SOMMAIRE

INTRODUCTION

Le passage de la théorie à la pratique est une opération délicate, dans l'univers informatique comme ailleurs. Monsieur **Bernard Ducourneau**, chef de projet au sein de la société **SOULÉ**, m'a proposé de vérifier cet adage dans le cadre du stage sanctifiant la formation de B.T.S. Informatique de Gestion.

En l'occurrence, la proposition, qui consistait au développement d'un logiciel de paramétrage, m'a vivement attiré et c'est avec un grand intérêt que je me suis plongé, pendant 10 semaines, dans ce travail.

Les lignes qui suivent tentent donc de refléter le plus fidèlement possible la démarche qui a été la mienne lors de la conception de cette application, de la définition des besoins, en passant par l'analyse des problèmes, jusqu'à la mise en application des solutions.

I - PRÉSENTATION

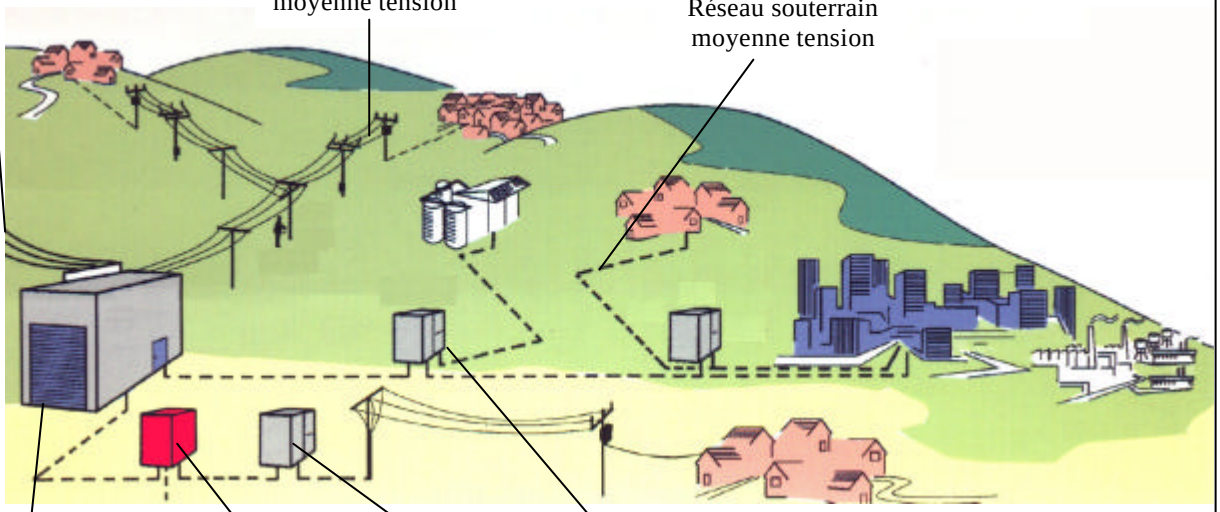
a) L'entreprise SOULÉ

L'entreprise SOULÉ a été créée en 1862 à Bagnères de Bigorre (65). SOULÉ est un groupe familial autonome. Le groupe opère dans les matériels électriques professionnels, de la haute à la base tension, dans trois familles de métiers : les protections foudre, l'appareillage de coupure, l'analyse et la détection de défauts.

Électricité provenant des centrales thermiques classiques (10%), nucléaires (70%) et hydrauliques (20%) via des postes de distribution haute tension

Réseau aérien moyenne tension

Réseau souterrain moyenne tension



Poste de distribution de courant (postes HTA)

Interrupteurs télécommandés et **coffret ITI**

Appareils qui détectent des défauts de courant et qui envoient des informations vers les postes de télé conduite

Réseau électrique classique

SOULÉ dispose de trois sites de production : Bagnères de Bigorre, Boulogne, Frelinghien.
Le groupe SOULÉ, c'est aujourd'hui :

- 500 professionnels
- Un chiffre d'affaires en 1998 de 375 millions de francs
- Un actif net de 94,3 millions de francs
- Un parc de 300 ordinateurs, micro – ordinateurs ou postes de travail informatisés
- 1 laboratoire de choc électrique, 1 laboratoire de puissance, 3 laboratoires électroniques
- 25 millions de francs de dépenses en Recherche et développement par an

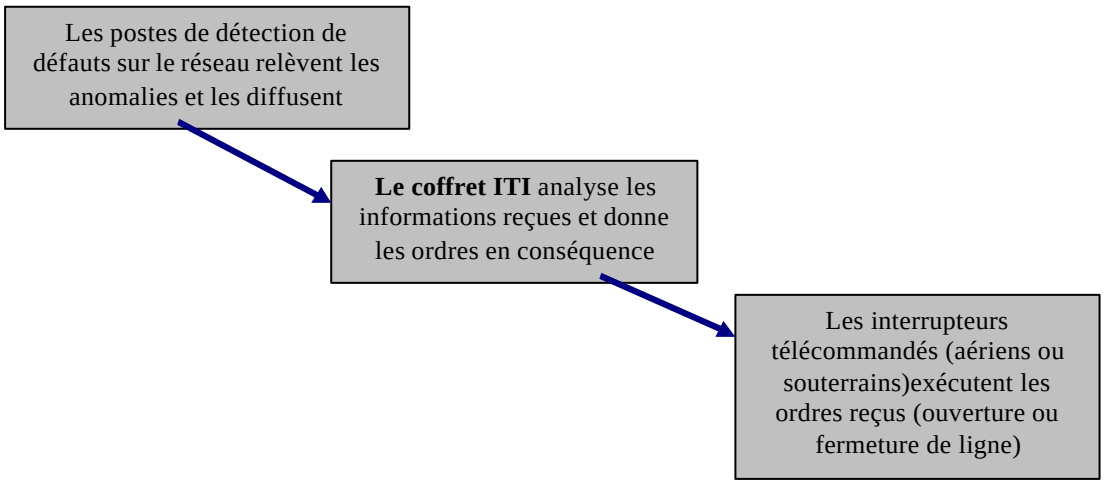
Historiquement, lorsqu'un problème quelconque survenait sur un réseau électrique, EDF (par exemple) envoyait des hommes sur le terrain pour s'informer de la situation. A l'heure actuelle, la nécessité d'une meilleure gestion du réseau et d'une meilleure qualité de services fait que des équipements électriques «intelligents » donnent la possibilité de connaître l'image du réseau en temps réel.
SOULÉ fabrique donc des appareils permettant la détection, le contrôle, la communication et l'exploitation des courants des réseaux souterrains et aériens afin de faciliter la conduite et l'auscultation.

b) Le service Laboratoire électronique

Le service dans lequel j'ai travaillé est le service Laboratoire électronique du département Appareillage – Distribution (5 techniciens, encadrés par un chef de projet, composent le service).
Ce service s'occupe du développement de nouveaux produits de l'entreprise. Plus précisément, développer des systèmes de commande d'interrupteurs et sectionneurs (à coupure dans l'air ou dans le gaz) est le rôle du département.
L'objectif du stage est de développer un logiciel de paramétrage et de test des **coffrets ITI,** créés par le service Laboratoire électronique.

