

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Suivi des versions-révisions et des validations du document.			
<p>Ce document annule et remplace tout document diffusé de version-révision antérieure.</p> <p>Dès réception de ce document, les destinataires ont pour obligation de détruire les versions-révisions antérieures, toutes les copies, et de les remplacer par cette version.</p> <p>Si les versions-révisions antérieures sont conservées pour mémoire, les destinataires doivent s'assurer qu'elles ne peuvent être confondues avec cette présente version-révision dans leur usage courant.</p>			
Version.	Date.	Auteurs.	Création, modification ou validation.
A	19 nov. 2003	JPD.	Création.
B	9 mar. 2004	JPD.	Mise à jour pour Up ! Vtm et le répertoire genere .

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

1 Tables

1.1 Table des matières

1	Tables.....	2
1.1	Table des matières	2
1.2	Table des illustrations.....	4
2	Références.....	6
2.1	Glossaire.....	6
2.2	Ressources	6
3	Introduction	7
3.1	Objet du document	7
3.2	Audience.....	7
3.3	Pré-requis	7
4	Normes de programmation	8
4.1	Formatage des fichiers sources	8
4.1.1	Indentation	8
4.1.2	Présentation des blocs.....	8
4.1.3	Terminaison d'un fichier	9
4.2	Dénomination.....	9
4.2.1	Principe général de dénomination	9
4.2.2	Dénomination d'un module	9
4.2.3	Dénomination d'une constante	11
4.2.4	Dénomination d'un énuméré.....	11
4.2.5	Dénomination d'un type de données	12
4.3	Déclaration des définitions	12
4.3.1	Variable.....	12
4.3.2	Fonction	13
4.4	Type de données.....	15
4.4.1	Types de base	15
4.4.2	Types construits	16
4.4.3	Types de fonction	16
4.5	Commentaires	16
4.5.1	En-tête d'un fichier	16
4.5.2	Déclaration.....	17
4.5.3	Macro-algorithme.....	17
4.5.4	Corrections apportées au module.....	18
4.6	Codage	19
4.6.1	Identificateurs prédéfinis	19
4.6.2	Dimensionnement des tableaux	20
4.6.3	Formatage des expressions.....	21
4.6.4	Conversion de type	21
4.6.5	Fonctions standards	22
4.6.6	Usage des pointeurs de fonction	23
4.6.7	Paramètres	23
4.7	Codifications	24
4.7.1	Booléens	24
4.7.2	Caractère Unicode	24
5	Architecture.....	26
5.1	Architecture d'un module	26
5.2	Architecture d'une bibliothèque.....	28
5.2.1	Composants d'une bibliothèque	29
5.2.2	Fichiers sources	29
5.2.3	Interface avec un logiciel de base	30
5.3	Architecture d'un composant.....	31
5.3.1	Programmation orientée objet.....	31
5.3.2	Architecture simplifiée	33
5.3.3	Architecture complète.....	36
5.4	Interface avec la machine	41
6	Organisation des versions-révisions.....	44
6.1	Répertoire Archive	44
6.2	Répertoire Documents	45

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

6.3	Répertoire Normes	46
6.4	Répertoire Tmp.....	46
6.5	Répertoire « UpsVRC ».....	46
6.5.1	Répertoire « Plate-forme »	46
6.5.2	Répertoire Dev	48
6.5.3	Répertoire En_avance_de_phase	50
6.5.4	Répertoire Essai	50
6.5.5	Répertoire Internet	50
6.5.6	Répertoire Tmp.....	51
6.6	Fichiers propres à une plate-forme.....	51
7	Normes de sûreté de fonctionnement	52
7.1	Gestion des ressources.....	52
7.2	Initialisation	52
7.3	Protection des unions.....	52
7.4	Contrôle des adresses	53
7.4.1	Ressources.....	53
7.4.2	Objets.....	54
7.5	Contrôle des codes retour.....	54
7.6	Contrôle de la table des méthodes.....	54
7.7	Contrôle de la taille des tampons.....	54
7.8	Contrôle des paramètres obligatoires	55
7.9	Contrôle des cas imprévus.....	55
7.10	Formats d'enregistrement	55
7.11	Redondance de code	55
7.12	Multi-tâches	56
8	Index des Application Program Interfaces.....	57
8.1	Par module.....	57
8.1.1	Up ! Kernel.....	57
8.1.2	Up ! Mathematical	57
8.1.3	Up ! Natural Language Support	58
8.1.4	Up ! Network.....	59
8.1.5	Up ! System	59
8.1.6	Up ! Virtual Technical Machine	63
8.2	Par dénomination	63
8.2.1	Via le nom de l'API standard Up ! Virtual Technical Machine	63
8.2.2	Via le nom de l'API standard C Posix.....	69

