de traitement informatique adapté. Ce système, développé à partir de la connaissance des protocoles de transmission des données, lui permet alors d'accéder à certaines données stockées dans le compteur équipé d'un modem raccordé au réseau téléphonique.

L'utilisateur du réseau, ou son mandataire, désirant accéder aux données d'un compteur par télé-relevé, doit obligatoirement, en préalable, demander aux services d'Enedis :

- la vérification de la présence dans le compteur d'un modem raccordé au réseau téléphonique RTC,
- la fourniture du code d'accès aux données du compteur,
- la ou les plages horaires d'appel qui lui sont attribuées pour l'accès aux données du compteur.

Les codes d'accès

Pour accéder aux données du compteur, il est nécessaire de disposer d'un code d'accès fourni sur demande par Enedis sous le nom de « clé client ». Dans le cas du présent compteur, ce code d'accès est aussi appelé « identifiant esclave » ou « identité de l'esclave ». Il devra être présenté par l'outil de traitement lors des échanges avec le compteur (cf. chapitre 3.4).
Lors de cette opération, un autre code devra être présenté qui est appelé « clé maître », « identifiant maître » ou « identité du maître ». Pour les opérations de lecture des données décrites ci-après, ce second code devra être affecté de la valeur nulle.

Les plages d'appel au compteur

L'accès aux données du compteur par le réseau téléphonique peut être utilisé par différentes entités telles que :

- d'une part, les différents services d'Enedis dans le cadre de leurs missions de relevé de données pour facturation, de relevé ou programmation de données pour contrôle ou maintenance des comptages ou de relevé de données pour l'exploitation du réseau (qualité de fourniture, ...) ;
- d'autre part, l'utilisateur du réseau et ses divers mandataires dans le cadre du suivi de sa consommation ou de prestations diverses.

Afin de garantir pour chacune de ces entités, la meilleure fiabilité possible dans l'exécution de ces accès téléphoniques au compteur et notamment, de ne pas perturber les opérations de télé-gestion du compteur effectuées par Enedis, un partage des possibilités d'accès téléphonique au compteur est mis en œuvre sous la forme d'une répartition temporelle en plages horaires d'appels qui sont réservées à chaque entité.

Enedis assure, pour chaque compteur, la gestion de ces plages et ainsi, la répartition des possibilités d'accès. Chaque entité s'engage à n'appeler le compteur que pendant la ou les plages horaires qui lui sont attribuées par Enedis. Ces plages pourront être plus ou moins importantes suivant le nombre d'entités concernées et le mode de raccordement du compteur au réseau téléphonique : raccordement sur une ligne dédiée au compteur ou sur une ligne partagée temporellement ou physiquement avec d'autres appareils (compteurs, appareils de mesure d'Enedis ou matériels de l'installation téléphonique de l'utilisateur du réseau).

Précisions sur les modes de raccordement au réseau téléphonique commuté (RTC)

Le raccordement du compteur au réseau téléphonique commuté peut se faire :

- soit directement sur une ligne dont l'usage est réservé à ce compteur (ligne dédiée au compteur),
- soit sur une ligne téléphonique utilisée par plusieurs appareils.

Dans ce dernier cas, la répartition des appels ou « partage de ligne » peut être effectué par un appareil d'aiguillage physique (aiguilleur, PABX, …) ou par une fonction de partage temporelle implémentée dans les appareils concernés (cas de la fenêtre d'écoute du compteur).
Dans le cas du compteur vert électronique, la technique de la fenêtre d'écoute n'est pas utilisable pour ce compteur. Néanmoins, ce compteur peut partager une ligne téléphonique avec des compteurs utilisant la technique de la fenêtre d'écoute.

Généralement, le partage de ligne a lieu :

- soit entre un compteur et les installations téléphoniques de l'utilisateur sur une ligne téléphonique dont il est le souscripteur,
- soit entre plusieurs compteurs sur une ligne téléphonique dont Enedis est le souscripteur.

De manière générale, en fonction des contraintes générées par la configuration de l'installation téléphonique et les différentes utilisations existantes, Enedis fournira à l'utilisateur du réseau, les plages horaires d'appel pendant lesquelles il peut appeler le compteur de son Site.

Afin d'éviter des échecs d'appels dus à une désynchronisation des heures courantes d'un compteur appelé et du système de traitement informatique appelant, notamment en cas de partage de ligne avec un appareil fonctionnant en fenêtre d'écoute, il est fortement recommandé de :

- maintenir avec précision la justesse de l'heure courante du système de traitement informatique appelant, ainsi que celle des informations qu'il gère concernant les plages horaires d'appel des compteurs,
- réaliser les appels à l'intérieur de chacune des plages horaires d'appel en préservant une marge d'erreur de 5 minutes, c'est à dire en appelant le compteur, au plus tôt, 5 minutes après le début de la plage d'appel et, au plus tard, 5 minutes avant la fin de la plage d'appel,
- n'effectuer qu'une seule tentative d'appel à un compteur donné au cours d'une même plage d'appel si celle-ci est une fenêtre d'écoute.

2.1.2. Vocabulaire

Le « **poste horaire** » désigne une catégorie d'heures de la journée pendant lesquelles s'applique un tarif donné (par exemple : Heures Pleines, Heures Creuses, Heures de Pointe, …).
La **« période tarifaire »** (appelée aussi **« poste horosaisonnier »** ou **« poste tarifaire »**) désigne une combinaison de la « **saison** » (par exemple : « Hiver » de novembre à mars, « Eté » d'avril à octobre, …) et du « **poste horaire** » (par exemple : heures pleines, heures creuses) qui permet de déterminer le tarif applicable à l'énergie à n'importe quel instant de l'année.

Le **« Tarif Vert »** géré par le Compteur Vert Electronique (CVE) comprend 4 variantes de la tarification basées sur 4 types de découpage horosaisonnier et répartis en 2 types de tarif (le tarif « A8 » et le tarif « A » appelé aussi « A5 ») et 2 options tarifaires pour chaque tarif (Base et EJP) :

- tarif Vert A en option Base, appelé A5-Base à 5 périodes tarifaires ;
- tarif Vert A en option E.J.P. (Effacement Jours de Pointe), appelé A5-EJP à 4 périodes tarifaires ;
- tarif Vert A8 en option Base appelé A8-Base à 8 périodes tarifaires ;
- tarif Vert A8 en option E.J.P., appelé A8-EJP à 6 périodes tarifaires.

La mise en application du **Tarif Vert A en option Base (A5-Base)** nécessite la définition au cours de l'année des **5 périodes tarifaires** suivantes réparties en 2 saisons : « Hiver » (de novembre à mars inclus) et « Eté» (d'avril à octobre inclus) :

- Heures de Pointe (P) : 2 fois 2 heures par jour du lundi au samedi inclus, et pour les mois de décembre, janvier et février (appelées aussi « Heures de Pointe Fixe ») ;
- Heures Pleines d'Hiver (HPH): toutes les heures de la saison « Hiver » qui ne sont ni en Heures de Pointe, ni en Heures Creuses ;
- Heures Creuses d'Hiver (HCH) : 8 heures par jour du lundi au samedi inclus (éventuellement non consécutives), et les dimanches entiers, pour tous les mois de la saison « Hiver » ;
- Heures Pleines d'Eté (HPE) : toutes les heures de la saison « Eté » qui ne sont pas en Heures Creuses ;
- Heures Creuses d'Eté (HCE) : 8 heures par jour du lundi au samedi inclus (éventuellement non consécutives), et les dimanches entiers, pour tous les mois de la saison « Eté ».

Les périodes tarifaires relatives à l'**option EJP du Tarif Vert A5 (A5-EJP)** sont les suivantes :

- Heures de Pointe Mobile (PM) : 18 heures par jour pendant 22 jours (non consécutifs) répartis sur les mois de la saison « Hiver » ;

- Heures d'Hiver (HH) : toutes les heures de la saison « Hiver » qui ne sont pas en Heures de Pointe Mobile. Ces heures continuent, même dans l'option EJP, à être réparties en Heures Pleines d'Hiver (HPH) et Heures Creuses d'Hiver (HCH) avec la même définition que dans le tarif A5 en option Base ;
- Heures Pleines d'Eté (HPE) : même définition que dans le tarif A5 en option Base ;
- Heures Creuses d'Eté (HCE) : même définition que dans le tarif A5 en option Base.

La mise en application du **Tarif Vert A8 en option Base (A8-Base)** nécessite la définition au cours de l'année des **8 périodes tarifaires** suivantes réparties en 4 saisons :

- « Hiver », de décembre à février inclus comportant des Heures Pleines (HPH), des Heures Creuses (HCH) et des Heures de Pointe (P) ;
- « Demi-saison », en novembre et mars, comportant des Heures Pleines (HPD) et des Heures Creuses (HCD) ;
- « Eté », d'avril à juin inclus et de septembre à octobre inclus, comportant des Heures Pleines (HPE) et des Heures Creuses (HCE) ;
- « Saison Creuse », en juillet et août, comportant uniquement des Heures Creuses (CC) et appelée également Juillet-Août (JA).

Les **8 périodes tarifaires** sont définies de la manière suivante :

- Heures de Pointe (P) : 2 fois 2 heures par jour du lundi au vendredi inclus, pendant toute la saison « Hiver » (appelées aussi « Heures de Pointe Fixe ») ;
- Heures Pleines d'Hiver (HPH): toutes les heures de la saison « Hiver » qui ne sont ni en Heures de Pointe, ni en Heures Creuses ;
- Heures Creuses d'Hiver (HCH) : 6 heures par jour du lundi au vendredi inclus, et les samedis, dimanches, jours fériés et assimilés durant toute de la saison « Hiver » ;
- Heures Pleines de Demi-saison (HPD) : toutes les heures de la saison « Demi-saison » qui ne sont pas en Heures Creuses ;
- Heures Creuses de Demi-saison (HCD) : même définition que les Heures Creuses d'Hiver (HCH) mais durant toute la saison « Demi-saison » ;
- Heures Pleines d'Eté (HPE) : toutes les heures de la saison « Eté » qui ne sont pas en Heures Creuses ;
- Heures Creuses d'Eté (HCE) : même définition que les Heures Creuses d'Hiver (HCH) mais durant toute la saison « Eté» ;
- Heures Creuses de saison Creuse (CC ou JA) : toutes les heures de tous les jours de la saison « Saison Creuse ».

Les samedis, dimanches, jours fériés et assimilés de toutes les saisons sont entièrement classés en Heures Creuses de la saison considérée. Tous les jours de la Saison Creuse sont entièrement classés en Heures Creuses de cette saison.

Les périodes tarifaires relatives à l'**option EJP du Tarif Vert A8 (A8-EJP)** sont les suivantes :

- Heures de Pointe Mobile (PM) : 18 heures par jour pour 22 jours répartis sur les mois des saisons « Hiver » et « Demi-saison » ;
- Heures d'Hiver (HH) : toutes les heures de la saison « Hiver » qui ne sont pas en Heures de Pointe Mobile ;
- Heures de Demi-saison (HD) : toutes les heures de la saison « Demi-saison » qui ne sont pas en Heures de Pointe Mobile ;
- Heures Pleines d'Eté (HPE) : même définition que pour le tarif A8 en option Base ;
- Heures Creuses d'Eté (HCE) : même définition que pour le tarif A8 en option Base ;
- Heures Creuses de saison Creuse (CC ou JA) : même définition que pour le tarif A8 en option Base.

Le **coefficient de transformation** appelé **TC.TT** est le produit des coefficients de transformation des réducteurs de mesure d'intensité et de tension. Il permet d'obtenir les énergies active et réactive respectivement en kWh et kvarh réellement consommées par le Site concerné. Ce coefficient est appliqué par le compteur sur les mesures qu'il a effectuées sur les circuits « secondaires » des transformateurs afin d'obtenir les grandeurs rapportées aux circuits « primaires » des transformateurs.

« **Valeurs de références**» : elles servent à vérifier que la ligne téléphonique répond correctement et que le dialogue est bien établi avec le modem du compteur.

Les « périodes contractuelles » gérées par le compteur sont des périodes mensuelles correspondant à celles définies par le Tarif Vert. Le compteur gère deux périodes contractuelles : **la période en cours appelé « mois en cours »** et **la période précédente appelée « mois précédent »**.

Le compteur effectue automatiquement le changement de période le premier jour de chaque mois à deux heures du matin.
Lors d'un changement de période contractuelle, le compteur ferme la période contractuelle en cours et ouvre une nouvelle période contractuelle. Pour cela, il effectue une opération appelée « glissement » qui consiste à copier les données gérées au titre de la période « mois en cours » dans les données stockées au titre de la période « mois précédent » où elles remplacent les données précédemment stockées.

Lors du « glissement », les données de la période « mois en cours » évoluent de la manière suivante :

- celles qui font l'objet d'un cumul supra-mensuel (de type « index ») continuent d'évoluer à partir des valeurs contenues au changement de période (cas des index d'énergie),
- celles qui sont de type cumul mensuel sont remises à zéro, puis évoluent jusqu'au prochain changement de période (cas des informations de dépassement de puissance).

Rappel : « **TRIMARAN** » est le nom du protocole utilisé pour les communications téléphoniques en RTC avec le compteur électronique de type « Compteur Vert Electronique » (CVE). Les spécifications détaillées de TRIMARAN sont fournies dans la partie 2 (chapitre 3) du présent document.

2.2. Liste des groupes de données télé-relevables et mode d'accès

Le compteur gère de nombreuses informations programmables ou relevables sur Site ou à distance via une liaison téléphonique.
La plupart de ces informations concernent le contrat en cours et les consommations mesurées. Elles sont accessibles à distance sous la forme de données classées en 5 groupes qui sont, chacun, identifiés par un « code de télé-relevé » correspondant au numéro du groupe de données.
Le tableau suivant donne les désignations de ces groupes de données, ainsi que les valeurs des « codes de télé-relevé » associés. Le code de télé-relevé d'un groupe de données devra être fourni au compteur par le système de traitement informatique lors des échanges de communication afin d'identifier les informations à fournir par le compteur.