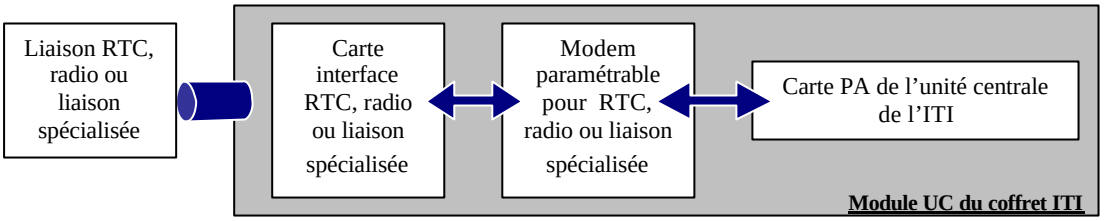
Coffret UN - ITI

Les coffrets ITI permettent la télécommande d'interrupteurs (HTA motorisés). En d'autres termes, ils constituent un élément intelligent du réseau, fournissant des informations pour la prise de décisions.



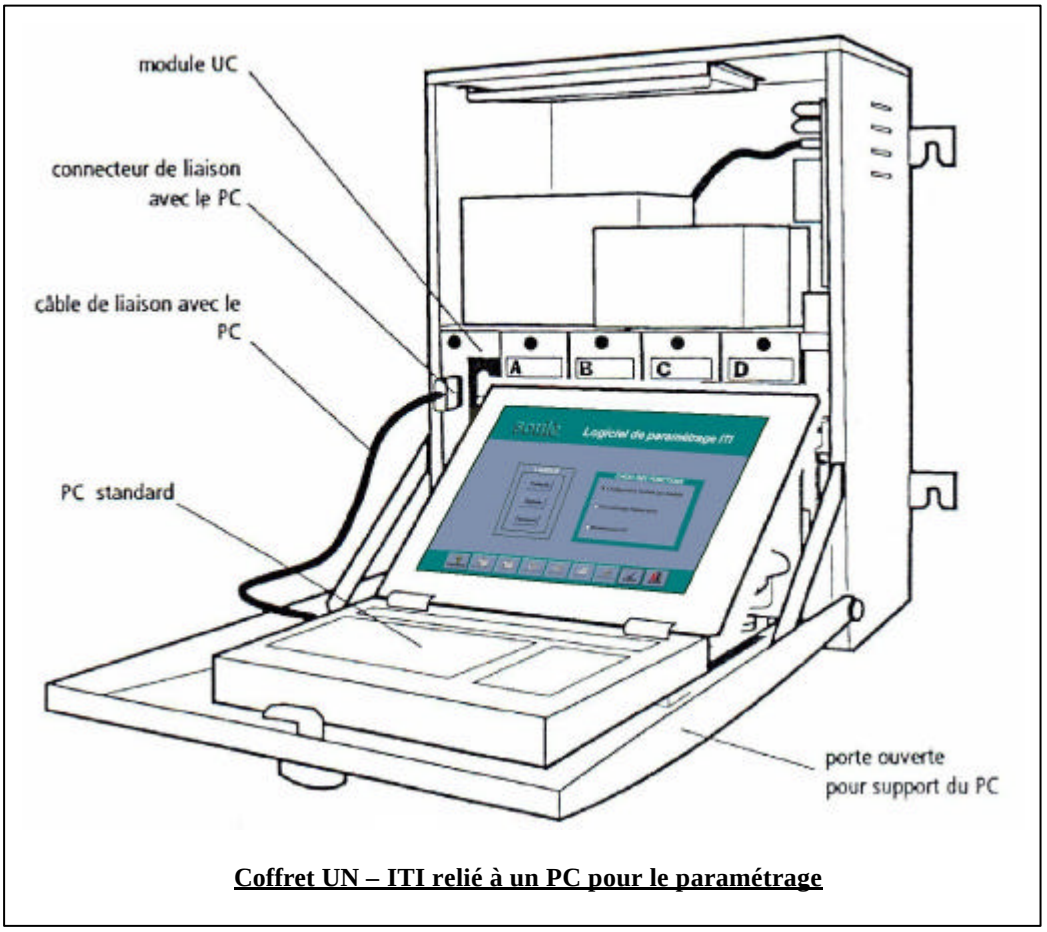
Le coffret ITI peut être utilisé en mode télécommandé (à distance) ou manuellement (grâce à des boutons poussoirs présents sur le coffret).
Le cœur intelligent de ce coffret est une carte électronique appelée carte PA : elle est constituée d'un microprocesseur, de mémoires EPROM, flash, RAM, d'un UART,... . Les logiciels présents sur les EPROM de la carte ont été écrits en langage C. C'est le Laboratoire électronique qui est à l'origine de la création de cette carte.

Remarque : pour envoyer et recevoir des informations depuis le coffret ITI, il existe 3 supports de communications, le réseau téléphonique commuté (RTC), les communications radio ou les lignes spécialisées. C'est le modem inclus dans le coffret qui gère ces communications.



c) Le logiciel à développer

L'agent qui va devoir paramètrer ce coffret a la possibilité d'agir sur celui - ci à distance mais, à cette manière de paramétrage s'ajoute l'alternative de relier le coffret électrique à un PC par un câble (liaison série) pour agir sur les réglages de l'appareil.



Coffret UN – ITI relié à un PC pour le paramétrage

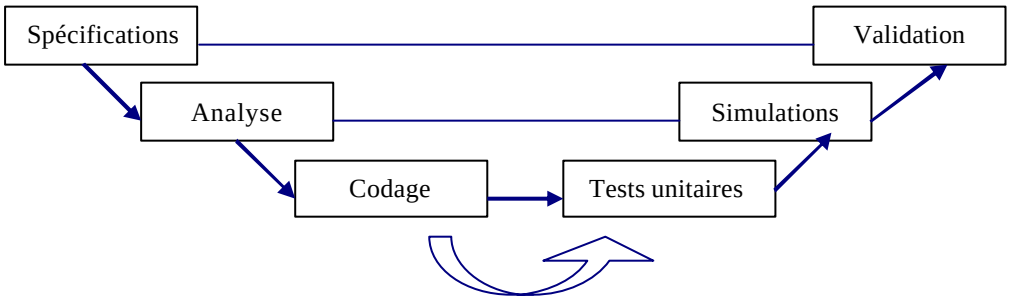
Je devais m'attacher à développer un logiciel qui donne la possibilité de paramétrer le coffret par la liaison série.
Les directives suivantes m'ont été données :

- Rédaction d'un **cahier des charges** complet et de **documents d'analyse**.
- **Réaliser la programmation des écrans** de paramétrage et de test du coffret ITI.
- Portabilité du logiciel en **3 langues** (français, anglais, espagnol) avec utilisation de fichiers pour permettre l'ajout de nouvelles langues (si possible en évitant une recompilation totale).
- Utilisable sur **tout type de PC** (sous un environnement Windows 3.x et 95/98/NT)
- Possibilité d'utiliser le logiciel **non connecté** à l'ITI, doit permettre de créer et sauvegarder des configurations pour un futur paramétrage du coffret.
- Gestion du **port série** pour paramètrer le coffret.
- Prévoir **l'intégration** des modules en cours de développement
- **Former le technicien** qui sera chargé de la maintenance et de l'évolution du logiciel.

Un logiciel de paramétrage existait déjà. Cette version a été créée avec le logiciel de programmation WinDev.
Le développement devait se baser sur la version du logiciel développé sous WinDev.
Les raisons principales du changement d'environnement de développement sont :

- Des bugs dans la version actuelle du logiciel.
- Le départ de techniciens compétents sous WinDev.
- Des problèmes avec la hot-line de WinDev.
- La nécessité d'une compatibilité totale avec Windows.
- Le standard de fait que représente Visual Basic pour les applications Windows.
- La facilité de former rapidement un développeur à Visual Basic.