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

1.2 Table des illustrations

Texte 1 – Exemple d'indentation	8
Texte 2 – Exemple d'indentation	9
Texte 3 – Contre-exemple d'indentation	9
Texte 4 – Exemple de dénomination.....	9
Texte 5 – Contre-exemple de dénomination.....	9
Tableau 6 – Appellation des modules d'Up ! Application System.....	11
Texte 7 – Exemple de dénomination d'une constante – 1	11
Texte 8 – Exemple de dénomination d'une constante – 2	11
Texte 9 – Exemple d'emploi de la dénomination d'une constante	11
Texte 10 – Exemple de dénomination d'un énuméré – 1	11
Texte 11 – Exemple de dénomination d'un énuméré – 2	12
Texte 12 – Exemple d'emploi de la dénomination d'un énuméré	12
Texte 13 – Exemple de dénomination d'un type – 1.....	12
Texte 14 – Exemple de dénomination d'un type – 2.....	12
Texte 15 – Exemple d'emploi de la dénomination d'un type	12
Texte 16 – Exemple de déclaration d'une variable – 1.....	13
Texte 17 – Exemple de déclaration d'une variable – 2.....	13
Texte 18 – Exemple de déclaration d'une fonction – 1.....	13
Texte 19 – Exemple de déclaration d'une fonction – 2.....	13
Texte 20 – Exemple de déclaration du résultat d'une fonction	14
Texte 21 – Contre-exemple de déclaration du résultat d'une fonction.....	14
Texte 22 – Exemple de déclaration du résultat d'une procédure.....	14
Texte 23 – Exemple de déclaration du prototype d'une fonction – 1	14
Texte 24 – Contre-exemple de déclaration du prototype d'une fonction	14
Texte 25 – Exemple de déclaration du prototype d'une fonction – 2.....	15
Tableau 26 – Type de données d'Up ! Virtual Technical Machine.....	15
Texte 27 – Exemple de permutation d'octets.....	16
Texte 28 – Exemple de type de fonction.....	16
Texte 29 – Contre-exemple de type de fonction.....	16
Texte 30 – Exemple de commentaire pour un en-tête de fichier	17
Texte 31 – Exemple de commentaire pour une déclaration.....	17
Texte 32 – Exemple de commentaire pour un macro-algorithme – 1	18
Texte 33 – Exemple de commentaire pour un macro-algorithme – 2	18
Texte 34 – Exemple de commentaire pour une correction apportée à un module.....	19
Texte 35 – Exemple de dimensionnement des tableaux.....	20
Texte 36 – Exemple de formatage d'une expression	21
Texte 37 – Exemple de formatage d'une expression	21
Texte 38 – Contre exemple de formatage d'une expression.....	21
Texte 39 – Contre-exemple d'affectation dans une expression	21
Texte 40 – Exemple de conversion de type – 1.....	22
Texte 41 – Exemple de conversion de type – 2.....	22
Texte 42 – Exemple d'usage d'une expression sans conversion de type	22
Texte 43 – Contre-exemple d'usage d'une conversion de type dans une expression	22
Texte 44 – Localisation des fonctions standards	23
Texte 45 – Exemple d'usage des pointeurs de fonction.....	23
Texte 46 – Contre-exemple d'usage d'une conversion de type dans une expression	23
Texte 47 – Exemple de contrôle des paramètres obligatoires	24
Texte 48 – Exemple d'emploi des codifications Unicode	25
Figure 49 – Organisation logique d'un module	28
Figure 50 – Organisation logique d'une bibliothèque	29
Figure 51 – Organisation physique d'une bibliothèque	30
Figure 52 – Organisation physique d'un composant	30
Figure 53 – Exemple d'appel inter-module pour utiliser une API native	31
Texte 54 – Exemple de protection contre l'inclusion multiple et de déclaration du langage cible	33
Texte 55 – Exemple de paramètre complexe – Architecture simplifiée.....	34
Texte 56 – Exemple d'interface de traitements – Architecture simplifiée	34
Texte 57 – Exemple d'exportation de la fonction de démarrage – Architecture simplifiée	35
Texte 58 – Exemple de prototypes des API encapsulées – Architecture simplifiée.....	35
Texte 59 – Exemple d'un composant d'interface avec Up ! Module – Architecture simplifiée.....	35
Texte 60 – Exemple d'encapsulation d'une API – Architecture simplifiée.....	36
Texte 61 – Exemple d'interface de données d'une bibliothèque – Architecture complète.....	37
Texte 62 – Exemple d'interface de traitements d'une bibliothèque – Architecture complète.....	37
Texte 63 – Exemple d'alias sur l'interface de traitements d'une bibliothèque – Architecture complète	38
Texte 64 – Exemple de prototypes des fonctions exportées – Architecture complète.....	38
Texte 65 – Exemple d'un composant d'interface avec Up ! Module – Architecture complète	39