Liste des données de télé-relevé

Désignation des données télé-relevées	Code de télé-relevé
Tableau des puissances moyennes	04
Informations du mois en cours	02
Informations du mois précédent	01
Etat du compteur	03
Valeurs de référence	05

Précisions sur le mode d'accès aux groupes de données télé-relevables

Pour accéder à un groupe de données télé-relevables, il convient de communiquer au compteur les informations correspondant à ce groupe dans le double-octet de la variable « Code » conformément à ce qui est précisé dans le chapitre 3 décrivant les protocoles utilisés pour la communication, au chapitre 3.4.

Le contenu du 1er octet du double-octet de la variable « Code » est la valeur (codée en hexadécimal) du code de télé-relevé du groupe de données désiré conformément au tableau ci-dessus.

Le contenu du 2ème octet du double-octet de la variable « Code » est variable en fonction du type de compteur et du groupe de données désiré. De manière générale, il peut contenir soit une référence identifiant l'application du compteur, soit un code complémentaire du code de télé-relevé précisant l'identité des données désirées.

Pour le compteur vert électronique, le contenu de ce 2ème octet n'est pas géré par le compteur. Néanmoins, par convention, la valeur 00 devra lui être affectée.

2.3. Tableau des puissances moyennes (code 04).

2.3.1. Généralités sur le tableau des puissances moyennes

Ce tableau fournit la totalité des puissances moyennes atteintes mesurées par le compteur. Les valeurs fournies correspondent à des puissances actives avec une résolution de 1 kW (sauf période avec coupure d'alimentation). La moyenne de puissance est calculée par période de 10 minutes. Les valeurs fournies sont horodatées.

Les informations qu'il contient permettent de reconstituer la courbe de charge du Site considéré par période de temps de 10 minutes.

La taille du tableau est de 4 kilo-octets. Le nombre de valeurs fourni correspond à une durée totale d'environ 12 jours. Pour éviter toute perte d'information, il convient donc de relever ce tableau tous les 11 jours.

Le compteur est conçu pour pouvoir enregistrer dans un espace mémoire minimal les plus fortes puissances atteintes par des utilisateurs du réseau avec la meilleure résolution possible. La puissance souscrite maximale des utilisateurs du réseau équipés de ce type de compteur (Tarif Vert ou autres) est de 10 MW. Le compteur peut enregistrer des puissances atteintes allant

jusqu'à 32 MW avec une résolution de 1 kW grâce à un enregistrement des valeurs sur 15 bits utiles.

Le tableau des puissances moyennes est donc constitué d'éléments de 16 bits, soit 2 octets. Les octets de poids faibles sont envoyés en tête lors de la transmission.

Afin de permettre la reconstitution complète de la courbe de charge, ce tableau comporte des informations de puissance appelées « élément-puissance », ainsi que des éléments de datation appelés « élément-datation » contenant des informations de date, heure et poste horaire. Il permet également de distinguer les périodes « 10 minutes » pendant lesquelles s'est produite une coupure d'alimentation.

Le tableau est organisé de manière cyclique (pile circulaire) et comporte donc un marqueur appelé « élément-pointage » indiquant l'emplacement des données les plus récentes.

2.3.2. Élément-puissance

Un élément-puissance peut être présenté sous deux formats différents en fonction de la présence ou non d'une coupure de l'alimentation pendant la période de 10minutes qu'il représente.

Période sans coupure de l'alimentation :

L'élément-puissance mémorisant l'information de puissance moyenne d'une période « 10 minutes » pendant laquelle il n'y a pas eu de coupure de l'alimentation, est caractérisé par la présence d'une valeur 0 dans le bit 15 (poids fort du double-octet). Les bits 14 à 0 du double-octet contiennent la valeur, codée en binaire, de la puissance moyenne mesurée avec une résolution de 1 kW.

bit 15	bits 14 à 0
0	PUISSANCE MOYENNE SANS COUPURE (LSB = 1 kW)

Période avec coupure de l'alimentation :

En cas de coupure de l'alimentation, l'information de coupure est signalée par la présence des valeurs 1 et 0 placées respectivement dans les bits 15 et 14 du double-octet de chacun des éléments contenant la puissance moyenne d'une période « 10 minutes » concernée par la coupure. Les bits 13 à 0 du double-octet contiennent alors la valeur, codée en binaire, de la puissance moyenne mesurée, et la résolution de cette valeur est alors de 2 kW pour permettre de mémoriser des valeurs atteignant 32 MW.

bit 15	bit 14	bits 13 à 0
1	0	PUISSANCE MOYENNE SANS COUPURE (LSB = 2 kW)

Dans le cas d'une coupure d'alimentation d'une durée supérieure à 10 minutes, le compteur enregistre une valeur de puissance moyenne nulle dans les bits 13 à 0 du double-octet de chacune des périodes 10 minutes entièrement comprises dans la coupure. Cette opération a lieu au moment du retour de l'alimentation. Le compteur reconstitue alors autant d'éléments-puissance à zéro qu'il y a eu de périodes dix minutes pendant lesquelles la coupure a été totale. Chacun de ces éléments-puissance contient des bits 15 et 14 aux valeurs 1 et 0 et les autres bits à la valeur 0. Au milieu de ces éléments-puissance sont également insérées les éléments-datation nécessaires tel que décrit ci-après.

2.3.3. Élément-datation

Les informations de date, d'heure et de poste horaire sont codées dans un double-octet.

Pour distinguer cette information de celle des éléments-puissance, les bits 15 et 14 du double-octet contiennent tous les deux la valeur 1.

L élément-datation fournit par ses bits 13 et 12 un code indiquant le poste horaire en cours à partir de ce point de datation, conformément au tableau ci dessous. L'information de poste horaire en cours est valable pour tous les élément-puissance qui suivent chronologiquement cet élément-datation, et cela jusqu'au prochain élément-datation du tableau.

bit 13	bit 12	Poste horaire
0	0	Heures pleines
1	0	Heures creuses
0	1	Pointe Fixe
1	1	Pointe Mobile

Les bits 11 à 0 du double-octet contiennent les informations de date et d'heure conformément au format suivant :

- le jour du mois est fourni en valeur modulo 16 (jour dans la quinzaine) et est codé en binaire sur les bits 11 à 8 (le 16 du mois est codé par la valeur 0 et le 31 du mois est codé à la valeur 15 ;
- l'heure courante du jour est codée en binaire sur les bits 7 à 3 (valeur de 0 à 23) ;
- la dizaine de minutes courante est codée en binaire sur les bits 2 à 0 (valeur de 0 à 5).

bit 15	bit 14	bits 13 à 12	bits 11 à 8	bits 7 à 3	bits 2 à 0
1	1	Poste horaire	Jour dans la quinzaine	Heure	Dizaine de minutes

Un élément-datation est inséré entre les deux éléments-puissance qui encadrent l'instant considéré lors de chacun des évènements suivants :

- chaque passage à une heure ronde (pour établir une chronologie de base des valeurs mémorisées),
- chaque changement de poste horaire ne se produisant pas à une heure ronde,
- chaque remise à l'heure de l'horloge interne de l'appareil (programmation externe),
- chaque changement d'heure légale (changement d'heure courante réalisé automatiquement par le compteur en fonction de ses programmations internes, lors du passage de l'heure d'été à l'heure d'hiver et du passage de l'heure d'hiver à l'heure d'été).

2.3.4. Élément-pointage

Le tableau étant organisé de manière cyclique, un marqueur, appelé « élément-pointage » et de format particulier, est inséré dans le tableau afin d'indiquer l'emplacement des données les plus récentes et de faciliter la reconstitution de la chronologie des éléments du tableau

Cet élément est positionné à la suite du dernier élément-puissance enregistré par le compteur. Les éléments-puissance transmis à la suite de ce marqueur sont donc les plus anciennes que le compteur ait enregistrées.

Cet élément-pointage est constitué de deux doubles-octets soit 32 bits qui sont tous positionnés à la valeur 1 (soit quatre octets à la valeur hexadécimale FF).

2.4. Informations du mois en cours (code 02)

Ce tableau fournit les informations concernant la période contractuelle « mois en cours » sous la forme d'un ensemble de 81 octets définis dans le tableau suivant.
Lorsque les informations sont fournies sur plusieurs octets, les octets de poids faibles sont transmis en tête.

Dans le cas où un poste horaire n'existe pas pour le mois considéré, les informations correspondant à ce poste horaire sont fournies avec des valeurs nulles.

Informations concernant la période contractuelle « mois en cours ».

Groupe	Description de la donnée	Codage ou unité	Taille	Positionnement du 1er octet dans la chaîne
Date du relevé	Jour, mois, an (JJMMAA)	BCD	3 octets	1
Heure du relevé	Heure, Minute (HHMM)	BCD	2 octets	4
TARIF	Informations tarifaires	Cf. détails ci-après	1 octet	6
MOIS	Mois concerné (1 à 12)	BCD	1 octet	7
POINTE	Energie active	kWh	4 octets	8
	Energie réactive	kvarh	4 octets	12
	Temps de fonctionnement	En nombre de périodes 10min	2 octets	16
	Dépassement quadratique	kW	3 octets	18
	Nombre de dépassements		2 octets	21
	Puissance maximale atteinte	kW	2 octets	23
HEURES PLEINES	Energie active	kWh	4 octets	25
	Energie réactive	kvarh	4 octets	29
	Temps de fonctionnement	En nombre de périodes 10min	2 octets	33
	Dépassement quadratique	kW	3 octets	35
	Nombre de dépassements		2 octets	38
	Puissance maximale	kW	2 octets	40
HEURES CREUSES	Energie active	kWh	4 octets	42
	Energie réactive	kvarh	4 octets	46
	Temps de fonctionnement	En nombre de périodes 10min	2 octets	50
	Dépassement quadratique	kW	3 octets	52
	Nombre de dépassements		2 octets	55
	Puissance maximale	kW	2 octets	57
PUISSANCES SOUSCRITES	PSP (Heures de Pointe)	kW	2 octets	59
	PS HPH (Heures Pleines d'Hiver)	kW	2 octets	61
	PS HCH (Heures Creuses d'Hiver)	kW	2 octets	63
	PS HPE (Heures Pleines d'Eté)	kW	2 octets	65
	PS HCE (Heures Creuses d'Eté)	kW	2 octets	67
	PS PM (Heures de Pointe Mobile)	kW	2 octets	69
	PS HPD Heures Pleines de Demi-saison	kW	2 octets	71
	PS HCD (Heures Creuses de Demi-saison)	kW	2 octets	73
	PS CC (Heures Creuses de saison Creuse)	kW	2 octets	75
COEFFICIENT	Rapport TCxTT		2 octets	77
ENERGIE DE DEPASSEMENT	$\sum \Delta P$ (en option EJP)	kWh	3 octets	79

L'information TARIF est codée sur un octet sous la forme suivante :

bit n°	7	6	5	4	3	2	1	0
	P	HP	HC	CC	PM	D	EJP	A/B

P, HP et HC décrivent le poste horaire en cours.
CC décrit l'état du « contact client » (contact réservé à l'utilisateur du réseau).
PM décrit l'état de poste tarifaire en Pointe Mobile.

D décrit le mode de calcul des dépassements qui est appliqué par le compteur :

- valeur 0 pour le calcul arithmétique (S = N Δ P),
- valeur 1 pour le calcul quadratique (S = $\sqrt{\sum \Delta P^2}$),

EJP et A/B décrivent le type de contrat en cours qui est appliqué par le compteur :

- bit 1 : option tarifaire Base ou EJP (valeurs : 1 pour EJP et 0 pour Base),
- bit 0 : type de tarif A5 ou A8 (valeurs : 1 pour A8 et 0 pour A5).

2.5. Informations du mois précédent (code 01)

Ce tableau fournit des informations concernant la période contractuelle « mois précédent » sous la forme d'un ensemble d'octets de même structure que celui décrit au chapitre précédent (« registres du mois en cours »).

2.6. Etat du compteur (code 03)

Ce tableau fournit les informations concernant l'état courant du compteur sous la forme d'un ensemble de 30 octets définis dans le tableau suivant.

Informations sur l'état du compteur

Groupe	Description de la donnée	Codage ou unité	Taille	Positionnement du 1er octet dans la chaîne
Date du relevé	Jour, mois, an (JJMMAA)	BCD	3 octets	1
Heure du relevé	Heure, minute, seconde (HHMMSS)	BCD	3 octets	4
TARIF	Informations tarifaires	(a)	1 octet	7
MODETA	(b)		1 octet	8
SOMMOD	(b)		2 octets	9
ERRFAT	Indicateur d'erreur du protocole	(c)	1 octet	11
ERRSES	Indicateur d'erreur du protocole	(c)	1 octet	12
PUISSANCES SOUSCRITES DU MOIS PROCHAIN	PSP (Heures de Pointe)	kW	2 octets	13
	PS HPH (Heures Pleines d'Hiver)	kW	2 octets	15
	PS HCH (Heures Creuses d'Hiver)	kW	2 octets	17
	PS HPE (Heures Pleines d'Eté)	kW	2 octets	19
	PS HCE (Heures Creuses d'Eté)	kW	2 octets	21
	PS PM (Heures de Pointe Mobile)	kW	2 octets	23
	PS HPD Heures Pleines de Demi-saison	kW	2 octets	25
	PS HCD (Heures Creuses de Demi-saison)	kW	2 octets	27
	PS CC (Heures Creuses de saison Creuse)	kW	2 octets	29

Précisions :

(a) pour plus de détails, se reporter au chapitre 2.4
(b) Informations techniques à l'usage exclusif d'Enedis
(c) pour plus de détails, se reporter au chapitre 3.5

2.7. Valeurs de référence (code 05)

Ce relevé se résume à la transmission par le compteur d'un ensemble de 256 octets dans le seul but de vérifier que le compteur est en mesure de transmettre des données au système de traitement informatique. La structure des données transmises suit une progression arithmétique de valeur initiale égale à 0 et de raison 1.