Le cycle de développement d'un logiciel se déroule ainsi dans le service:



II - ÉTUDE PRÉLIMINAIRE

Avant de se lancer aveuglement dans le développement du logiciel, j'ai au préalable étudié si le développement sous l'EDI Visual Basic 4.0 était adapté à la situation.
J'ai rédigé un **cahier des charges**. Au regard des besoins exprimés dans ce cahier des charges, les points particuliers suivants ont été l'objet d'une attention particulière.

a) *Internationalisation d'une application sous Visual Basic*

Le logiciel devait fonctionner en anglais, espagnol et français. J'ai donc cherché quel outil satisfaisait à ce besoin. La réponse qui s'imposait était les **fichiers de ressources**.
Les fichiers de ressources ont été ajoutés à Visual Basic pour faciliter l'internationalisation des applications.
Avant la disponibilité de ce type de fichiers, il fallait éditer tous les titres dans le projet Visual Basic lui-même.
Les fichiers de ressources permettent d'isoler toutes les chaînes d'une application, ainsi que tous les fichiers (images, icônes, …) qui concernent directement le projet.
Les **ressources de chaîne** regroupent la totalité du texte qui apparaît dans l'interface utilisateur de l'application. Menus, boîtes de dialogues, messages d'erreur, d'avertissement et d'information en constituent une liste non exhaustive.

Bloc de données + bloc de code = Produit

Le bloc de données contient toutes les ressources de chaîne de l'interface utilisateur mais aucun code. À l'inverse, **le bloc de code** ne contient que le code de l'application commun à tous les paramètres régionaux. Il en résulte :

- **Efficacité**. Le développement d'une version dans une autre langue de l'application implique simplement la création d'un nouveau fichier de ressources. En effet, toutes les versions utilisent le même bloc de code. Il est ainsi possible de créer directement des versions en plusieurs langues de l'application Visual Basic.
- **Sécurité accrue**. Que l'on décide de traduire une application en interne ou d'utiliser les services d'une autre société, il n'est pas nécessaire d'accéder au code source pour développer des versions en d'autres langues de l'application. Cette approche diminue également le nombre de tests nécessaires pour valider la version en langue étrangère.
- **Adaptation de meilleure qualité**. Les ressources de chaîne figurant toutes dans un même fichier, la traduction en est facilitée et les risques de laisser des chaînes non traduites sont moindres.

Remarque : un document d'analyse détaillé a été rédigé pour donner une vision précise de l'utilisation des fichiers de ressources sous Visual Basic.

b) Portabilité sous Windows 16 et 32 bits

Un logiciel est dit «portable » sous un système d'exploitation (Microsoft Windows, Linux, MacOS, …) lorsqu'il fonctionne sur ce système.
Le logiciel à développer devait être portable sur Windows 16 (3.x) et 32 (95, 98 et NT) bits.

Remarque : les différences entre Windows 16 bits et Windows 32 bits sont nombreuses. On peut citer notamment la gestion de la mémoire lors de l'exécution d'un programme : sous Windows 32 bits, une plage mémoire délimitée précisément est allouée pour le programme que l'on exécute. Il en résulte une sécurité accrue lors de l'exécution du programme.

Comme Visual Basic 5.0 et 6.0 sont exclusivement 32 bits, ils permettent de créer des programmes uniquement pour Windows 32 bits ; par contre, **Visual Basic 4.0** est utilisable sur Windows 16 et 32 bits donc le choix de cet L4G s'imposait ici.
Par ailleurs, l'utilisation des constantes de compilation conditionnelle (#If Win16 … #Else … #EndIf) donne au code la possibilité d'être compilé au mieux sous un OS 16 et 32 bits, ce qui est intéressant pour la maintenance future de l'application.

c) La communication série

La communication série sous Visual Basic 4.0 est facilité par l'utilisation d'un contrôle ActiveX fourni par Microsoft, **le contrôle Comm**. C'est un composant complet, puissant et d'utilisation facile.
Ce composant fournit les outils nécessaires à l'envoi ou la réception de données via le port série. Plus précisément, le contrôle COMM16.ocx (ou COMM32.ocx en version 32 bits) permet d'utiliser un objet de type COM dont les propriétés et les méthodes servent à gérer la communication avec le port série.
Un projet d'exemple fourni avec Visual Basic donne une base de référence complète pour en déduire que la communication série entre le coffret ITI et le PC sera implémentable sous Visual Basic 4.0.

Remarque : un contrôle **ActiveX** est un élément logiciel réutilisable qui respecte le modèle COM (Component Objet Model), modèle à la base des différents «briques » utilisés dans l'environnement OLE (Object Linking and Emdebbing) de Microsoft. La création de contrôle ActiveX est un domaine très intéressant, il est d'ailleurs possible de créer de nouveaux contrôles à partir de la version 5.0 de Visual Basic. Cependant, l'utilisation de Microsoft Visual C++ et les MFC (Microsoft Fundation Classes) constituent une solution plus professionnelle, mais qui nécessite des connaissances solides concernant la communication des composants OLE avec leurs containers…

d) Distribution d'une application sous Visual Basic

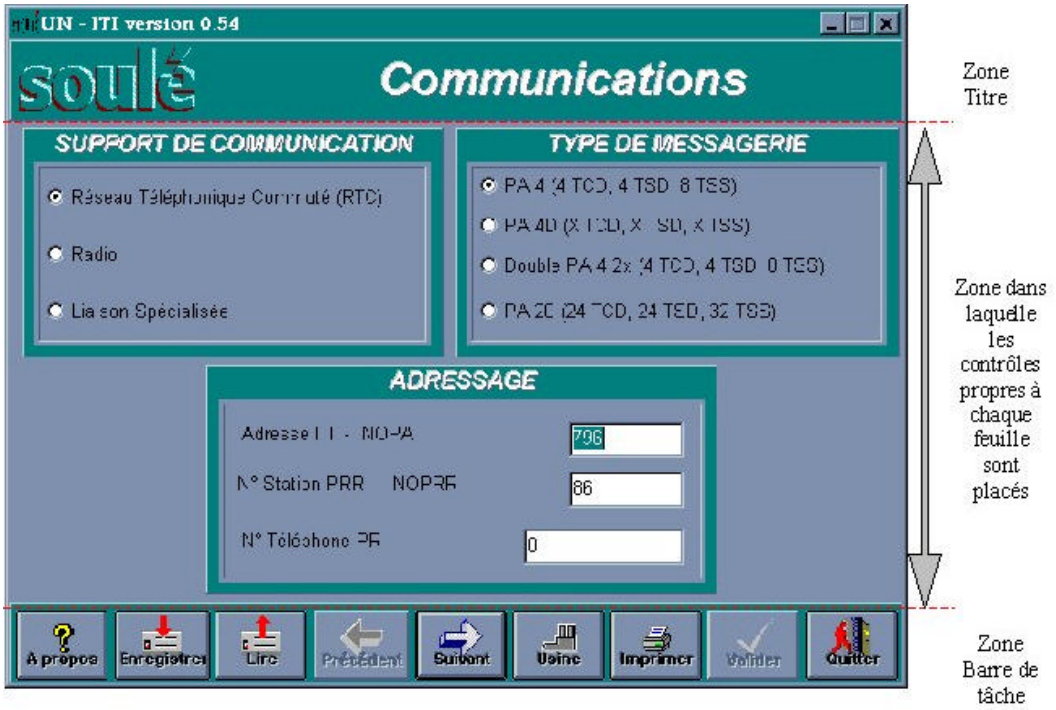
Développer un logiciel, c'est bien. Le distribuer correctement à ses clients, c'est mieux. Dans cette optique, Visual Basic fournit un outil adéquat : l'**assistant d'installation** donne la possibilité de créer un programme d'installation conforme au standard Windows.
Ainsi, l'assistant se charge de rechercher tous les fichiers nécessaires au fonctionnement de l'application (DLL, OCX, fichier de ressources, …), les compresse et les place sur le support de distribution de son choix (disquettes, dossier unique ou dossiers séparés). Il permet également lors de l'installation du logiciel d'inscrire les données dans le registre Windows pour permettre un désinstallation «propre » de l'application. Par ailleurs, la procédure d'installation est facilement modifiable puisque le code source du projet Visual Basic à la base du programme d'installation est fourni par Microsoft. Il est par conséquent possible de modifier selon ses besoins l'enchaînement des écrans du programme d'installation (ajout d'écran, modification du script d'installation pour copier des fichiers particuliers lors de l'installation).
En résumé, la distribution du logiciel est une partie facilement réalisable.