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Texte 66 – Exemple d'un composant cœur – Architecture complète.....	40
Diagramme 67 – Exemple d'usage d'une interface avec la machine	41
Texte 68 – Exemple d'interface avec la machine	42
Texte 69 – Exemple de publication d'une interface avec la machine	43
Diagramme 70 – Organisation des versions-révisions	44
Diagramme 71 – Organisation des documents internes	45
Diagramme 72 – Organisation des versions-révisions d'Up ! Application System	46
Diagramme 73 – Organisation du répertoire d'une plate-forme.....	47
Diagramme 74 – Organisation du répertoire objet.....	47
Diagramme 75 – Organisation du répertoire des développements	48
Diagramme 76 – Organisation du répertoire des sources générés automatiquement.....	48
Diagramme 77 – Organisation du répertoire de production	48
Diagramme 78 – Organisation du répertoire de qualification	49
Diagramme 79 – Organisation du répertoire des sources écrits manuellement.....	50
Diagramme 80 – Organisation du répertoire de travail.....	50
Diagramme 81 – Organisation du répertoire du site Internet	51
Texte 82 – Exemple de protection des unions	53
Texte 83 – Exemple de contrôle de la taille des tampons	54
Texte 84 – Exemple de contrôle d'un paramètre obligatoire.....	55
Texte 85 – Exemple de contrôle des cas imprévus	55
Texte 86 – Exemple de gestion du multi-tâches	56
Texte 87 – API d'Up ! Kernel	57
Texte 88 – API d'Up ! Mathematical	58
Texte 89 – API d'Up ! Natural Language Support	59
Texte 90 – API d'Up ! Network	59
Texte 91 – API d'Up ! System.....	63
Texte 92 – API d'Up ! Virtual Technical Machine	63
Texte 93 – Index général des APIs par leur nom selon le standard Up ! Virtual Technical Machine	69
Texte 94 – Index général des APIs par leur nom selon le standard C Posix.....	71

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

2 Références

2.1 Glossaire

Liste des définitions des termes employés.	
Ce tableau recense tous les termes, les concepts particuliers ainsi que les abréviations employés dans ce document.	
Terme, concept, abrégé.	Définition du terme, du concept ou de l'abréviation.
Interface avec la machine	Voir page 41.

2.2 Ressources

Liste des documents applicables et en référence.		
Un document est applicable à partir du moment où son contenu est validé et que l'activité ou le projet fait partie de son périmètre d'application. Il est obligatoire d'appliquer son contenu.		
Un document est en référence à partir du moment où son contenu n'est pas validé ou que l'activité ou le projet ne fait partie de son périmètre d'application. Il est recommandé d'appliquer son contenu mais cela n'est pas obligatoire.		
Un document applicable est indiqué par A1, A2, A3 , etc. Un document en référence est indiqué par R1, R2, R3 , etc.		
Index.	Nom du document.	Commentaire.
A1	UpComp-Plan Qualité-000005	Méthode documentaire.
A2	UpComp-UpsVtm-000004	Programmation en C-- .