Les valeurs des octets, codées en hexadécimal, sont les suivantes :

- 1er octet : 00,
- 2ème octet : 01,
- 3ème octet : 02,
- .....
- 256ème octet : FF.

3. PARTIE 2 - Protocoles des couches de communication de la liaison téléphonique

3.1. Généralités sur les communications du télé-relevé

3.1.1. Structure des échanges

Le télé-relevé fait appel à une structure simplifiée d'échanges de données à trois couches : une couche ***Physique***, une couche ***Liaison*** et une couche ***Session***. En effet les couches « présentation », « réseau » et « transport » n'ont pas de raison d'être dans cette procédure de télé-relevé pour les raisons suivantes :

- l'absence de conversions de données importantes et systématiques à réaliser ;
- le système ou l'appareil de relevé automatique sont reliés au compteur par une simple liaison point à point ; la numérotation, la commutation du central téléphonique et le décrochage de l'appelé sont équivalents du point de vue logique à la phase d'établissement de la connexion physique ; une fois que cette connexion est établie, les informations transitent toutes par le même chemin et ne sont à aucun moment mémorisées dans un nœud intermédiaire.

Compte tenu des particularités du télé-relevé, la couche « application » telle que décrite dans la référence ISO 7498-1 se trouve intégrée ipso facto dans la couche session.
Les échanges au niveau de chaque couche sont décrits par des **transitions d'état** représentées sous forme de tableaux. La syntaxe utilisée pour la constitution de ces tableaux est définie par un **langage de spécification**. En cas de divergence d'interprétation entre le contenu d'un tableau de transitions d'état et une partie du texte d'accompagnement, il convient de considérer que le tableau constitue une référence qui prévaut sur le texte d'accompagnement.

Le réseau téléphonique commuté (RTC) est employé comme média de communication pour les échanges de données.

3.1.2. Vocabulaire de base et langage de spécification

Toute communication fait intervenir deux équipements représentés par les expressions « système **Appelant »** et « système **Appelé »**. L'Appelant est le système qui décide d'initialiser une communication avec un équipement distant dit **Appelé** ; ces dénominations restent valables pendant toute la durée de vie de la communication.
Une communication est décomposée en un certain nombre de transactions. Chaque transaction se traduit par une émission de l'**Emetteur** vers le **Récepteur**. Au gré de l'enchaînement des transactions, les systèmes Appelant et Appelé jouent tour à tour le rôle d'Emetteur et de Récepteur.
Les termes **Maître** et **Esclave** caractérisent l'utilisation des commandes au niveau session, ainsi qu'un rôle différencié en mode test et en mode écho. Le Maître est l'appelant, l'esclave l'appelé.

Pour décrire sans ambiguïté le rôle de chaque couche du protocole mis en œuvre pour les échanges de données, la spécification utilise un **formalisme en tableau** modélisant le comportement réel par un **automate** à nombre fini d'états.
A chaque automate, correspond un unique **tableau logique** qui peut éventuellement être représenté sous la forme de plusieurs **tableaux physiques**. Ce découpage se justifie lorsque le tableau logique est particulièrement conséquent.
A chaque **occurrence d'automate** correspond une **instance** (copie active distincte) du tableau logique de l'automate de référence.

Chaque tableau physique est composé de lignes appelées **lignes d'état**. Chaque ligne d'état décrit la **condition de déclenchement** (colonne 2) pour que la machine passe d'un **état initial** (colonne 1) à un **état final** (colonne 4) en exécutant un **ensemble d'actions** (colonne 3).
Le premier état initial est l'**état de démarrage** de l'automate. Cet état est unique ; il est particularisé au moyen de l'attribut souligné.

Un **état d'arrêt** de l'automate est un état final pour lequel aucune ligne d'état n'est définie avec cet état comme état initial. Un **automate** est **infini** lorsqu'il ne possède aucun état d'arrêt. Un **automate fini** peut posséder un ou plusieurs états d'arrêt. Ces états sont également représentés avec l'attribut souligné. Compte tenu de cette convention, l'ordre dans lequel sont présentés les états dans un tableau physique n'a aucune importance.
Cette même règle s'applique lorsque plusieurs lignes d'état référencent le même état initial car les conditions de déclenchement sont toujours **exclusives** les unes des autres. L'ordre des lignes d'un tableau physique n'est donc guidé que par

de simples considérations de présentation. Il est cependant logique de commencer par décrire les transitions de l'état de démarrage.
Un ensemble d'actions d'une ligne d'état doit être considéré comme une **section critique** (c'est-à-dire une séquence non interruptible). Les actions qui y sont décrites **doivent être exécutées dans l'ordre séquentiel où elles sont écrites**. Une action est définie par un appel à une **procédure nommée** instanciée avec une liste de zéro, un ou plusieurs **paramètres** entre parenthèses. Toute procédure nommée référencée doit faire l'objet d'une description séparée. Il existe cependant deux actions prédéfinies : **l'affectation** = et **l'action vide** $none() (absence d'action).

La condition de déclenchement associée à une ligne d'état peut éventuellement être composée de plusieurs sous-conditions. L'évaluation d'une condition de déclenchement composée passe toujours par l'évaluation de toutes les sous-conditions qu'elle contient. Ainsi, l'ordre d'écriture des sous-conditions est sans importance.

Les opérateurs supportés pour exprimer une condition composée sont d'une part les opérateurs logiques & (et logique), | (ou logique), not() (non logique) et d'autre part les opérateurs de comparaison (<, >, <=, >=, = et <>).
Il existe deux types de condition de déclenchement. Une condition de type simple est par définition évaluée instantanément. Elle peut éventuellement être composée mais, dans ce cas, toutes les sous-conditions sont de type simple. Une **fonction** nommée **booléenne** est un exemple de condition de type simple. Toute fonction nommée booléenne référencée doit faire l'objet d'une description séparée.

Une **condition de type événementiel** exprime l'attente d'un événement. Elle peut éventuellement être composée de plusieurs sous-conditions événementielles ou simples.

Lorsque l'évaluation d'une condition de déclenchement conduit à un résultat vrai, la condition se trouve **réalisée**. La réalisation d'une condition de déclenchement contient toujours une **transition d'état**. Un **événement** peut être défini comme étant un élément contribuant à la réalisation d'une condition de déclenchement de type événementiel.
Lorsqu'un événement est inclus dans une condition de déclenchement de type événementiel qui se trouve réalisée, il est automatiquement **consommé**. Un événement ne peut être consommé **qu'une seule fois**.

Tout événement, survenant lorsque l'occurrence d'automate qui est susceptible de le consommer se trouve dans un état où cette consommation est impossible, est stocké chronologiquement dans une zone appelée **file inter-automate**.
Ainsi, chaque automate dispose d'une unique file qu'il partage entre ses propres occurrences d'automate. La taille de cette file est supposée quasi-infinie ; son organisation et sa gestion ne sont pas décrites dans le présent document. Toutefois, il est indiqué qu'une **purge partielle** de la file (c'est-à-dire liée aux seuls événements concernant l'occurrence d'automate courante) est automatiquement effectuée pour toute transition d'état partant de l'état de démarrage.

Il doit exister également un mécanisme d'**auto-purge** conduisant à la suppression automatique des événements entrants et manifestement non consommables. En outre, il existe une procédure nommée prédéfinie $purge() qui correspond à l'action de **purge totale** de la file inter-automate courante. Toutes les occurrences de l'automate correspondant se retrouvent alors dans l'état de démarrage.

La **production** d'événements est assurée par certaines des actions décrites dans un ensemble d'actions associé à une ligne d'état. Un **événement interne** ne peut être consommé que par l'automate qui l'a produit. Un **événement externe** est toujours consommé par un autre automate que celui qui l'a produit.

A noter que **l'absence d'un événement,** exprimée par une sous-condition de type événementiel encapsulée dans l'opérateur logique not(), est toujours une sous-condition de type simple.

Lorsque, pour un état initial, il existe une ligne d'état où la condition de déclenchement est d'un certain type, alors toutes les lignes d'état du même état doivent posséder des conditions de déclenchement du même type.

Lorsque ce type est simple, l'état initial est appelé **sous-état**. Un sous-état est particularisé au moyen de l'attribut italique. Il est **transitoire** et peut toujours être éliminé. Sa présence dans un tableau physique n'est justifiée que par un souci de clarté de la présentation. Dans le cas particulier d'un sous-état de démarrage, une condition particulière a été prédéfinie ; il s'agit de la condition $true() qui est toujours vraie.

Enfin, les **variables** référencées dans les conditions de déclenchement et les actions décrites dans un tableau physique restent **locales** à chaque **occurrence d'automate**. Il existe également une variable prédéfinie (la **variable non liée**) destinée à remplacer tout paramètre inexploité dans n'importe quelle fonction ou procédure nommée.

USART : Universal Synchronous Asynchronous Receiver Transmitter (identifié à **Sérialisateur** dans cette spécification)

CRC : Cyclical Redundancy Check. Sa mise en oeuvre permet de détecter toutes les erreurs simples, doubles ou triples dans des blocs de données de 128 octets, ainsi que toutes les erreurs sur des suites de 16 bits

SPDU : Session Protocol Data Unit

SSDU : Session Service Data Unit

SPDU : Data Signal Display Unit

En l'absence de mention particulière concernant la valeur des octets, dans le cas des champs ou variables dont la taille dépasse un octet, les octets de poids le plus faible sont placés en tête.

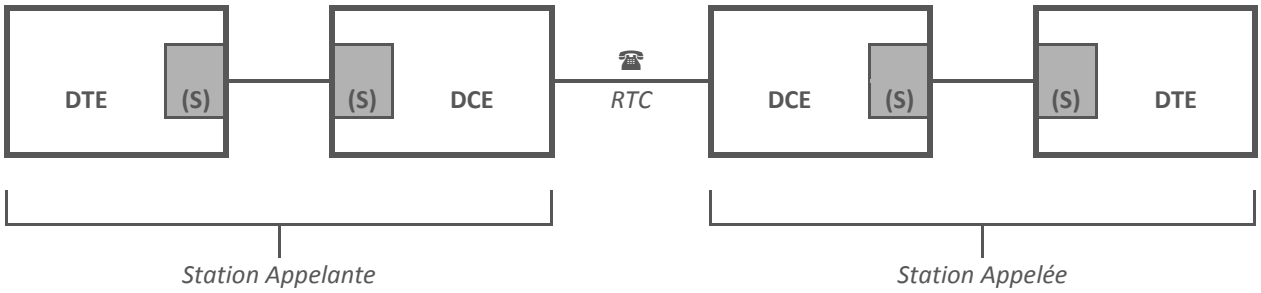
3.2. Couche Physique

3.2.1. Généralités sur la Couche Physique

Les équipements (compteurs et système de traitement informatique) échangent leurs données à travers le réseau téléphonique commuté public grâce à des modems semi-duplex. L'établissement de la communication se fait en appel/réponse automatique, conformément à l'avis V25 ou V25bis de l'Union Internationale des Télécommunications (UIT).

3.2.2. Description de l'environnement V24/V28

Un système communiquant sur réseau téléphonique commuté public peut-être modélisé comme indiqué sur la figure suivante :



DTE Data Terminal Equipment (**ordinateurs**, terminaux ou imprimantes). Un DTE est un système qui envoie ou reçoit des informations. Certains manuels le nomment aussi **ETTD** (Equipement Terminal de Traitement de Données).

DCE Data Communication Equipment (interface communicante ou **modem**). Un DCE est un équipement utilisé comme interface sur un réseau de communication. Son rôle est de transférer les données. Il est aussi nommé **ETCD** (Equipement de Terminaison du Circuit de Données).

(S) Un sérialisateur est généralement un circuit intégré qui synchronise l'émission et la réception de bits.
La fonction « transmetteur » convertit des octets en une suite de 8 bits en série. La fonction « récepteur » convertit une série de 8 bits en un octet.

Rappel des principaux circuits de l'interface V24

Circuit	Désignation	Abréviation	DTE/DCE *
	Terre		
101	Terre de Protection	TP	-
102	Terre de Signalisation	TS	-
	Signaux de données		
103	Emission des Données	ED	S/E
104	Réception des Données	RD	E/S
	Signaux de contrôle		
105	Demande Pour Emettre	DPE	S/E
106	Prêt A Emettre	PAE	E/S
107	Poste de Données Prêt (modem prêt)	PDP	E/S
108	Terminal de Données Prêt	TDP	S/E
	Autres signaux de contrôle		
109	Détection de Signal	DS	E/S
125	Indicateur de Sonnerie	IS	E/S
	Signaux d'horloge		
113	horloge émetteur (DTE source)	-	S/E
114	horloge émetteur (DCE source)	-	E/S
115	horloge récepteur (DCE source)	-	E/S

Notes :

S ≡ le circuit défini est une sortie

E ≡ le circuit défini est une entrée

3.2.3. Protocole de la couche *Physique*

Le protocole de la couche ***Physique*** est conçu pour un fonctionnement asynchrone (étant donné le faible débit) en mode semi-duplex (une ligne bifilaire et une seule fréquence porteuse). Il est identique pour l'Appelant et pour l'Appelé (comportement totalement symétrique).