III - ANALYSE

Le **suivi du développement** a été clairement défini. Entres autres, un classeur devait regrouper tous les documents nécessaires à la création du logiciel, un fichier texte rassemblait les informations à connaître pour comprendre le travail réalisé.
La philosophie de développement consistait en la création de composants les plus généraux possibles, en particulier pour les modules. Par exemple, lors de la communication entre le PC et l'ITI, seul le contenu des messages diffère d'une transmission à une autre alors que la forme des messages est toujours la même. L'intérêt est donc évident : lorsque les modules permettant la communication sont opérationnels, le développement des éléments spécifiques (écrans, …) à un échange précis d'informations peut être réalisé rapidement.

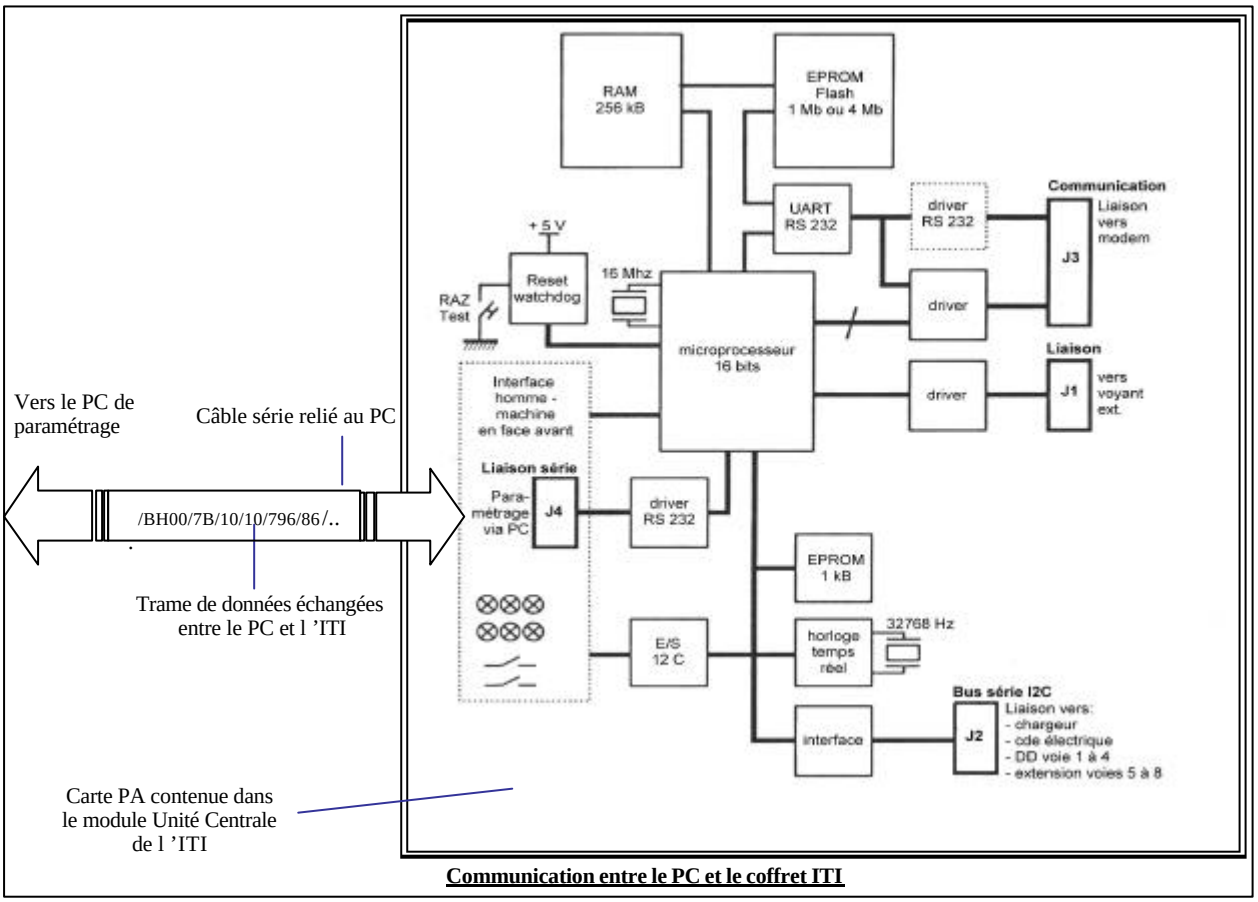
a) Charte graphique

La communication visuelle d'informations entre l'utilisateur et le logiciel constitue **l'interface graphique**. Plus l'interface graphique est conviviale, plus le logiciel est facilement utilisable. Il est donc nécessaire lors de la création d'écrans de se conformer à un modèle précis, que j'ai défini dans une charte graphique.
La charte graphique regroupe l'ensemble des éléments affichés à l'écran, la couleur des fenêtres, le pointeur de la souris, …Par exemple, les écrans de l'application, obtenus après plusieurs maquettes de démonstration, doivent respecter l'ergonomie suivante :



b) Trames de communications série

Le logiciel, via la liaison série du PC, peut envoyer ou demander des informations de configuration à l'ITI. Le dialogue entre le PC et le coffret se déroule en fait comme indiqué dans la figure ci – après.