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

3 Introduction

3.1 Objet du document

L'objet du document est de présenter le plan de programmation applicable à tous les modules écrits directement en **C--** d'**Up ! Application System**, en particulier les suivants :

- **Up ! Kernel.**
- **Up ! Mathematical.**
- **Up ! Natural Language Support.**
- **Up ! Network.**
- **Up ! Object Management System.**
- **Up ! Object Request Broker.**
- **Up ! Security Management System.**
- **Up ! Starter.**
- **Up ! System.**
- **Up ! Virtual Technical Machine.**

Up ! Virtual Technical Machine correspond à la fois :

- A des bonnes pratiques.
Elles sont recensées dans le chapitre intitulé **Normes de programmation.**
- A des standards de codification.
Elles sont recensées dans le chapitre intitulé **Normes de programmation.**
- A des techniques d'organisation du code.
Elle est présentée dans le chapitre intitulé **Architecture d'un module.**
- A la mise en oeuvre de **C--**.
Soit au travers du compilateur **C Gnu** ou soit au travers du processeur virtuel **Up ! P32.**
- A des **Application Program Interfaces (API)** standardisées.
Elle est présentée dans le chapitre intitulé **Application Program Interfaces.**

3.2 Audience

Ce document s'adresse à tout ingénieur chargé de l'étude, de la réalisation ou de la maintenance d'un module natif.

3.3 Pré-requis

Le pré-requis est la connaissance de :

- La programmation en **C--**.
- Les **Application Program Interfaces (API)** pour exploiter les ressources d'une machine physique ou le potentiel des logiciels de base.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

4 Normes de programmation

L'objet de ces normes de programmation est de rendre :

- Le code des programmes impersonnel de la sorte à être plus facile à comprendre et à maintenir.
- Le code des programmes écrit manuellement similaire à celui généré automatiquement.

Ces normes sont à appliquer sans réserve.

4.1 Formatage des fichiers sources

4.1.1 Indentation

M Les indentations sont toujours réalisées par l'insertion de trois caractères **espace** et non par l'emploi de caractère **tabulation**. Ceci permet de conserver le même formatage quel que soit l'éditeur de fichiers source utilisé.

Voici un exemple :

```

Texte non indente.
  Texte indente d'un cran.
    Texte indente de deux crans.

```

Texte 1 – Exemple d'indentation

4.1.2 Présentation des blocs

M Les blocs délimités par les caractères **accolade ouvrante {** et **accolade fermante }** sont toujours indentés d'un cran dès lors qu'ils sont imbriqués. Ceci est notamment le cas pour le corps d'un test ou d'une boucle.

Il y a une ligne pour l'ouverture du bloc et une ligne pour sa fermeture. Elles contiennent uniquement les accolades et celles-ci sont alignées. Cela permet de distinguer visuellement le bloc de façon aisée.

Voici un exemple :

```

void MonAppel (void)
{
...
for ( ; )
{
...
if (...)
{
...
}
else
{
...
}
...
}
...
}

```

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Texte 2 – Exemple d'indentation

Voici un contre-exemple :

```
void MonAppel(void) {
...
for ( ; ) {
...
if (...) {
}
else {
...
}
...
}
...
}
```

Texte 3 – Contre-exemple d'indentation

4.1.3 Terminaison d'un fichier

M

Un fichier se termine toujours par un caractère saut de ligne. Cela évite des déconvenues avec des outils – éditeurs, compilateurs, etc. – comportant des dysfonctionnements.

4.2 Dénomination

4.2.1 Principe général de dénomination

Une dénomination est composée d'une juxtaposition de mots donnant son sens à l'objet nommé, chaque mot commençant par une majuscule, le reste étant écrit en minuscule.

Voici des exemples :

```
CeciEstUnExemple CeciEstUnSecondExemple2
```

Texte 4 – Exemple de dénomination

Voici des contre-exemples :

```
ceciestunmauvaisexemple
CECIESTUNMAUVAISEXEMPLE
Ceci_Est_Un_Mauvais_Exemple
```

Texte 5 – Contre-exemple de dénomination

4.2.2 Dénomination d'un module

Tous les modules comportent un préfixe composé de :

- L'identifiant de la société.
Pour **Up ! Company**, le préfixe est **Ups** pour rappeler **Up ! Software**.
- L'identifiant du module.
Il s'agit généralement soit :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- Des trois premières lettres du nom commercial du module.
- Des trois initiales des premiers mots du nom commercial du module.
- Des trois premières consonnes du nom commercial du module.