Ce protocole comprend :

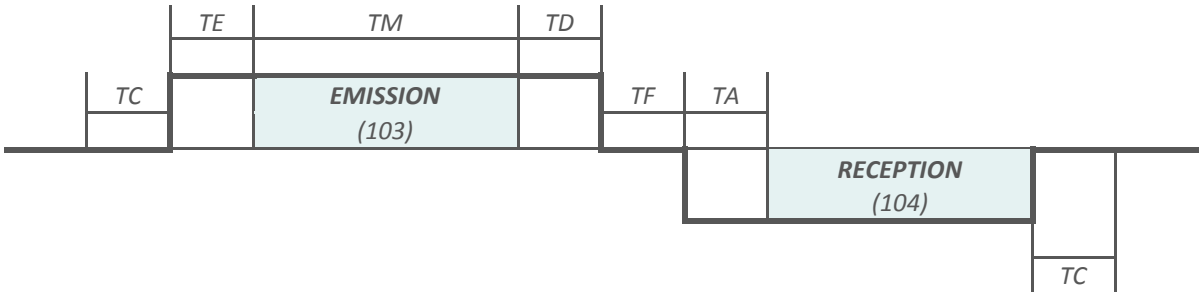
- une interface de communication conforme aux avis V24 et V28 de l'UIT ainsi qu'à la norme ISO 2110 ;
- un échange systématique de jeton par échange de porteuse.

Le protocole de la couche ***Physique*** agit directement sur le sérialisateur relié au modem. Le terminal et le modem communiquent à travers l'interface V24 avec un débit d'échange de 1200 bits/s.

Le modus operandi de la transmission asynchrone consiste en un regroupement par paquets de 8 des bits transmis, sans parité (octets non codés en ASCII). Chaque caractère comprend un bit de *start* en tête et un bit de *stop* en queue, qui correspondent respectivement à une mise à 0 et une mise à 1 du signal de données en sortie du sérialisateur. Ces deux bits font office de "points de synchronisation". Le moment *start* permet essentiellement de synchroniser les horloges alors que le moment *stop* laisse un temps suffisant au récepteur pour mettre en parallèle le caractère avant de traiter le suivant.

3.2.4. Diagramme des temps

La figure suivante schématise les temporisations et les réveils utilisés pour la gestion des échanges en régime permanent :



Chaque trame est concrétisée par l'émission d'une porteuse non modulée d'une durée minimum TE, suivie par la transmission d'un bloc de données d'une durée TM bornée et d'une temporisation TD avant la coupure de la porteuse (afin d'éviter tout risque de perte du dernier octet émis).

TC : temps de coupure de porteuse à l'alternat. Ce temps est contrôlé par un réveil.
TC = 40 ms.

TE : durée d'émission de la porteuse non modulée. Cette durée est une temporisation.
TE = 300 ms.

TM : durée de modulation pour l'émission des données. Cette durée dépend de la taille de la trame à émettre (comprise entre 4 et 127 octets) et de la vitesse de transmission.

TD : durée d'attente avant la coupure de la porteuse. Cette durée est une temporisation équivalente au minimum à la transmission de 2 octets.
TD = 20 ms.

TF : durée de sécurité entre l'émission et la réception. Cette durée est une temporisation.
TF = 150 ms.

TA : durée d'attente de stabilisation de la porteuse après détection du signal DCD. Cette durée est une temporisation qui permet d'éliminer les caractères parasites.
TA = 60 ms.

Il existe également quatre réveils (TB, TR, TR1 et TCM) non représentés sur cette figure.
TB : temps de réception maximum admissible. Ce temps est contrôlé par un réveil armé du côté récepteur lorsque la porteuse est détectée et désarmé au passage en émission.
TB = 2 s.

TR : délai d'attente maximum de retour de la porteuse après une émission. Ce temps est contrôlé par un réveil.
TR = 500 ms. La valeur de TR a été spécifiée initialement à 200 ms, mais c'est la valeur 500 ms qui est utilisée par les applications existantes.

TR1 : temps de réponse maximum admissible entre d'une part la montée du signal 105 par le DTE et la montée du signal 106 par le DCE, et d'autre part la descente du signal 108 par le DTE et la descente du signal 107 par le DCE. Ce temps est contrôlé par un réveil.
TR1 = 1 s.

TCM : durée maximale d'une communication. Ce temps est contrôlé par un réveil.
TCM = 10 minutes.

3.2.5. Services et primitives de service de la couche Physique

L'utilisateur du protocole de la couche ***Physique*** dispose des services et primitives de service suivants :

Services	Primitives de service
Phy_DATA	Phy_DATA.req(Frame) Phy_EndDATA.ind(Frame) Phy_DATA.ind(Frame, ErrpliP)
Phy_ABORT	Phy_ABORT.req() Phy_ABORT.ind(ErrorNb)

Le rôle attribué à chaque primitive est le suivant :

- Phy_DATA.req(Frame) permet à la couche Liaison de demander à la couche Physique l'émission d'une trame Frame ;
- Phy_EndDATA.ind() permet à la couche Physique d'informer la couche Liaison de la fin de l'émission de la dernière trame ;
- Phy_DATA.ind(Frame, ErrpliP) permet à la couche Physique d'informer la couche Liaison qu'une trame Frame est disponible, avec éventuellement une erreur détectée à la réception (ErrpliP différent de 0) ;
- Phy_ABORT.req() permet à la couche Liaison de demander à la couche Physique de mettre fin à son activité ;
- Phy_ABORT.ind(ErrorNb) permet à la couche Physique d'informer la couche Liaison de l'occurrence d'une erreur fatale repérée par le numéro ErrorNb.

3.2.6. Paramètres de la couche Physique

Les valeurs des temporisations et des réveils ont déjà été définies dans le diagramme des temps.
La valeur de la taille maximale **MaxIndex** d'une trame en réception est fixée à 127.
Le nombre maximum d'émissions successives sans réponse **MaxErrpj** est fixé à 7.

3.2.7. Transitions d'état

La machine d'état de l'Appelant est strictement identique à celle de l'Appelé. Les temps sont exprimés en millisecondes dans le tableau suivant (les 2 autres tableaux qui lui font suite donnent respectivement les définitions des procédures « ensemble d'actions » et la signification du terme « Etat» utilisé dans la première colonne et la dernière colonne de ce tableau).

Transitions d'état décrivant le protocole de la couche Physique

Etat initial	Cond. de déclenchement	Ensemble d'actions	Etat final
Init	$true()	MaxErrpj = 7 MaxIndex =127 TD = 20 TC = 40 TA = 60 TF = 150 TR = 200 TE = 300 TR1 = 1000 TB = 2000 TCM = 600000	Stopped
Stopped	Connection_event & Master	Errpj = 0 ErrpliP = 0 init_timer(TCM) set(RTS) init_timer(TR1)	W.CTS
Stopped	Connection_event & not(Master)	Errpj = 0 ErrpliP = 0 init_timer(TCM) init_timer(TR) wait_time(TF)	W.DCD
W.CTS	CTS_activation_event	stop_timer(TR1) wait_time(TE)	*M.Send*
W.CTS	Phy_ABORT.req()	stop_timer(TCM) stop_timer(TR1) clear(RTS) clear(DTR) init_timer(TR1)	W.DSR
W.CTS	time_out(TCM)	Phy_ABORT.ind(EP-R7F) stop_timer(TR1) clear(RTS) clear(DTR) init_timer(TR1)	W.DSR
W.CTS	time_out(TR1)	Phy_ABORT.ind(EP-R6F) stop_timer(TCM) clear(RTS) clear(DTR) init_timer(TR1)	W.DSR
W.DCD	DCD_activation_event	stop_timer(TR) init_timer(TB) Errpj = 0 Index = 1 Frame="" Veloc = FAUX wait_time(TA)	Receiving

W.DCD	time_out(TR) & Errpj < 6	Errpj = Errpj + 1 set(RTS) init_timer(TR1)	W.CTS
W.DCD	time_out(TR) & Errpj >= 6	Phy_ABORT.ind(EP-R5F) stop_timer(TCM) clear(DTR) init_timer(TR1)	W.DSR
W.DCD	Phy_ABORT.req()	stop_timer(TR) stop_timer(TCM) clear(DTR) init_timer(TR1)	W.DSR
W.DCD	time_out(TCM)	Phy_ABORT.ind(EP-R7F) stop_timer(TR) clear(DTR) init_timer(TR1)	W.DSR
M.Send	Phy_Data.req(Frame)	Size = size(Frame) Index=1 send_octet(Frame, Index)	Sending
M.Send	not(Phy_Data.req(Frame))	clear(RTS) init_timer(TR) wait_time(TF)	W.DCD
Sending	octet_sent_event & Index < Size	Index=Index+1 send_octet(Frame, Index)	Sending
Sending	octet_sent_event & Index >= Size	wait_time(TD) Phy_EndData.ind() clear(RTS) init_timer(TR) wait_time(TF)	W.DCD
Sending	Phy_ABORT.req()	stop_timer(TCM) clear(DTR) init_timer(TR1)	W.DSR
Sending	time_out(TCM)	Phy_ABORT.ind(EP-R7F) clear(DTR) init_timer(TR1)	W.DSR
Receiving	octet_received_event & Index <= MaxIndex & Veloc =FALSE	Index = Index +1 read_data(RecB) concat(Frame, RecB)	Receiving
Receiving	octet_received_event & Index > MaxIndex & Veloc =FALSE	read_data(RecB) Veloc =TRUE	Receiving
Receiving	octet_received_event & Veloc =TRUE	read_data(RecB)	Receiving
Receiving	DCD_inactivation_event	init_timer(TC) Veloc = TRUE	M.RecC

Receiving	time_out(TB)	Phy_ABORT.ind(EP-R4F) stop_timer(TCM) clear(DTR) init_timer(TR1)	W.DSR
Receiving	Phy_ABORT.req()	stop_timer(TB) stop_timer(TCM) clear(DTR) init_timer(TR1)	W.DSR
Receiving	time_out(TCM)	Phy_ABORT.ind(EP-R7F) stop_timer(TB) clear(DTR) init_timer(TR1)	W.DSR
Receiving	err_USART_evt	ErrpliP = ErrpliP + EP_R1 Veloc =TRUE	Receiving
M.RecC	DCD_activation_event & not(time_out(TC))	stop_timer(TC)	Receiving
M.RecC	time_out(TC) & Index < 4	stop_timer(TB) set(RTS) init_timer(TR1) ErrpliP = 0	W.CTS
M.RecC	time_out(TC) & Index >= 4	Phy_DATA.ind(Frame, ErrpliP) stop_timer(TB) set(RTS) init_timer(TR1) ErrpliP = 0	W.CTS
M.RecC	Phy_ABORT.req()	stop_timer(TC) stop_timer(TB) stop_timer(TCM) clear(DTR) init_timer(TR1)	W.DSR
M.RecC	time_out(TCM)	Phy_ABORT.ind(EP-R7F) stop_timer(TC) stop_timer(TB) clear(DTR) init_timer(TR1)	W.DSR
M.RecC	time_out(TB)	Phy_ABORT.ind(EP-R4F) stop_timer(TC) stop_timer(TCM) clear(DTR) init_timer(TR1)	W.DSR
W.DSR	DSR_inactivation_event	stop_timer(TR1) set(DTR)	Stopped
W.DSR	time_out(TR1)	Set(DTR)	Stopped

Définition des procédures, des fonctions et des événements classés dans l'ordre alphabétique

Procédure, fonction ou événement	Définition
clear(RTS)	Désactivation du circuit 105 du modem
clear(DTR)	Désactivation du circuit 108 du modem
concat(Frame, RecB)	concaténation de l'octet RecB dans la trame Frame en cours de constitution
connection_event	événement signalant que la communication téléphonique est établie
CTS_activation_event	événement issu du modem informant de l'activation du circuit 106
DCD_activation_event	événement issu du modem informant de l'activation du circuit 109
DCD_inactivation_event	événement issu du modem informant de la désactivation du circuit 109
DSR_inactivation_event	événement issu du modem informant de la désactivation du circuit 107
err_USART_evt	événement issu de l'USART signalant un problème
(Phy_Data.req(Frame))	consommation d'un événement de type Phy_DATA.req(Frame)
init_timer(TB), init_timer(TR), init_timer(TR1) ou init_timer(TC), init_timer(TCM)	armement du réveil TB, TR, TR1, TC ou TCM
octet_received_event	événement informant qu'un octet a été reçu
octet_sent_event	événement informant qu'un octet a été émis
read_data(RecB)	traitement de l'événement octet_received_event par lecture de l'octet RecB reçu
send_octet(Frame, Index)	Emission de l'octet de rang Index dans la trame Frame
set(RTS)	activation du circuit 105 du modem
set(DTR)	activation du circuit 108 du modem
size(Frame)	calcul du nombre d'octet de la trame Frame
stop_timer(TB), stop_timer(TR), stop_timer(TR1) stop_timer(TC) ou stop_timer(TCM)	désarmement du réveil TB, TR, TR1 TC ou TCM
time_out(TB), time_out(TR), time_out(TR1) time_out(TC), ou time_out(TCM)	déclenchement du réveil TB, TR, TR1, TC ou TCM
wait_time(TA), wait_time(TE), wait_time(TD) ou wait_time(TF)	temporisation pendant le temps TA, TE, TD ou TF

Précisions concernant les fonctions wait_time( ) :

Pendant les attentes des fonctions wait_time( ), les événements affectant la liaison série ne sont pas traités, mais sont mémorisés. Ils sont traités (ou rejetés) lors des traitements ultérieurs. Dans une approche « multi-threads », il est possible d'implémenter ces attentes sous la forme d'un « blocage » de l'automate.

Précisions concernant la variable VELOC :

Cette variable a été définie dans la spécification initiale et conservée pour des raisons historiques. Le rôle de cette variable VELOC est de verrouiller l'accès au buffer de réception de la couche physique lorsque l'état est incompatible avec la réception de caractères :

- la porteuse n'est pas présente (DCD absent),
- le nombre de caractères reçus est supérieur à la capacité du buffer (il y a des parasites, ou un problème de fonctionnement important, ...),
- un caractère a fait l'objet d'une signalisation d'erreur (ou le driver série).