Communication entre le PC et le coffret ITI

Ces échanges d'informations doivent nécessairement être formatés. A cette fin, on utilise une trame de 255 caractères maximum dont le modèle est le suivant :

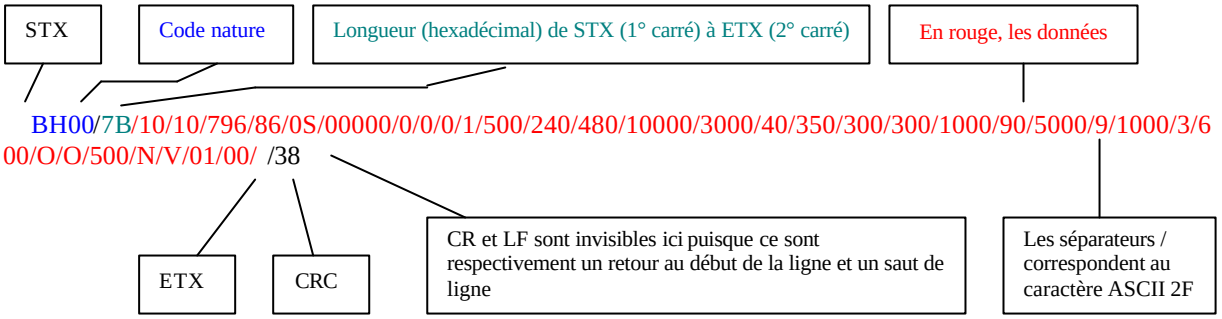
STX	Code	Longueur	Données	ETX	CRC	CR LF

- **STX** : Une trame commence toujours par STX (caractère ASCII 02)
- **Code** : Codé sur 4 octets, il permet de formuler une demande précise
- **Longueur** : Longueur en hexadécimal (sur 2 octets) de STX à ETX
- **Données** : Liste des paramètres du module
- **ETX** : Fin des données (caractère ACSII 03)
- **CRC** : Codé en hexadécimal, le Cycling Redency Code est calculé suivant un algorithme particulier et identifie les données d'une trame
- **CR LF** : Une trame finit toujours par un Cariage Return et un Line Feed (caractères ASCII 13 et 10)

Il existe différents types de demande de configuration (partie intitulée «Code » dans le modèle de trame ci - dessus) :

- Demande de configuration d'un module particulier.
- Demande de configuration matérielle de l'ITI (tous les modules).
- Envoie de paramétrage des modules vers l'ITI.

Voici un exemple de trame pour les paramètres du module Communications :



IV - DÉVELOPPEMENT

a) Généralités

L'écriture et la lecture des données sur le disque dur ne font pas l'objet d'un paragraphe à part entière car le format des données est le même que pour la liaison série (on utilise en effet la même fonction pour prendre en compte les données lues sur un fichier ou dans le tampon de réception série). La manipulation de fichiers sous VB est quant à elle simple, le langage fournissant des outils efficaces. Il est à souligner que les opérations sur les fichiers en mode binaire (Open NomFich For Binary As #1) sont généralement plus rapides que les antiques instructions Input et Output. Par ailleurs, l'OCX COMM16DLG (ou COMM32DLG en version 32 bits) a été utilisé pour implémenter les boîtes de dialogue «Ouvrir » et «Enregistrer sous ».
Pour **l'impression**, l'objet Printer fournit par VB est à la fois souple et facile à utiliser. Il permet d'envoyer du texte formaté vers l'imprimante.
Les exemples de développement présentés dans les paragraphes qui suivent ont été choisi en raison de leurs caractères représentatifs de ce qu'a été l'ensemble du développement.

b) Interface graphique

Lors de la réalisation des écrans, certains points nécessitaient des recherches particulières. La réalisation d'info - bulles, par exemple, a donné lieu à un développement intéressant.

Exemple d'info - bulle :

J'ai volontairement reproduit à l'identique le document d'analyse relatif aux info - bulles car, outre l'aspect programmation pure sous Visual Basic, il montre comment se déroulait le stage d'un point de vue organisationnel (avec l'entête du document par exemple). En l'occurrence, lorsque on implémentera des info - bulles dans l'application, il faudra faire ré férence (via un commentaire) à ce document d'analyse. Ce lien entre document d'analyse et code de l'application permet en outre de mieux cerner le travail réalisé.

Projet UN – ITI		**Visual Basic 4.0**
	Info – bulles sous Visual Basic 4.0	
Répertoire : \itivb\suivi	**Fichier : infobul.doc**	**Créé par : David Rousse, le 16/02/1999**
Validé par : , le _ / _ / _	**Indice : 0**	**Codé dans : *.frm** **Réf : 5060**

Principe : les info - bulles qui s'affichent à côté d'un contrôle représentent une aide permanente pour l'utilisateur. Sous Visual Basic 5.0, il existe une propriété appelée ToolTipText qui permet de spécifier le texte à afficher dans l'info - bulle pour un contrôle donné. Sous Visual Basic 4.0, il faut simuler les info - bulles.

Implémentation:
La simulation d'info - bulles pour un contrôle bouton (appelé Command1 par exemple) passera par les étapes suivantes :

1. Placer un bouton de commande Command1 sur la feuille
2. Mettre dans la propriété Tag du bouton le texte que l'on veut afficher dans l'info - bulle.
3. Ajouter un contrôle Timer (de nom par défaut Timer1) sur la feuille.
4. Mettre sa propriété Interval égale au délai d'apparition de l'info - bulle après que la souris ait été positionnée sur le contrôle. 500 est une «bonne » valeur pour Interval, ainsi l'info - bulle apparaît après 500 ms.
5. Taper le code suivant dans la procédure événementielle Load de la feuille (de nom Form1 par exemple) :
```
Private Sub Form_Load()
    Timer1.Enabled = True
    ToolTips Me, ToolTip, False
End Sub
```
6. Taper le code suivant dans la procédure événementielle Timer du contrôle Timer1 :
```
Private Sub Timer1_Timer()
    ToolTip.Visible = True
    ToolTip.ZOrder 0
    Timer1.Enabled = False
End Sub
```
7. Ajouter le code suivant dans la procédure événementielle MouseMove du form :
```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ToolTips Me, ToolTip, False
End Sub
```
8. Ajouter un contrôle PictureBox sur la feuille. C'est ce contrôle qui va contenir l'info - bulle. Modifier les propriétés suivantes du contrôle :
   AutoSize → True
   BackColor → &H80000005& (couleur crème)
   Name → ToolTip
   Visible → False
9. Taper le code suivant dans la procédure événementielle MouseMove du bouton, cela permettra de faire apparaître l'info - bulle:
```
Private Sub Command1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ToolTips Me, Command1, True
End Sub
```
10. Enfin, il faut à présent définir la fonction ToolTips :
```
    Sub ToolTips(Frm As Form, Ctl As Control, OnOff As Boolean)

      'si l'argument OnOff vaut True, on affiche l'info - bulle
      If OnOff Then
        'on efface le texte contenu dans la PictureBox
        Frm.ToolTip.Cls

        'on met le texte contenu dans la propriété Tag
```

```
        'du contrôle Ctl passé en paramètre dans la PictureBox
        Frm.ToolTip.Print " " & Ctl.Tag & " "

        'on ajuste la largeur de la PictureBox en fonction
        'de la largeur du texte que l'on vient d'y affecter
        Frm.ToolTip.Width = Frm.ToolTip.TextWidth(Ctl.Tag & "  ")

        'on cherche à quel endroit afficher l'info - bulle
        'Rappel : la propriété ScaleHeight et ScaleWidth dans form
        'définissent respectivement la hauteur et la largeur de la zone
        'dans laquelle les contrôles sont visibles

        'on cherche en premier lieu si on affiche l'info - bulle
        'en dessous ou en dessus du contrôle
        If Ctl.Top + Ctl.Height + Frm.ToolTip.Height + 40 < Frm.ScaleHeight Then
           'on affiche l'info - bulle au-dessous le contrôle
           Frm.ToolTip.Top = Ctl.Top + Ctl.Height + 40
        Else
           'on affiche l'info - bulle au-dessus de contrôle
           Frm.ToolTip.Top = Ctl.Top - Frm.ToolTip.Height - 40
        End If
        'on cherche si on affiche l'info - bulle alignée à droite
        'ou à gauche du contrôle
        If Ctl.Left + Frm.ToolTip.Width < Frm.ScaleWidth Then
           'on aligne le bord gauche de l'info - bulle
           'avec le bord gauche du contrôle
           Frm.ToolTip.Left = Ctl.Left
        Else
           'on aligne le bord droit de l'info - bulle
           'avec le bord droit du contrôle
           Frm.ToolTip.Left = Ctl.Left - Frm.ToolTip.Width + Ctl.Width
        End If

        'on active le compteur de temps
        'pour l'affichage de l'info - bulle
        'l'info - bulle sera affichée quand
        'l'intervalle de temps spécifié dans
        'la propriété Interval sera écoulée
        'Par exemple, si Interval=500, l'info - bulle
        'sera affichée après 500 ms
        Frm.Timer1.Enabled = True

    'si l'argument OnOff est faux
    Else
        'on efface l'info - bulle
        Frm.ToolTip.Visible = False
        'l'objet est placé à l'arrière (1) de la hiérarchie
        'au sein de son niveau graphique
        Frm.ToolTip.ZOrder 1
        'on évite que l'évènement Timer déclenché
        'par le contrôle Timer1 s'effectue
        Frm.Timer1.Enabled = False
    End If

End Sub
```

Fin du document d'analyse

Par ailleurs, le retour d'informations vers l'utilisateur a fait l'objet de soins particuliers. En fait, les temps d'attente ont été gérés par des changements de forme du pointeur de la souris (flèche qui se transforme en sablier par exemple), un écran d'attente au démarrage de l'application (communément appelé «splash screen»), des boîtes de dialogue avec une barre de progression, ...
Des outils graphiques (PaintShopPro et ImageEdit) ont été nécessaires pour créer les images BITMAP et les icônes (soulé) utilisées dans le logiciel.

On peut également souligner que les écrans de l'application devaient disposer de l'écran entier lors de l'exécution. Pour cacher les diverses barres de tâches qui peuvent être affichées sur le bureau (l'écran principal Windows), j'ai utilisé des fonctions API.

Remarque : les fonctions API (Application Programming Interface) de Windows sont contenues dans des bibliothèques de liens dynamiques (DLL) d'extension EXE (EXEActiveX) ou DLL (Dynamic Link Library). Ces DLL sont à la base du fonctionnement des environnements de type WINDOWS.
Les avantages liés à l'utilisation des DLL se résument ainsi :

- Elles sont déjà présentes sur le PC utilisant sous WINDOWS.
- Elles fonctionnent sans problème.
- Elles permettent d'avoir accès à des fonctions avancées de l'OS.

En l'occurrence, deux fonctions sont parfaitement adaptées à la situation, FindWindows() et SetWindowsPos(). FindWindow() retrouve une fenêtre en fonction de son nom et de la classe (le «moule ») à partir de laquelle elle a été définie. La fonction renvoie le handle de la fenêtre trouvée ou la valeur NULL si aucune fenêtre ne correspond aux paramètres indiqués. Pour information, le handle est un identifiant d'objet, définissant de manière unique un objet.
SetWindowsPos() permet de changer la taille, la position ou le Z-Order (ordre d'apparition des fenêtres à l'écran) d'une fenêtre. La fonction renvoie TRUE si l'opération de modification aboutit, FALSE sinon.
Je les ai utilisées ainsi dans l'application :

```
Public Function cacherbarretaches() As Boolean

        On Error Goto GestErreur
        Dim Retour As Boolean ‘valeur à retourner
        Dim hWin As Long 'handle de la fenêtre cherchée (ici la barre des tâches Windows)

        hWin = FindWindow("Shell_traywnd", "") ‘le 1° argument est le nom de la classe à partir de laquelle la
        ‘fenêtre a été définie, le 2° désigne un nom particulier de fenêtre ou toutes les fenêtres ici avec ""

        Retour = SetWindowPos hWin, _ ‘handle de la fenêtre à modifier,
                0, _ ‘handle identifiant la fenêtre dans le Z-Order
                0, _ ‘nouvelle coordonnée x du coin supérieur gauche
                0, _ ‘nouvelle coordonnée y du coin supérieur gauche
                0, _ ‘nouvelle largeur
                0, _ ‘nouvelle hauteur
                &H80 ‘l’argument &H80 permet de cacher une fenêtre (voir la Visionneuse d’API, charger le
                        ‘fichier Win31api.txt et choisir Constantes puis aller à l’identificateur SWP)

        cacherbarretaches = Retour

        Exit Function

GestErreur :
        Debug.Print " Problème dans cacherbarretaches "  & err.Number

End Function
```

```
Public Function restaurerbarretaches() As Boolean

        On Error Goto GestErreur
        Dim Retour As Boolean ‘valeur à retourner
        Dim hWin As Long

        hWin = FindWindow("Shell_traywnd", "")
        Retour = SetWindowPos hWin, 0, 0, 0, 0, 0, &H40 ‘l’argument &H40 permet de montrer une fenêtre
        restaurerbarretaches = Retour

        Exit Function
GestErreur :
        Debug.Print " Problème dans retaurerbarretaches "  & err.Number

End Function
```

c) Modules de configuration de l’application

Les modules de configuration de l’application concernent la partie du programme qui sert à la recherche des informations sur le PC (l’OS plus précisément) et à la sauvegarde de choix faits pendant l’utilisation du logiciel.

Comme le logiciel est prévu pour un fonctionnement en version anglaise, espagnole et française, il est nécessaire, lors du premier lancement du programme, de connaître la langue utilisée par Windows ; ainsi le logiciel se lance pour la première fois avec l’information trouvée. Un module spécifique regroupe toutes les fonctions nécessaires à cette tâche. Le traitement décrit ci – dessus se déroule de la manière suivante :

```
Recherche de la version de Windows (grâce notamment à l’API GetVersion )
Si Windows 3.x alors
|       Recherche de la langue utilisée dans WIN.INI (avec, par exemple, l’utilisation de l’API
|       GetWindowsDirectory pour connaître le chemin et le nom complet du répertoire d’installation
|       de Windows, qui par défaut est C:\WINDOWS)
Sinon
|       Recherche de la langue utilisée dans la base de registre Windows (à la clé
|       HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\Locale)
Finsi
```

Remarque : Windows 3.x utilise les fichiers INI pour avoir des informations sur le matériel et les applications utilisés. Les inconvénients liés à ce concept sont nombreux, on peut citer la limite fatidique des 64 Ko pour la taille des fichiers par exemple. Le registre de Windows 95/98 (fichiers SYSTEM.DAT et USER.DAT) et NT (fichiers «ruches ») prend la relève des fichiers INI, offrant notamment un emplacement d’informations système centralisé, hiérarchisé, sécurisé, accessible à distance, …

Pour illustrer le recherche d’informations sur Windows, je présente un extrait du module infowin.bas avec la fonction recupwindir() qui renvoie le chemin du répertoire d’installation de Windows. Encore une fois, pour donner une idée de la manière dont le code a été écrit, j’ai inclus les en-têtes du module et de la fonction.