Cette variable est mise à l'état "FAUX" lors de la détection de porteuse et du passage en réception. Il convient de rester ensuite en réception jusqu'à la détection de la chute de cette porteuse, et VELOC peut alors être mis à l'état "VRAI", si une erreur est signalée par l'UART ou si le nombre maximum de caractères a été reçu. Les caractères ne sont mémorisés que si VELOC est à l'état "VRAI" (et que l'on est dans l'état "RECEIVING").

Remarque : il est possible, pour éliminer cette variable, d'introduire un état "REC_ERR", dans lequel l'automate passe en cas de détection d'une anomalie.

Signification des « états finaux ».

Etat	Signification
Init	état d'initialisation
Stopped	état de démarrage commun à l'Appelant et à l'Appelé
W.CTS (Wait for Clear To Send)	état de l'Emetteur en attente de la montée du circuit 106 après la montée du circuit 105, sous le contrôle du réveil TR1
W.DCD (Wait for Data Carrier Detect)	état d'attente de la montée du circuit 109, sous le contrôle du réveil TR
W.DSR (Wait for Data Set Ready)	état d'attente de la descente du circuit 107 sous le contrôle du réveil TR1
M.Send (Must Send)	état transitoire de l'Emetteur, pour tester s'il y a une trame à envoyer
Sending	état récurrent de l'Emetteur : émission d'un octet à la fois
Receiving	état du Récepteur. Pour rester dans cet état, le circuit 109 doit être monté en permanence
M.RecC (Must Receive but DCD Cut)	descente du circuit 109 en cours de réception

3.2.8. Répertoire et traitement des erreurs

Les erreurs sont répertoriées par le codage **EP**-**RNF** ou **EP**-**ENF** portant les significations suivantes :

EP erreur de la couche ***Physique,***
-R erreur en réception,
-E erreur en émission,
N numéro de l'erreur,
F erreur fatale.

Tableau récapitulatif des erreurs

EP-R1	Erreur matériel détectée au niveau de l'USART
	Cette erreur conduit la couche ***Physique*** à en informer la couche ***Liaison*** et à ne pas mémoriser les octets suivants de la trame en cours
EP-R4F	Time-out bavard : durée de réception trop longue
	Cette erreur conduit à réinitialiser la couche ***Physique*** après avoir informé la couche ***Liaison***
EP-R5F	7 émissions successives sans réponse (7 jetons ou 1 de données et 6 jetons)
	Cette erreur conduit à réinitialiser la couche ***Physique*** après avoir informé la couche ***Liaison***
EP-R6F	Blocage du modem constaté après l'échéance du réveil TR1
	Cette erreur conduit à réinitialiser la couche ***Physique*** après avoir informé la couche ***Liaison***
EP-R7F	Echéance du réveil TOCM
	Cette erreur conduit à réinitialiser la couche ***Physique*** après avoir informé la couche ***Liaison***

Les erreurs non fatales sont remontées localement grâce à la primitive DL_DATA.ind. Toute occurrence de l'une des erreurs fatales est remontée localement grâce à la primitive de service Phy_ABORT.ind. La liste complète des numéros d'erreur fatale est fournie au chapitre 3.5.

3.3. Couche Liaison

3.3.1. Généralités sur la Couche Liaison

Le protocole de la couche ***Liaison*** est conçu pour supporter une transmission bidirectionnelle des données.
Comme cette couche assure le contrôle du transfert correct des données, son fonctionnement en 2 modes (en mode ECHO et en mode normal) est spécifié dans les sous-chapitres suivants.

3.3.2. Couche Liaison en mode normal

3.3.2.1. Protocole Liaison en mode normal

En mode normal, le protocole est strictement identique pour l'Appelant (le Maître) et pour l'Appelé (l'esclave).

La couche ***Liaison*** en mode normal est chargée de transformer le canal physique exploité par la couche ***Physique*** en canal logique apte à transmettre l'information avec fiabilité. Ses fonctions principales sont :

- synchroniser les trames en émission et en réception ;
- assurer une protection efficace contre les erreurs de transmission.

La couche ***Liaison*** doit régler tous les problèmes courants causés par les défauts du canal physique à l'exception de la coupure matérielle de la liaison.
La communication entre stations peut être divisée en trois phases :

- l'établissement du circuit réalise la connexion physique des stations au réseau téléphonique public commuté. Cette fonction est assurée par Physique1, non décrit dans ce document ;
- le transfert d'information est la phase utile de la transmission avec échange de trames d'information et d'acquittement ;
- la terminaison du circuit libère la ligne téléphonique. Cette fonction est également assurée par Physique1.

On notera que ne sont pas évoquées ici, les phases d'ouverture de la liaison (reconnaissance des stations et négociation des conditions de l'échange) et de fermeture de la liaison (libération des stations). Le protocole ***Liaison*** est donc **orienté sans connexion**.

Deux variables globales, TestS et TestM, dont les valeurs sont gérées en dehors du protocole, permettent de définir un traitement des trames différent lorsque le mode test est opérationnel. Sur le Maître, TestM est à VRAI si l'équipement est en mode test, à FAUX sinon ; TestS est toujours à FAUX. Sur l'esclave, TestS est à VRAI si l'équipement est en mode test, à FAUX sinon ; TestM est toujours à faux.
Les traitements à effectuer en mode test sont les suivants :

- sur la station Esclave, l'octet Errpli est ajouté aux données, dans le champ Test, avant le calcul de CRC et l'envoi de la trame ;
- sur la station Maître, le champ Test est extrait et placé dans la variable CodeTest.

3.3.2.2. Procédure "envoyer et attendre"

Le traitement des défauts de ligne est effectué suivant la procédure "envoyer et attendre". Voici le mode de fonctionnement de cette procédure :

- à toute trame devant être émise, sont ajoutés deux octets de CRC (Cyclical Redundancy Check) ;
- toute trame reçue fait l'objet d'un contrôle de CRC. Si la trame est valide, un acquittement positif est envoyé. Dans le cas contraire, c'est un acquittement négatif qui est émis ;
- un réveil est armé à chaque envoi. Son échéance sans réception d'acquittement est équivalente à la réception d'un acquittement négatif ;
- pour éviter de possibles duplications suite à des retransmissions, le principe de la numérotation des trames est retenu.

3.3.2.3. Services et primitives de service de Liaison

L'utilisateur du protocole ***Liaison*** dispose des services et primitives de service suivants :

Service	Primitive
DL_DATA	DL_DATA.req(DSDU) DL_EndDATA.ind() DL_DATA.ind(DSDU)
DL_ABORT	DL_ABORT.req() DL_ABORT.ind(ErrorNb)

Voici le rôle attribué à chaque primitive :

- DL_DATA.req(DSDU) permet à la couche Session de demander à la couche Liaison le transfert d'un paquet de données DSDU ;
- DL_EndDATA.ind() permet à la couche Liaison d'informer la couche Sessiont du bon envoi du dernier paquet de données ;
- DL_DATA.ind(DSDU) permet à la couche Liaison d'informer la couche Session de l'arrivée d'un paquet de données DSDU ;
- DL_ABORT.req() permet à la couche Sessiont de demander à la couche Liaison de mettre fin à son activité ;
- DL_ABORT.ind(ErrorNb) permet à la couche Liaison d'informer la couche Session de l'occurrence d'une erreur fatale repérée par le numéro ErrorNb.

3.3.2.4. Description des trames

Le protocole ***Liaison*** est une procédure de ligne orientée bit c'est-à-dire qu'il dispose l'information dans des trames dont les champs contiennent le texte (à savoir les données en provenance de la couche ***session***) ainsi que les bits de contrôle associés. Les trames contiennent un nombre entier d'octets inférieur ou égal à 127. Comme tous les champs ont une taille fixe sauf le champ ***Text***, il suffit de disposer d'un champ ***Size*** pour définir la longueur réelle de chaque trame.

Dans le protocole ***Liaison***, il n'y a qu'un seul type de trame défini par les six champs suivants :

- Size : 1 octet,
- FrameType : 4 bits,
- NSEQ : 4 bits,
- Text (DSDU) : 0 à 122 octets,
- Test : 1 octet si présent (en mode test),
- BCC : 2 octets.

1 octet	4 bits	4 bits	0 à 123 octets		2 octets
Size	Type	NSEQ	Text	Test	BCC

Le champ ***Size*** correspond au nombre total d'octets dans la trame.

Le champ ***Type*** est codé "0000" pour une trame de données, "0110" pour un acquittement positif, "1011" pour un acquittement négatif.

Le champ ***NSEQ*** référence le numéro de la séquence en cours ; une séquence correspond à l'échange d'une trame de données et de l'acquittement en réponse. Il varie de 0 à 15.

Le champ ***Text*** n'est présent que dans les trames de données.

Le champ ***Test*** n'est présent qu'en mode test, dans le sens Esclave-Maître.

Enfin, le champ ***BCC*** (Block Check Characters) correspond aux 16 bits de redondance du CRC dont le principe est exposé au chapitre 3.6.

3.3.2.5. Gestion des échanges

Le protocole ***Liaison*** est symétrique, à part le traitement du mode test. Les systèmes Appelant et Appelé sont tour à tour Emetteur et Récepteur (cette dernière terminologie est la plus représentative de la gestion des échanges).
Après l'envoi d'une trame de donnée, la couche ***Liaison*** du côté Emetteur attend toujours une trame de contrôle venant de la couche ***Liaison*** du Récepteur. Cette attente est contrôlée par le réveil **TL**. L'échéance de TL est interprétée comme la réception d'un acquittement négatif.

Après l'envoi d'une trame de donnée et la réception d'un acquittement négatif de la trame précédemment envoyée, il y a ré-émission de la trame courante. Le nombre de ré-émissions est limité à MaxErrl. Au-delà de ce nombre, il y a arrêt de la communication au niveau ***Liaison*** et la couche ***Session*** en est informée.

A chaque réception d'une trame de donnée numérotée N par un des systèmes, il y a émission immédiate d'une trame de contrôle en réponse, portant le même numéro N.

Après l'échange d'une trame de données numérotée N, acquittée positivement, la prochaine trame de données portera le numéro N+1, quel que soit le sens de son échange.

L'alternance du sens d'échange des trames de données s'opère de façon quelconque.

3.3.2.6. Paramètres de Liaison

Le temps d'attente maximum TL, par l'Emetteur, de la trame de contrôle en retour du Récepteur est calculé sur la base de 6 pertes de jeton :

- TL > 5(TR + TE) + TR = 2700ms d'où TL fixé à 3 s.

La valeur du nombre MaxErrl d'envois d'une même trame provoquant la déconnexion est fixée à 8 ; le huitième envoi n'est pas effectué.

3.3.2.7. Transitions d'état

La machine d'état de l'Appelant est strictement identique à celle de l'Appelé. Les deux systèmes sont tour à tour Emetteur et Récepteur de DSDU. A tout instant, il n'existe qu'une seule occurrence de cet automate dans chaque équipement. Les temps sont exprimés en millisecondes.

Transitions d'état de Liaison

Etat initial	Cond. de déclenchement	Ensemble d'actions	Etat final
Init	$true()	MaxErrl = 8 Errl = 1 Vseq = 1 Ack_expected= FALSE Errpli = 0 CodeTest = 0 TL = 3300	Stopped
Stopped	DL_DATA.req(DSDU) &	Type="0000"B Text=DSDU Size=size(Text) + 4 Fr=concat(Size, Type, VSeq, Text,_) Phy_DATA.req(concat_Fr(Fr, crc(Fr))) Ack_expected=TRUE Errl = Errl + 1	M.RecPE
Stopped	Phy_DATA.ind(Frame, ErrpliP) & not(check_frame(Frame, ErrpliP))	Errpli = Errpli + ErrpliP + EL_R1 Type = "1011"B	*M.SendCr*
Stopped	Phy_DATA.ind(Frame, ErrpliP) & check_frame(Frame, ErrpliP) & is_text(Frame) & (Nseq = Vseq I Nseq = Vseq - 1)	$none()	*T.NbD*
M.RecPE	Phy_EndDATA.ind() & Ack_expected=TRUE	Errpli = 0 Init_timer(TL)	M.RecCr
M.RecPE	Phy_EndDATA.ind() & Ack_expected=FALSE & Type = "0110" & not(check_ELR2(Errpli))	DL_DATA.ind(extract_text (Frame) Errpli = 0 Vseq = Vseq + 1	Idle
M.RecPE	Phy_EndDATA.ind() & Ack_expected=FALSE & Type = "0110"& check_ELR2(Errpli)	Errpli = 0 Vseq = Vseq + 1	Idle
M.RecPE	Phy_EndDATA.ind() & Ack_expected=FALSE & Type = "1011"	Errpli = 0	Idle
M.RecPE	DL_ABORT.req()	Errl = 1 Vseq = 1 Ack_expected= FALSE Phy_ABORT.req()	Stopped
M.RecPE	Phy_ABORT.ind(ErrorNb)	Errl = 1 Vseq = 1 Ack_expected= FALSE DL_ABORT.ind(ErrorNb)	Stopped
M.RecCr	Phy_DATA.ind(Frame, ErrpliP) & check_frame(Frame, ErrpliP) & not(is_nack(Frame))	stop_timer(TL) extract_test(Frame, TestM, CodeTest)	*T.NbCr*
M.RecCr	Phy_DATA.ind(Frame, ErrpliP) & check_frame(Frame, ErrpliP) & is_nack(Frame) & Errl < MaxErrl	stop_timer(TL) extract_test(Frame, TestM, CodeTest) Phy_DATA.req(Fr) Errl=Errl +1	M.RecPE