Début de l’extrait du module

```
'*************************************************************************
'*
'* Liste des projets     : UN - ITI
'* Objet du module       :
'*               - module permettant de récupérer la langue utilisée par Windows
'*               recherche dans la base de registre si Windows 32 bits, dans Win.ini si Windows 16 bits,
'*               - la fonction recupwinlangue renvoie un String qui contient la langue utilisée
'*               ou "?" si la recherche n'a pas aboutie
```

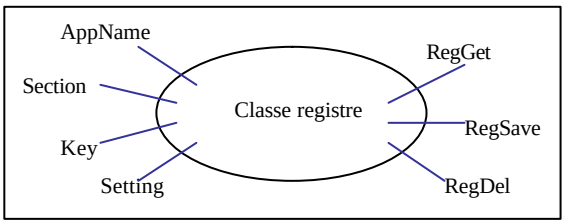
```
'*               - recupdirwindows → chemin d'installation de Windows
'*
'* Interface          :
'*           Dim langue as String
'*           langue=recupwinlangue
'*           ...
'*           Dim dirWin as String
'*           dirWin = recupdirwindows
'*
'*************************************************************************

'*************************************************************************
'*
'* Répertoire              : \itivb
'* Nom du fichier          : infowin.bas
'* Nom du module           : infowin
'* Nom du fichier d'analyse  :
'* Auteur                  : David ROUSSE
'* Date de création        : 28/01/1999
'* Date de mise a jour     : 31/01/1999
'* Validé par              :
'* Date de validation      :
'* Indice du module        : 0
'*
'*************************************************************************

'*************************************************************
' Déclaration Fonctions API
'*************************************************************
#If Win16 then
   'fonction API pour Windows 16 bits déterminant
   'le chemin d'installation de Windows
    Private Declare Function GetWindowsDirectory _
   Lib "Kernel" ( _
      ByVal lpBuffer As String, _
      ByVal nSize As Integer _
   ) As Integer
#Elseif Win32 then
   'fonction API pour Windows 32 bits déterminant
   'le chemin d'installation de Windows
    Private Declare Function GetWindowsDirectory _
   Lib "kernel32" Alias "GetWindowsDirectoryA" ( _
      ByVal lpBuffer As String, _
      ByVal nSize As Long _
   ) As Long

#End if

'*************************************************************
'* NOM PROJET : UN -ITI
'* NOM : recupdirwindows
'* DATE CREATION : 27/01/1999
'* DATE MISE A JOUR : 31/01/1999
'* INDICE : 0
'* AUTEUR : David Rousse
'* DESCRIPTION :    renvoie le répertoire d'installation de Windows
'                utilise une fonction API appelée GetWindowsDirectory
'                Si le répertoire n'a pas pu être trouvé, renvoie «? »
```

```
'* SYNTAXE : Dim dirWin as String
'*                  dirWin = recupdirwindows
'* PORTEE : publique
'* N.B. : utilise la fonction API GetWindowsDirectory
'*************************************************************

Public Function recupdirwindows() As String

   'détermination du répertoire d'installation de Windows
    On Error GoTo errdirwindows
    Dim dirwindows As String 'répertoire d'installation de Windows
    Dim retour As Integer
    Const MAXDIRWINDOWS = 144
   dirwindows = Space$(MAXDIRWINDOWS)
    retour = GetWindowsDirectory(dirwindows, MAXDIRWINDOWS)
    dirwindows = Left$(dirwindows, retour) 'on coupe à l'octet à 0
    recupdirwindows = dirwindows
    Exit Function

'gestion des erreurs
errdirwindows:
   recupdirwindows = "?"
   Debug.Print "recupdirwindows " & err.Number

End Function
```

Fin de l'extrait du module

Une interface précise pour chaque module est définie, c'est à dire que seules quelques variables et quelques fonctions sont publiques (connues du reste de l'application). Le reste du code qui permet au module de remplir son travail reste inconnu du «monde extérieur ».
Cette **approche «objet»** est encore plus visible dans les modules relatifs à la sauvegarde ou à la restauration des paramètres choisis par l'utilisateur (langue et port série).

En effet, deux paramètres sont à sauvegarder entre chaque utilisation de l'application. En ce qui concerne la langue utilisée, il est possible d'en changer à partir de l'écran principal par l'appui sur un simple bouton. Le choix sélectionné est alors sauvegardé. De même, un écran permet de choisir le port série que le logiciel doit utiliser pour communiquer. Ces options sont sauvegardées grâce à un **module de classe** appelé registre. Ce module donne la possibilité d'enregistrer ou de lire des paramètres sur un support de stockage (disque dur par exemple).
Lorsque l'on désire faire appel à ces fonctionnalités, on crée une instance de la classe registre, c'est à dire que l'on utilise un élément en mémoire créé selon le moule registre. Ce procédé est très utile car il donne la possibilité de modifier la manière dont le traitement de l'action demandée est réalisé sans impact sur le reste de l'application, il est seulement nécessaire de conserver l'interface de l'objet.
En fait, ce module de classe fournit les outils suivants :



- Propriétés :
  Appname → nom de l'application
  Section → section
  Key → clé
  Setting → valeur à lire ou écrire
- Méthodes :
  RegGet → pour lire une valeur à l'emplacement Appname-Section-Key-Setting
  RegSave → pour effectuer l'enregistrement dans la base de registre
  RegDel → pour effacer l'enregistrement dans la base de registre

Je présente ci –dessous la manière de créer une propriété et méthode, en l'occurrence la propriété Key et la méthode RegGet.

```
'Key
'création de la propriété en lecture
Public Property Get Key() As String
   Key = msKey
End Property
'création de la propriété en écriture
Public Property Let Key(By Val sKey As String)
   msKey = sKey
End Property

'RegGet
'appelle la fonction GetSetting de VB
'on n'utilise pas le paramètre optionnel Default de cette fonction
'donc si la clé msKey n'est pas trouvée dans la Base de Registres
'la fonction GetSetting renvoie une chaîne vide
Public Sub RegGet()
   msSetting = GetSetting(msAppname, msSection, msKey)
End Sub
```

Pour utiliser un objet registre, on écrit les lignes de code suivantes :

```
'création d'une instance de la classe registre
Dim basereg As New registre
…
'sauvegarde de la langue dans la Base de Registres ou
'dans souleiti.ini (si souleiti.exe est le nom de l'exécutable)
basereg.Appname = App.Title
basereg.Section = "Settings"
basereg.Key = "numlangue"
basereg.Setting = numlangue
basereg.RegSave
```

d) Gestion des communications série

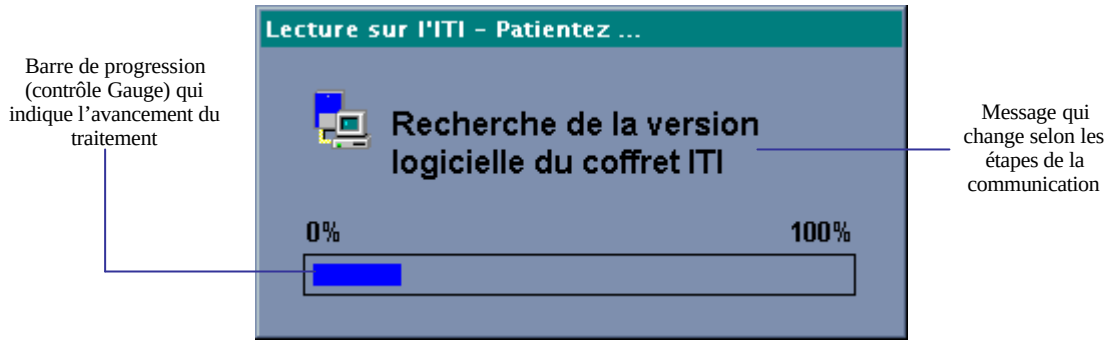
L'un des objectifs principaux du logiciel est de pouvoir paramétrer le coffret ITI ou de lire la configuration de celui-ci via une liaison série.
Avant chaque communication avec l'ITI, il est nécessaire d'initialiser la liaison série ; la séquence suivante s'en charge :

```
'initialisation de la liaison série
With Frm.ctlCom
    .CommPort = com 'choix port de communication selon le contenu de la variable com
    .Settings = "9600,N,8,1" 'les paramètres du port sont 9600 Bds, pas de parité, 8 bits de données, 1 bit de
    'stop
    .InputLen = 0 'indique au contrôle de lire la totalité du tampon d'émission
    .InBufferSize = 1024 'la taille du buffer de réception est par défaut 1024 octets
    .OutBufferSize = 512 'la taille du buffer d'émission est par défaut 512 octets
    .PortOpen = True 'ouverture du port
End With
```