M.RecCr	Phy_DATA.ind(Frame, ErrpliP) & check_frame(Frame, ErrpliP)) & is_nack(Frame) & Errl >= MaxErrl	Errl = 1 Vseq = 1 Ack_expected= FALSE stop_timer(TL) extract_test(Frame, TestM, CodeTest) DL_ABORT.ind(EL-E4F) Phy_ABORT.req()	Stopped
M.RecCr	Phy_DATA.ind(Frame, ErrpliP) & not(check_frame(Frame, ErrpliP)) & Errl < MaxErrl	Errpli = Errpli + EL_R1 stop_timer(TL) Phy_DATA.req(Fr) Errl=Errl +1	M.RecPE
M.RecCr	Phy_DATA.ind(Frame, ErrpliP) & not(check_frame(Frame, ErrpliP)) & Errl >= MaxErrl	Errl = 1 Vseq = 1 Ack_expected= FALSE stop_timer(TL) DL_ABORT.ind(EL-E4F) Phy_ABORT.req()	Stopped
M.RecCr	time_out(TL) & Errl < MaxErrl	Errpli = Errpli + EL_E1 Phy_DATA.req(Fr) Errl=Errl +1	M.RecPE
M.RecCr	time_out(TL) & Errl >= MaxErrl	Errl = 1 Vseq = 1 Ack_expected= FALSE DL_ABORT.ind(EL-E4F) Phy_ABORT.req()	Stopped
M.RecCr	DL_ABORT.req()	Errl = 1 Vseq = 1 Ack_expected= FALSE stop_timer(TL) Phy_ABORT.req()	Stopped
M.RecCr	Phy_ABORT.ind(ErrorNb)	Errl = 1 Vseq = 1 Ack_expected= FALSE stop_timer(TL) DL_ABORT.ind(ErrorNb)	Stopped
Idle	DL_DATA.req(DSDU) & not(TestS)	Type="0000"B Text=DSDU Size=size(Text) + 4 Fr=concat(Size, Type, VSeq, Text,_) Phy_DATA.req(concat_Fr(Fr, crc(Fr))) Ack_expected=TRUE Errl = Errl + 1	M.RecPE
Idle	DL_DATA.req(DSDU) &TestS	Type="0000"B Text=DSDU Size=size(Text) + 5 Fr=concat(Size, Type, VSeq, Text, Errpli) Phy_DATA.req(concat_Fr(Fr, crc(Fr))) Ack_expected=TRUE Errl = Errl + 1	M.RecPE
Idle	Phy_DATA.ind(Frame, ErrpliP) & not(check_frame(Frame, ErrpliP))	Errpli = Errpli + ErrpliP + EL_R1 Type="1011"B	*M.SendCr*
Idle	Phy_DATA.ind(Frame, ErrpliP) & check_frame(Frame, ErrpliP) & is_text(Frame) & (Nseq = Vseq I Nseq= Vseq - 1)	extract_test(Frame, TestM, CodeTest)	*T.NbD*

Idle	Phy_DATA.ind(Frame, ErrpliP) & check_frame(Frame, ErrpliP) & (not(is_text(Frame)) I (is_text(Frame) & (Nseq<>Vseq) & (Nseq<> Vseq - 1)))	Errl = 1 Vseq = 1 Ack_expected= FALSE extract_test(Frame, TestM, CodeTest) DL_ABORT.ind(EL- R3F) Phy_ABORT.req()	Stopped
Idle	DL_ABORT.req()	Errl = 1 Vseq = 1 Ack_expected= FALSE Phy_ABORT.req()	Stopped
Idle	Phy_ABORT.ind(ErrorNb)	Errl = 1 Vseq = 1 Ack_expected= FALSE DL_ABORT.ind(ErrorNb)	Stopped
M.SendCr	not(TestS)	Size= 4 Fr=concat(Size, Type, VSeq, _,_) Phy_DATA.req(concat_Fr(Fr, crc(Fr)))	M.RecPE
M.SendCr	TestS	Size= 5 Fr=concat(Size, Type, VSeq, _, Errpli) Phy_DATA.req(concat_Fr(Fr, crc(Fr)))	M.RecPE
T.NbCr	is_ack(Frame) & (Nseq = Vseq)	Errl = 1 Vseq = Vseq + 1 DL_EndDATA.ind() Ack_expected= FALSE	Idle
T.NbCr	is_ack(Frame) & Nseq <> Vseq	Errl = 1 Vseq = 1 Ack_expected= FALSE DL_ABORT.ind(EL-E5F) Phy_ABORT.req()	Stopped
T.NbCr	is_text(Frame) & (Nseq = Vseq - 1) & Errl < MaxErrl	Phy_DATA.req(Fr) Errl=Errl +1 Errpli = Errpli + EL_E3	M.RecPE
T.NbCr	is_text(Frame) & (Nseq = Vseq - 1) & Errl = MaxErrl	Errl = 1 Vseq = 1 Ack_expected= FALSE DL_ABORT.ind(EL-E4F) Phy_ABORT.req()	Stopped
T.NbCr	is_text(Frame) & (Nseq = Vseq + 1)	Errpli = Errpli + EL_E2 Errl = 1 Vseq = Vseq + 1 DL_EndDATA.ind() Ack_expected= FALSE	Idle
T.NbCr	is_text(Frame) & (Nseq <> Vseq + 1) & (Nseq <> Vseq -1)	Errl = 1 Vseq = 1 Ack_expected= FALSE DL_ABORT.ind(EL- E5F) Phy_ABORT.req()	Stopped
T.NbD	Nseq = Vseq	$none()	*M.SendCr*
T.NbD	Nseq = Vseq – 1	Errpli = Errpli + EL_R2 Vseq = Vseq - 1	*M.SendCr*

Signification des états mentionnés dans le tableau précédent

Etat	Signification
Init	état d'initialisation
Stopped	état de démarrage commun à l'Appelant et à l'Appelé
M.RecPE (Must Receive Phy_Enddata.ind)	état de l'Emetteur : attente de l'événement généré par le protocole physique signalant la fin de l'envoi de la dernière trame
M.RecCr (Must Receive frame ContRol)	état du Récepteur : sous le contrôle du réveil TL, attente d'une trame de contrôle en provenance de l'Emetteur
Idle	état opérationnel d'attente de réception d'une trame ou d'une demande d'envoi de données
M.SendCr (Must Send frame ContRol)	état de l'Emetteur : envoi d'une trame de contrôle
T.NbCr (Test sequence NumBer of a frame ContRol)	Vérification du numéro de séquence d'une trame de contrôle
T.NbD (Test sequence NumBer of a Data frame)	Vérification du numéro de séquence d'une trame de données

Définition des procédures et des fonctions classées dans l'ordre alphabétique

Procédure ou fonction	Définition
check_frame(Frame, ErrpliP)	vérification que la trame Frame reçue est correcte : ■ CRC correct, ■ type de trame correct, ■ nombre d'octets compatible avec le champ Size, ■ ErrpliP = 0.
check_ELR2(Errpli)	vérification que le bit d'erreur EL_R2 a été mis dans Errpli
Fr = concat(Size, Type, Vseq, Text,Errpli)	construction d'une chaîne binaire Fr par concaténation de plusieurs paramètres
concat(Fr,crc(Fr))	construction d'une chaîne binaire par concaténation du premier paramètre avec le deuxième paramètre ;
crc(OctetString)	calcul du CRC de la chaîne d'octets OctetString
exist_dl_data_req(DL_DATA.req(DSDU))	consommation d'un événement de type DL_DATA.req(DSDU)
extract_test(Frame, TestM, CodeTest)	extraction du champ Test de la trame Frame et recopie dans CodeTest, si TestM est à VRAI
extract_text(Frame)	extraction du champ Text de la trame Frame
init_timer(TL)	armement du réveil TL
Is_ack(Frame)	vérification que la trame Frame reçue est un acquittement positif
Is_nack(Frame)	vérification que la trame Frame reçue est un acquittement négatif
Is_text(Frame)	vérification que la trame Frame reçue est une trame de données
size(Text)	calcul du nombre d'octets du champ Text
Stop_timer(TL)	désarmement du réveil TL
Time_out(TL)	déclenchement du réveil TL

Précisions concernant la variable NSEQ :
Pour des raisons historiques, la description de l'automate fait apparaître une variable Nseq qui n'est jamais affectée. Cette variable Nseq fait référence au champ NSEQ de la trame reçue et qui est en cours d'analyse. Il est possible d'utiliser une « vraie » variable, mise à jour (implicitement) par la fonction check_frame, ou une fonction « contextuelle » de la forme « extract_Nseq(Frame) ».

3.3.3. Couche Liaison en mode Echo

La station Esclave peut fonctionner en mode Echo ; dans ce cas, la couche Liaison en mode ECHO remplace à la fois la couche Liaison en mode normal et la couche Session. Cette couche se contente de ré-émettre toute trame reçue.
La machine d'état n'existe que sur une station Esclave (le système appelé).

Transitions d'état de Liaison en mode ECHO

Etat initial	Cond. de déclenchement	Ensemble d'actions	Etat final
Stopped	Phy_DATA.ind(Frame, ErrpliP)	Fr = Frame Phy_DATA.req(Fr)	M.RPEE
M.RPEE	Phy_EndDATA.ind()	$none ()	M.RecFE
M.RPEE	Phy_ABORT.ind(_)	$none ()	Stopped
M.RecFE	Phy_DATA.ind(Frame, ErrpliP)	Fr = Frame Phy_DATA.req(Fr)	M.RPEE
M.RecFE	Phy_ABORT.ind(_)	$none ()	Stopped

Signification des états mentionnés dans le tableau précédent

Etat	Signification
Stopped	état de démarrage
M.RPEE (Must Receive Phy_Enddata.ind and Echo)	état de l'émetteur : attente de l'événement généré par le protocole physique signalant la fin de l'envoi de la dernière trame
M.RecFE (Must RECeive Frame and Echo)	état du Récepteur : attente d'une trame en provenance de l'Emetteur

3.3.3.1. Répertoire et traitement des erreurs

Les erreurs sont répertoriées par le codage **EL**-**RNF** ou **EL**-**ENF** portant les significations suivantes :

EL erreur de la couche ***Liaison,***
-**R** erreur en réception,
-**E** erreur en émission,
N numéro de l'erreur,
F erreur fatale.

Tableau récapitulatif des erreurs

EL-R1	Trame incorrecte. Cette erreur peut avoir les origines suivantes : ■ erreur détectée au niveau physique, ■ erreur de CRC, ■ type de trame différent de "0000 ", "0110", "1011", ■ nombre d'octets non compatible avec le champ Size.
	Cette erreur conduit à renvoyer la trame de données en cours, ou un NACK (acquittement négatif)
EL-R2	Réception d'une trame de données avec un numéro de séquence égal au numéro attendu moins 1 (trame de données déjà acquittée positivement)
	Cette erreur conduit à décrémenter le numéro attendu de 1, puis à envoyer un acquittement positif
EL-E1	Expiration du délai TL sans qu'aucune trame de contrôle n'ai été reçue
	Cette erreur conduit à renvoyer la trame de données en cours, l'expiration du délai TL étant interprétée comme un acquittement négatif
EL-E2	Réception d'une trame de données à la place d'une trame de contrôle, avec un numéro de séquence égal au numéro attendu plus 1
	Cette erreur conduit à incrémenter le numéro attendu de 1 ; la trame reçue est interprétée comme un acquittement positif
EL-E3	Réception d'une trame de données à la place d'une trame de contrôle, avec un numéro de séquence égal au numéro attendu moins 1
	Cette erreur conduit à renvoyer la trame de données en cours, la trame reçue étant interprétée comme un acquittement négatif
EL-R3F	Réception d'une trame de données avec un numéro de séquence différent du numéro attendu ou du numéro attendu moins 1 ; ou réception d'une trame de contrôle sans envoi d'une trame de données au préalable
	Cette erreur conduit à réinitialiser la couche ***Liaison*** après avoir informé la couche ***Session*** et fait avorter la couche ***Physique***
EL-E4F	Une quantité égale à (MaxErrl moins 1) envois de la même trame effectuée sans qu'aucune trame d'acquittement positif n'ait été reçue
	Cette erreur conduit à réinitialiser la couche ***Liaison*** après avoir informé la couche ***Session*** et fait avorter la couche ***Physique***
EL-E5F	Réception d'une trame de contrôle avec un mauvais numéro de séquence ; ou réception à la place d'une trame de contrôle d'une trame de données, avec un numéro de séquence différent du numéro attendu moins 1 ou du numéro attendu plus un
	Cette erreur conduit à réinitialiser la couche ***Liaison*** après avoir informé la couche ***Session*** et fait avorter la couche ***Physique***

Les erreurs non fatales sont signalées dans la variable ERRPLI. Toute occurrence de l'une des erreurs fatales est remontée localement grâce à la primitive de service DL_ABORT.ind. La liste complète des numéros d'erreur fatale est fournie au chapitre 3.5.

3.4. Couche Session

3.4.1. Généralités sur la Couche Session

La couche ***Session*** est responsable du dialogue entre les stations. Elle permet l'échange de données entre une station Maître et une station Esclave. La station Maître est toujours l'Appelant, la station Esclave est toujours l'Appelé.
La couche Session offre deux types de services :

- les services d'administration de la connexion : établissement, maintien et coupure entre les deux stations,
- les services d'échanges de données en lecture ou en écriture: composition du dialogue entre stations par échanges d'invitation à émettre, invitations à recevoir, segmentation des données à échanger.

Le protocole de la couche ***Session*** présente des spécificités selon que l'équipement fonctionne en Maître ou en Esclave.

Précisions sur la variable « Code »
Dans le présent chapitre, la variable « Code » est utilisée pour différents services ou primitives décrits au sous-chapitre 3.4.2 et pour des commandes SPDU décrites au sous-chapitre 3.4.3.

Le contenu de cette variable est le suivant :

- le 1er octet du double-octet de la variable « Code » contient la valeur (codée en hexadécimal) du Code de télé-relevé du groupe de données à transférer,
- le 2ème octet du double-octet de la variable « Code » est variable en fonction du type de compteur et du groupe de données désiré. De manière générale, il peut contenir soit une référence identifiant l'application du compteur, soit un code complémentaire du code de télé-relevé précisant l'identité des données désirées.

Les valeurs exactes de la variable « Code » sont précisées au chapitre 2.2.

Précisions sur les variables « SlaveId » et « MasterId »
Dans le présent chapitre, les variables « SlaveId » et « MasterId » sont utilisées pour différents services ou primitives décrits au sous-chapitre 3.4.2 et pour des commandes SPDU décrites au sous-chapitre 3.4.3.

Le contenu de ces variables est le suivant :

- la variable « SlaveId » contient la valeur de la « clé client » de l'appareil concerné,
- la variable « MasterId » contient la valeur de la « clé maître » de l'appareil concerné.

Les valeurs exactes des variables « SlaveId » et « MasterId » sont précisées au chapitre 2.1.1.

3.4.2. Services et primitives de service de Session

L'utilisateur du protocole ***Session*** dispose des services et primitives de service suivants :

Service	Primitive de service
S_Connect	S_Connect.req(MasterId, SlaveId) S_Connect.cnf()
S_Read	S_Read.req(Code, Lg, SSDU) S_Read.cnf(SSDU) S_Read.ind(Code) S_Read.rsp(SSDU) S_EndRead.ind()
S_Write	S_Write.req(Code,SSDU) S_ Write.cnf() S_Write.ind(Code) S_Write.rsp(Lg, SSDU) S_EndWrite.ind(SSDU)
S_Disconnect	S_Disconnect.req() S_Disconnect.cnf()
S_ABORT	S_ABORT.ind ()

Les contenus des variables « Code », « SlaveId » et « MasterId » sont décrits en début du chapitre 3.4.
Le rôle attribué à chaque primitive est décrit dans les 2 tableaux ci-après selon qu'elles existent du côté Maître ou du côté Esclave, les requêtes s'adressant dans tous les cas à la couche Session.

Tableau des rôles de primitive existant du côté Maître

Primitive	Rôle de la primitive
S_Connect.req(MasterId, SlaveId)	Permet de demander l'ouverture d'une connexion entre un Maître d'identité MasterId et un esclave d'identité SlaveID
S_Connect.cnf()	Permet de confirmer l'ouverture d'une connexion préalablement demandée
S_Read.req(Code, Lg, SSDU)	Permet de demander une lecture de données caractérisées par un code et une longueur (SSDU permet de récupérer les données lues)
S_Read.cnf(SSDU)	Permet de transmettre les données qui ont fait l'objet d'une demande préalable de lecture
S_Write.req(Code, SSDU)	Permet de demander une écriture des données, caractérisée par un code
S_Write.cnf()	Permet de signaler la bonne fin d'une écriture préalablement demandée
S_Disconnect.req()	Permet de demander la fermeture d'une connexion
S_Disconnect.cnf()	Permet de confirmer la fermeture d'une connexion préalablement demandée
S_ABORT.ind()	Permet à la couche ***Session*** d'informer d'une fermeture brutale de la connexion à la suite d'une erreur

Les contenus des variables « Code », « SlaveId » et « MasterId » sont décrits en début du chapitre 3.4.

Tableau des rôles de primitive existant du côté Esclave :

Primitive	Rôle de la primitive
S_Read.ind(Code)	Permet de prévenir de l'arrivée d'une demande de lecture caractérisée par un code
S_Read.rsp(SSDU)	Permet de transmettre les données qui ont fait l'objet d'une demande préalable de lecture
S_EndRead.ind()	Permet de prévenir de la fin de la lecture en cours
S_Write.ind(Code)	Permet de prévenir de l'arrivée d'une demande d'écriture caractérisée par un code
S_Write.rsp (Lg, SSDU)	Permet de transmettre la longueur des données qui ont fait l'objet d'une demande préalable d'écriture, et un buffer SSDU pour recevoir ces données
S_EndWrite.ind(SSDU)	Permet de transmettre les données qui viennent d'être écrites, à la fin de l'écriture en cours
S_ABORT.ind()	permet à la couche ***Session*** d'informer d'une fermeture brutale de la connexion à la suite d'une erreur

Les contenus des variables « Code », « SlaveId » et « MasterId » sont décrits en début du chapitre 3.4.

3.4.3. Description des SPDU

Les messages échangés entre entités de session sont acheminés dans des SPDU qui contiennent chacune un nombre entier d'octets inférieur ou égal à 122. Les SPDU possibles et leurs enchaînements sont décrits ci-après.

SPDU de type **XID** :

- SPDUType (XID) : 0F Hexadécimal,
- MasterId : 0 si la session est en lecture seule ou Identité du Maître sinon,
- SlaveId : Identité de l'Esclave.

1 octet	2 octets	2 octets
XID	MasterId	SlaveId

Ce SPDU est envoyé à l'ouverture de connexion par le Maître ; l'Esclave répond par le même SPDU.
Les contenus des variables « SlaveId » et « MasterId » sont décrits en début du chapitre 3.4.

SPDU de type **EOS** :

- SPDUType (EOS) : 01 Hexadécimal.

1 octet
EOS

Ce SPDU est envoyé à la fermeture de connexion par le Maître ; l'Esclave ne répond pas.

SPDU de type **ENQ** :

- SPDUType (ENQ) : 09 H,
- Code : 2 octets.

1 octet	2 octets
ENQ	Code

Ce SPDU est envoyé par le Maître pour initier une lecture caractérisée par le champ Code. Il sera suivi de l'envoi par l'Esclave des données demandées dans un ou plusieurs SPDU de type DAT. La connexion doit être ouverte. Le contenu de la variable « Code » est décrit en début du chapitre 3.4.

SPDU de type **REC** :

- SPDUType (REC) : 06 H,
- Code : 2 octets.

1 octet	2 octets
REC	Code

Ce SPDU est envoyé par le Maître pour initier une écriture caractérisée par le champ Code. Il sera suivi de l'envoi par le Maître des données à écrire dans un ou plusieurs SPDU de type DAT. La connexion doit être ouverte.
Le contenu de la variable « Code » est décrit en début du chapitre 3.4.

SPDU de type **DAT** :

- SPDUType (DAT) : 0C H,
- Data : Données, 1 à 121 octets.

1 octet	1 à 121 octets
DAT	Data

Ce SPDU sert à transférer des données après une demande de lecture ou d'écriture ; il est envoyé par le Maître dans le cas d'une écriture, par l'Esclave dans le cas d'une lecture ; le nombre de SPDU de type DAT envoyés successivement dépend de la taille totale des données à écrire ou à lire.

SPDU de type **EOD** :

- SPDUType (EOD) : 03 Hexadécimal.

1 octet
EOD

Ce SPDU est envoyé pour indiquer que toutes les données ont été transmises, à la fin d'une lecture ou d'une écriture ; il est envoyé par le Maître dans le cas d'une écriture, par l'Esclave dans le cas d'une lecture ; il suit donc toujours un SPDU de type DAT.

SPDU de type **WTM** :

- SPDUType (WTM) : 0A H.

1 octet
WTM

Ce SPDU est envoyé par le Maître pour garder la connexion ouverte, alors qu'il n'a pas de commande à transmettre ; l'Esclave répond par un SPDU de même type.

3.4.4. Paramètres de la couche Session

La couche Session gère trois délais :

- TSE correspondant à la durée maximale d'attente d'une SPDU de l'autre station, et imposé par le type d'échange en cours ; il est calculé sur la base de 6(TM+ TL)+TM max, avec une valeur de TM correspondant à une trame de 6 octets ; TSE est fixé à 22 s ;
- TPA est le délai maximal d'attente d'une commande application du coté Maître ; son écoulement provoque le passage en mode attente si la session n'était pas déjà dans ce mode et l'envoi d'un SPDU de type WTM ; il est calculé sur la base de TSE/3 ; TPA est fixé à 6 s ;
- TMA correspond à la duré maximale en mode attente ; il est fixé à 300 s.

La taille maximale des données véhiculées dans un SPDU de type DAT est égale à MaxPktSize ; elle est fixée à 121.

3.4.5. Transitions d'état

La machine d'état de l'Appelant diffère de celle de l'Appelé. Les deux systèmes jouent tour à tour le rôle d'Emetteur et de Récepteur de SSDU. Le Maître est toujours appelant, l'esclave toujours appelé. Les temps sont exprimés en millisecondes.

Transitions d'état de Session - Maître

Etat initial	Cond. de déclenchement	Ensemble d'actions	Etat final
Init	$true()	MaxPktSize = 121 TPA = 6000 TSE = 22000 TMA = 300000	Stopped
Stopped	S_Connect.req(MasterId, SlaveId)	EndTMA = FALSE SlId = SlaveId MstId = MasterId Comm = XID store_error(_, Comm) DL_DATA.req(concat(XID, MstId, SlId))	M.RecDE
M.RecDE	DL_EndDATA.ind() & (Comm= XID I Comm = ENQ I Comm = WTM)	init_timer(TSE)	M.RData
M.RecDE	DL_EndDATA.ind() & Comm = EOD	S_Write.cnf () init_timer(TPA)	W.Comm
M.RecDE	DL_EndDATA.ind() & Comm = EOS &EndTMA	DL_ABORT.req() S_ABORT.ind()	Stopped
M.RecDE	DL_EndDATA.ind() & Comm = EOS & not(EndTMA)	stop_timer(TMA) DL_ABORT.req() S_Disconnect.cnf()	Stopped
M.RecDE	DL_EndDATA.ind() & Comm = DAT &LgSend >0	store_error(_,Comm) substr_pack(SMsg, MaxPktSize, Packet) LgSend = LgSend -size(Packet) DL_DATA.req(concat(DAT, Packet)	M.RecDE
M.RecDE	DL_EndDATA.ind() & Comm = DAT&LgSend = 0	Comm = EOD store_error(_,Comm)	M.RecDE
M.RecDE	time_out(TMA)	EndTMA =TRUE	M.RecDE
M.RecDE	DL_ABORT.ind(ErrNb)	stop_timer(TMA) store_error(ErrNb, _) S_ABORT.ind()	Stopped
M.RData	DL_DATA.ind(SPDU) & check_type(extract_type(SPDU), Comm)	stop_timer(TSE) store_error(_,Type)	*T.Type*
M.RData	DL_DATA.ind(SPDU) & not(check_type(extract_type(SPDU), Comm))	store_error(ES.R2F,_) stop_timer(TMA) DL_ABORT.req () S_ABORT.ind()	Stopped
M.RData	time_out(TSE)	store_error(ES.R3F,_) stop_timer(TMA) DL_ABORT.req () S_ABORT.ind()	Stopped
M.RData	time_out(TMA)	EndTMA =TRUE	M.RData
M.RData	DL_ABORT.ind(ErrNb)	store_error(ErrNb, _) stop_timer(TSE) stop_timer(TMA) S_ABORT.ind()	Stopped

T.Type	Type = XID & check_id(SPDU)	S_Connect.cnf() init_timer(TPA)	W.Comm
T.Type	Type = XID & not(check_id(SPDU))	store_error(ES.R1F, _) stop_timer(TMA) DL_ABORT.req() S_ABORT.ind()	Stopped
T.Type	Type = DAT	Packet = extract_pkt(SPDU) LgPacket = size(Packet) Comm = DAT store_error(_,Comm)	*T.size*
T.Type	Type = EOD	Comm = EOD store_error(_,Comm)	*T.size*
T.Type	Type = WTM &not(EndTMA)	init_timer(TPA)	W.Comm
T.Type	Type = WTM &EndTMA	Comm = EOS store_error(_,Comm) DL_DATA.req(EOS)	M.RecDE
T.Size	Type = DAT & LgPacket < = LgReceive	Rmsg = concat(Rmsg, Packet) LgReceive = LgReceive - LgPacket	M.RData
T.Size	(Type = DAT & (LgPacket > LgReceive) I (Type = EOD & LgReceive <> 0)	store_error(ES.R5F,_) stop_timer(TMA) DL_ABORT.req() S_ABORT.ind()	Stopped
T.Size	Type = EOD & LgReceive = 0	S_Read.conf(SSDU) init_timer(TPA)	W.Comm
W.Comm	time_out(TPA) & Type <> WTM	Comm =WTM store_error(_, Comm) DL_DATA.req(WTM) init_timer(TMA)	M.RecDE
W.Comm	time_out(TPA) & Type = WTM	DL_DATA.req(WTM)	M.RecDE
W.Comm	S_Read.req(Code, Lg, SSDU)	stop_timer(TPA) EndTMA = FALSE stop_timer(TMA) Rmsg = SSDU LgReceive = Lg Comm = ENQ store_error(_,Comm) DL_DATA.req(concat(ENQ, Code))	M.RecDE
W.Comm	S_Write.req(Code,SSDU) & MstId <> 0	stop_timer(TPA) EndTMA = FALSE stop_timer(TMA) Smsg =SSDU LgSend =size(SMsg) store_error(_,REC) DL_DATA.req(concat(REC, Code)) Comm = DAT	M.RecDE
W.Comm	S_Write.req(Code,SSDU) & MstId = 0	S_Write.cnf(Err_access)	W.Comm

W.Comm	S_Disconnect.req()	stop_timer(TPA) EndTMA = FALSE stop_timer(TMA) Comm = EOS store_error(_,Comm) DL_DATA.req(EOS)	M.RecDE
W.Comm	time_out(TMA)	EndTMA =TRUE	W.Comm
W.Comm	DL_Abort.ind (Errnb)	store_error(ErrNb,_) stop_timer(TPA) stop_timer(TMA) S_ABORT.ind()	Stopped

Les contenus des variables « Code », « SlaveId » et « MasterId » sont décrits en début du chapitre 3.4.