En fait, deux schémas de communication peuvent se présenter :

- Le PC envoie une trame de **demande de configuration** à l'ITI ; l'ITI répond par la même trame (pour indiquer qu'il a «compris » la demande) suivie de la trame qui contient les informations demandées.
- Le PC envoie un **ordre de paramétrage** à l'ITI ; l'ITI répond en renvoyant la même trame pour acquittement de l'ordre.

Pendant la communication entre le PC et l'ITI, une feuille d'attente est affichée à l'écran, pour indiquer à l'utilisateur ce que le logiciel est en train de faire. Par exemple, lors de la lecture de données sur l'ITI, on affiche la feuille suivante :



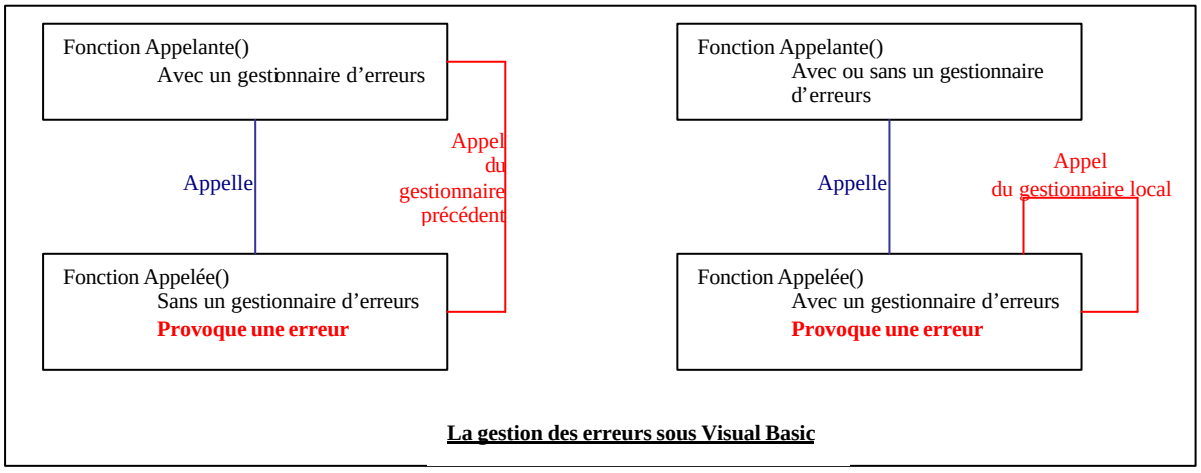
Remarque : des écrans similaires sont affichés pendant les opérations d'entrées / sorties sur le disque.

e) Débogage

Lorsqu'une application s'interrompt sans raison particulière, c'est qu'une erreur d'exécution que le programmeur n'a pas prévue s'est produite.
Cette **gestion d'erreurs** m'est apparue comme l'un des aspects les plus fastidieux de la programmation. En règle générale, tous les efforts réalisés pour rendre le logiciel plus fiable sont transparents pour l'utilisateur final. Mais je me suis rendu compte que cet effort de programmation permet d'obtenir un logiciel plus stable.

En l'occurrence, la phase de débogage des modules relatifs à la communication entre le logiciel et l'ITI est à mettre en lumière. En effet, les données envoyées ou reçues sur le port COM ont fait l'objet de tests rigoureux. L'emploi de l'instruction Debug.Print (pour écrire des données dans la fenêtre de débogage), l'exécution pas à pas de certaines parties du code, l'ajout d'expressions «espionnes » (pour observer le comportement de variables au cours de l'exécution), la mise en place du piégeage d'erreurs en ligne (avec l'instruction On Error Goto err par exemple) sont autant de techniques qui m'ont permis de créer un code source «propre ».

Plus généralement, la gestion d'une erreur sous Visual Basic peut s'illustrer ainsi :



La gestion des erreurs sous Visual Basic

En conséquence, toutes les fonctions de l'application ainsi que les parties de code «critique » ont été pourvues d'un piégeage en ligne des erreurs, avec une étiquette et une instruction de branchement automatique On Error Goto errFunction. Cette méthode me paraît simple et efficace si elle est combinée avec la vérification des valeurs de retour des fonctions.

CONCLUSION

Ces dix semaines de stage constituent une première expérience très enrichissante pour l'apprentissage du développement d'un logiciel.
Visual Basic m'est apparu comme un langage à la fois simple et performant, donnant la possibilité d'écrire des programmes pour Microsoft Windows rapidement et à faible coût.

Au fil de la conception, j'ai pu me familiariser avec des étapes charnières de l'ensemble du processus de réalisation, la définition des besoins, l'analyse des problèmes et la mise en application des solutions.
L'adoption de conventions de programmation s'est avérée nécessaire pour permettre à plusieurs personnes de s'intégrer au projet.
De plus, j'ai pris conscience de l'importance de la réflexion avant la réalisation: une analyse réussie évite des pertes de temps inutiles lors de l'écriture du code.
En résumé, la rigueur et l'esprit critique sont les deux maître – mots à suivre lors de la conception d'un logiciel.

Le stage a donc été, pour ma part, une réussite. Je tiens par conséquent à remercier les membres du service Laboratoire électronique du département Appareillage – Distribution de l'entreprise SOULÉ pour leur aide et leur conseil. En particulier, travailler avec Monsieur **Jean Gabriel Duvigneau** fut un réel plaisir, ses compétences et sa bonne humeur ayant contribuées à rendre le stage productif.

ANNEXES

Annexe 1

Je présente dans le tableau qui suit les activités à caractère professionnel que j'ai réalisées au cours de stages en milieu professionnel et pendant les horaires d'actions professionnelles.

Numéro	*Intitulé de l'action*	*Description de l'action*	*Année de réalisation*
1	**Assemblage**	**Architecture matérielle**	**1° année**
2	**Macros Excel**	**Outils et logiciels bureautiques**	**1° année**
3	**Formation Internet**	**Logiciels de communications**	**1° année**
4	**Création d'un site Web**	**Développement d'applications informatiques et génie logiciel**	**1° année**
5	**Fichiers de ressources**	**Développement d'applications informatiques et génie logiciel**	**2° année**
6	**Conversion €uro**	**Développement d'applications informatiques et génie logiciel**	**2° année**
7	**Contrôle ActiveX**	**Développement d'applications informatiques et génie logiciel**	**2° année**

Annexe 2

Les certificats de stage de première et deuxième années de B.T.S. Informatique de Gestion sont présentés dans les deux pages qui suivent.