Transitions d'état de Session - Esclave

Etat initial	Cond. de déclenchement	Ensemble d'actions	Etat final
Init	$true()	TSE = 220 MxPktSize = 121	Stopped
Stopped	DL_DATA.ind(SPDU) & check_type(extract_type(SPDU), Comm)	Comm = EOS store_error(_,Type)	*T.Type*
M.RecDE	DL_EndDATA.ind() & (Comm= XID I Comm = WTM)	init_timer(TSE)	M.RData
M.RecDE	DL_EndDATA.ind() & Comm = DAT & LgSend >0	substr_pack(SMsg, MaxPktSize, Packet) LgSend = LgSend -size(Packet) DL_DATA.req(concat(DAT, Packet))	M.RecDE
M.RecDE	DL_EndDATA.ind() & Comm = DAT & LgSend = 0	Comm = EOD store_error(_, Comm) DL_DATA.req(EOD)	M.RecDE
M.RecDE	DL_EndDATA.ind() & Comm = EOD	S_EndRead.ind() Init_timer(TSE)	M.RData
M.RecDE	DL_ABORT.ind(ErrNb)	store_error(ErrNb, _) S_ABORT.ind()	Stopped
M.RData	DL_DATA.ind(SPDU) & check_type(extract_type(SPDU), Comm)	stop_timer(TSE) Type = extract_type(SPDU)	*T.Type*
M.RData	DL_DATA.ind(SPDU) & not(check_type(extract_type (SPDU), Comm))	stop_timer(TSE) store_error(ES.R2F,_) S_ABORT.ind() DL_ABORT.req()	Stopped
M.RData	time_out(TSE)	store_error(ES.R3F,_) S_ABORT.ind() DL_ABORT.req()	Stopped
M.RData	DL_ABORT.ind(ErrNb)	stop_timer(TSE) store_error(ErrNb, _) S_ABORT.ind()	Stopped
T.Type	Type = XID	MstID = extract_Mst(SPDU) SlID = extract_Sl(SPDU) Comm = XID store_error(_, Comm)	*T.id*
T.Type	Type = ENQ	Comm = ENQ store_error(_, Comm) Code = extract_code(SPDU) S_Read.ind(Code)	W.Rrsp
T.Type	Type = REC	Comm = REC store_error(_, Comm) Code = extract_code(SPDU) S_Write.ind(Code)	W.Wrsp

T.Type	Type = DAT	Comm = DAT store_error(_, Comm) Packet = extract_pkt(SPDU) LgPacket = size(Packet)	*T.size*
T.Type	Type = EOD	Comm = EOD store_error(_, Comm)	*T.size*
T.Type	Type = WTM	Comm =WTM store_error(_, Comm) DL_DATA.req(WTM)	M.RecDE
T.Type	Type = EOS	store_error(_, EOS) S_ABORT.ind() DL_ABORT.req()	Stopped
T.id	check_id(SPDU)	store_error(_, XID) DL_DATA.req(concat(XID, MstId, SlId))	M.RecDE
T.id	not(check_id(SPDU))	store_error(ES.R1F, XID) S_ABORT.ind() DL_ABORT.req()	Stopped
T.Size	LgPacket < = LgReceive & Type = DAT	Rmsg = concat(Rmsg, Packet) LgReceive = LgReceive - LgPacket init_timer(TSE)	M.RData
T.Size	(Type = DAT & LgPacket > LgReceive) I (LgReceive <> 0 & Type = EOD)	store_error(ES.R5F,_) S_ABORT.ind() DL_ABORT.req ()	Stopped
T.Size	LgReceive = 0 & Type = EOD	S_EndWrite.ind(Rmsg) init_timer(TSE)	M.RData
W.Rrsp	S_Read.rsp(SSDU)	Comm = DAT store_error(_, Comm) Smsg =SSDU LgSend =size(SMsg) substr_pack(SMsg, MaxPktSize, Packet) Lgsend = Lgsend - size(Packet) DL_DATA.req(concat(DAT, Packet))	M.RecDE
W.Rrsp	DL_ABORT.ind(Errnb)	store_error(ErrNb,_) S_ABORT.ind()	Stopped
W.Wrsp	S_Write.rsp(Lg, SSDU) & MstId <> 0	LgReceive = Lg Rmsg = SSDU init_timer(TSE)	M.RData
W.Wrsp	S_Write.rsp(Lg, SSDU) & MstId = 0	store_error(ES.R4F,_) S_ABORT.ind() DL_ABORT.req()	Stopped
W.Wrsp	DL_ABORT.ind(Errnb)	store_error(ErrNb,_) S_ABORT.ind()	Stopped

Les contenus des variables « Code », « SlaveId » et « MasterId » sont décrits en début du chapitre 3.4.

Signification des états mentionnés dans les tableaux précédents

Etat	Signification
Init	état d'initialisation
Stopped	état de démarrage
M.RecDE (Must RECeive Dl_Enddata.ind)	état d'attente de l'événement généré par le protocole liaison signalant l'envoi réussi des dernières données (données acquittées positivement)
M.RData (MustReceive dl_DATA.ind)	état d'attente de l'événement généré par le protocole liaison signalant la réception de données
W.Rrsp (Wait s_Read.RSP)	état de l'Esclave : attente du retour de l'application donnant les données à envoyer, lors d'une demande de lecture
W.Wrsp (Wait s_Write.RSP)	état de l'Esclave: attente du retour de l'application donnant la longueur des données qui seront reçues, lors d'une demande d'écriture
W.Comm (Wait Command)	état du Maître : attente d'une commande en provenance de l'application sous contrôle du délai TPA
T.Id (Test Identity)	état de l'Esclave : vérification des identités reçues dans un SPDU de type XID
T.Type (Test Type)	vérification du type du dernier SPDU reçu
T.Size (Test Size)	état de vérification de la taille des données reçues dans un SPDU de type DAT

Définition des procédures et des fonctions classées dans l'ordre alphabétique

Procédure ou fonction	Définition
check_id (SPDU)	vérification que les identifiants Maître et Esclave reçu dans un SPDU de type XID sont corrects
check_type(Type, Comm)	vérification que le type du SPDU reçu est correct, ainsi que son enchaînement : ■ pour le Maître, à vrai si : • Type = XID et Comm = XID, • Type = DAT et Comm = ENQ ou DAT, • Type = EOD et Comm = DAT, • Type = WTM et Comm = WTM, ■ pour l'Esclave, à vrai si : • Type = XID et Comm = EOS, • Type = ENQ ou REC et Comm = XID ou EOD ou WTM, • Type = DAT et Comm = REC ou DAT, • Type = EOD et Comm = DAT, • Type = WTM et Comm = WTM ou XID ou EOD, • Type = EOS et Comm = EOD ou WTM.
concat(Type, Packet) ou concat(Type, Code) concat(Type,MstId, SlId)	construction d'une chaîne binaire par concaténation de plusieurs paramètres
extract_code(SPDU)	extraction du champ code d'une SPDU de type REC ou ENC (Esclave)
extract_Mst(SPDU)	extraction du champ identité Maître d'une SPDU de type XID (Esclave)
extract_pkt(SPDU)	extraction du champ de données d'une SPDU de type DAT
extract_Sl(SPDU)	extraction du champ identité Esclave d'une SPDU de type XID (Esclave)
extract_type(SPDU)	extraction du type de la SPDU reçue
init_timer(TSE), init_timer(TPA) ou init_timer(TMA)	armement du réveil TSE, TPA ou TMA
size(SMsg)	calcul de la taille en octets du message SMsg
stop_timer(TSE), stop _timer(TPA) ou stop _timer(TMA)	désarmement du réveil TSE, TPA ou TMA
store_error(Errnb, Comm)	recopie de la dernière commande Comm dans les 4 bits de poids fort de ERRSES et du numéro de la dernière erreur fatale dans ERRFAT ou ERRSES
substr_pack(SMsg, MaxPktSize, packet)	extraction de MaxPktSize octets du message SMsg et recopie dans Packet ; si Smsg contient moins d'octets que MaxPktSize, recopie de Smsg dans Packet
time_out(TSE), time_out(TPA) ou time_out(TMA)	déclenchement du réveil TSE, TPA ou TMA

3.4.6. Répertoire et traitement des erreurs de la couche Session

Les erreurs sont répertoriées par le codage **ES-RNF** ou **ES-ENF** portant les significations suivantes :

ES erreur de la couche ***Session,***
-R erreur en réception,
-E erreur en émission,
N numéro de l'erreur,
F erreur fatale.

Tableau récapitulatif des erreurs

ES-R1F	Réception d'un mauvais identificateur Maître ou Esclave
	Cette erreur conduit à réinitialiser la couche ***Session,*** faire avorter la couche ***Liaison***, et à en informer l'application si l'équipement est Maître
ES-R2F	Réception d'une SPDU non répertoriée
	Cette erreur conduit à réinitialiser la couche ***Session,*** faire avorter la couche ***Liaison***, et à en informer l'application si l'équipement est Maître
ES-R3F	Ecoulement du délai TSE sans réception d'une SPDU
	Cette erreur conduit à réinitialiser la couche ***Session,*** faire avorter la couche ***Liaison***, et à en informer l'application si l'équipement est Maître
ES-R4F	Réception d'une demande d'écriture dans une session lecture seule (Esclave)
	Cette erreur conduit à réinitialiser la couche ***Session*** et à avorter la couche ***Liaison***
ES-R5F	Réception d'un nombre de données non prévu
	Cette erreur conduit à réinitialiser la couche ***Session*** après en avoir informé l'application et fait avorter la couche ***Liaison***

Toute occurrence de l'une des erreurs fatales est signalée dans ERRFAT et ERRSES (cf. chapitre 3.5)

3.5. Liste des erreurs

Les erreurs sont consignées dans les variables ERRPLI (pour les couches Physique et Liaison), ERRFAT (pour les erreurs fatales toutes couches confondues) et ERRSES (erreurs de la couche Session).

Codage de ERRPLI

Numéro du bit	0	1	2	3	4	5	6	7
Erreur	**EP-R1**			**EL-R1**	**EL-R2**	**EL-E1**	**EL-E2**	**EL-E3**

Codage de ERRFAT

Numéro du bit	0	1	2	3	4	5	6	7
Erreur	**EP-R4F**	**EP-R5F**	**EL-R3F**	**EL-E4F**	**EL-E5F**	**ES-R1F**	**ES-R2F**	**ES-R3F**

Codage de ERRSES

Numéro du bit	0	1	2	3	4	5	6	7
Erreur	**ES-R4F**	**ES-R5F**	**EP-R6F**	**EP-R7F**	**dernière commande correcte**			

Les erreurs fatales sont mémorisées par la couche Session dans les variables ERRFAT et ERSES (Errses mémorise également la dernière commande correcte avant l'erreur fatale).
La couche Liaison mémorise dans la variable ERRPLI les dernières erreurs non fatales détectées par les couches Physique et Liaison ; ERRPLI est remis à 0 à chaque fois qu'une trame est envoyée sans erreur.

Quelle que soit la couche incriminée, l'occurrence d'une erreur fatale conduit à :

- éventuellement, faire s'arrêter la couche inférieure au moyen d'une primitive de service abort.req,
- éventuellement, informer la couche supérieure au moyen d'une primitive abort.ind avec comme paramètre le numéro de l'erreur fatale,
- effectuer une réinitialisation totale de l'occurrence d'automate correspondante.

3.6. Principe du CRC

3.6.1. Généralités sur le CRC

Le contrôle de la transmission correcte des bits se fait globalement sur un bloc d'information exprimé en octets (cf. le chapitre 3.3 décrivant la couche Liaison). La clef de contrôle est un ensemble de bits appelé **champ de contrôle**. Son calcul repose sur la théorie des codes cycliques qui fait appel à la **division des polynômes** et aux propriétés algébriques des **restes de la division** des polynômes.

3.6.2. Opérations sur les polynômes

Soit A = (a_1, a_2, ..., a_n) la suite de bits sur laquelle doit être calculée la clef de contrôle. Cette suite peut être vue comme un polynôme de degré n-1 :

$$A(X) = a_1X^{n-1} + ... + a_{n-1}X + a_n$$

Supposons choisi un polynôme diviseur D(X) de degré m. La division polynomiale de $A(X)*X^m$ par D(X) donne la relation suivante :

$$A(X)*X^m = D(X)*Q(X) + R(X)$$

où R(X) est un polynôme de degré inférieur ou égal à m-1 représentant la suite de bits R = (r_1, r_2, ..., r_m)

Compte tenu des propriétés de l'arithmétique booléenne, cette relation s'écrit aussi :

$$A(X)*X^m + R(X) = A(X)*X^m - R(X) = D(X)*Q(X)$$

ce qui représente la suite de bits (a_1, a_2, ..., a_n, r_1, r_2, ..., r_m)

3.6.3. Procédure de contrôle

Le champ de contrôle du CRC (Cyclical Redundancy Check) est représenté par la suite de bits R = (r_1, r_2, ..., r_m). Son calcul pratique courant est à base de registres de décalage et de cumul permettant d'évaluer la clef au fur et à mesure que les bits de données se présentent. L'algorithme correspondant n'est pas décrit dans le présent document.
Comme l'indique la théorie, l'Emetteur doit évaluer la suite de bits R = (r_1, r_2, ..., r_m), concaténer cette suite à la séquence des bits à protéger A = (a_1, a_2, ..., a_n) et, transmettre au Récepteur la suite de bits résultante (a_1, a_2, ..., a_n, r_1, r_2, ..., r_m).
Le Récepteur considère qu'il n'y a pas d'erreur de transmission dès lors que la suite de bits reçues correspond à un polynôme de degré m+n-1 divisible exactement par D(X).

3.6.4. Paramètres de fonctionnement

m	16
D(X)	$X^{16} + X^{15} + X^2 + 1$