Pour **Up ! System**, l'identifiant du module est **Sys**. L'identifiant du module doit être suffisamment discriminant.

Voici les appellations des modules d'**Up ! Application System** :

Préfixe	Nom littéral du module
5gl	<i>Up ! Fifth Generation Language – Basic.</i>
Ana	<i>Up ! Analyzer.</i>
Cmp	<i>Up ! Compiler.</i>
Com	<i>Up ! Component Object Broker – Run-time.</i>
Cp1	<i>Up ! Compiler – Level 1.</i>
Crb	<i>Up ! Corba – Run-time.</i>
Gc1	<i>Up ! C / C++ Generator.</i>
Gcb	<i>Up ! Corba – Server Adapter.</i>
Gcm	<i>Up ! Component Object Module – Server Adapter.</i>
Gjv	<i>Up ! Java – Server Adapter.</i>
Gl1	<i>Up ! Fifth Generation Language Generator – Level 1.</i>
Gr1	<i>Up ! Grammar.</i>
Gun	<i>Up ! Network Adapter – Server Adapter.</i>
Icb	<i>Up ! Corba – Client Adapter for Interface Definition Language.</i>
Icm	<i>Up ! Component Object Module – Client Adapter for Interface Definition Language.</i>
Ijv	<i>Up ! Java – Client Adapter.</i>
Irc	<i>Up ! Corba – Client Adapter for Interface Repository.</i>
Jav	<i>Up ! Java – Run-time.</i>
Krn	<i>Up ! Kernel.</i>
Lg1	<i>Up ! Fifth Generation Language – Level 1.</i>
Mat	<i>Up ! Mathematical.</i>
Mod	<i>Up ! Module.</i>
Msn	<i>Up ! Microsoft Network.</i>
Nap	<i>Up ! Named Pipes.</i>
Net	<i>Up ! Network.</i>
Nls	<i>Up ! Natural Language Support.</i>
Oms	<i>Up ! Object Management System.</i>
Orb	<i>Up ! Object Request Broker.</i>
Sec	<i>Up ! Security Management System.</i>

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Str	<i>Up ! Starter.</i>
Sys	<i>Up ! System.</i>
Vm	<i>Up ! Virtual Machine.</i>
Tcp	<i>Up ! Transmission Control Protocol.</i>
Tlb	<i>Up ! Component Object Module – Client Adapter for Type Library.</i>
Win	<i>Up ! Windows.</i>

Tableau 6 – Appellation des modules d'Up ! Application System

a Dans la suite du document, les exemples sont donnés pour le module *Mon Module* de préfixe *MonMdl*.

4.2.3 Dénomination d'une constante

Le nom d'une constante définie par l'instruction `#define` comporte le préfixe particulier *CO_* permettant de les identifier. Voici un exemple :

```
CO_TailleBuffer
```

Texte 7 – Exemple de dénomination d'une constante – 1

L'alias qui suit la déclaration de la constante est le nom de la constante écrit normalement.

Voici un exemple :

```
#define CO_TailleBuffer 256
```

Texte 8 – Exemple de dénomination d'une constante – 2

Voici un exemple d'emploi de la dénomination d'une constante :

```
TypUpsVmUnicode Tmp[CO_TailleBuffer+1];
```

Texte 9 – Exemple d'emploi de la dénomination d'une constante

4.2.4 Dénomination d'un énuméré

Le nom d'un énuméré défini par l'instruction `enum` comporte le préfixe particulier *Enu* permettant de les identifier. Voici un exemple :

```
EnuUpsModCible
```

Texte 10 – Exemple de dénomination d'un énuméré – 1

M L'alias qui suit l'instruction `enum` est le nom de l'énuméré écrit normalement. Les valeurs d'un énuméré sont toujours associées à un nombre entier commençant à un et s'incrémentant.

Il n'y a jamais d'alias qui suit la déclaration de l'énuméré.

Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```
enum EnuUpsModCible
{
CibleUnix=1,
CibleWindows=2
};
```

Texte 11 – Exemple de dénomination d'un énuméré – 2

Voici un exemple d'emploi de la dénomination d'un énuméré :

```
enum EnuUpsModCible Cible;
```

Texte 12 – Exemple d'emploi de la dénomination d'un énuméré

4.2.5 Dénomination d'un type de données

Le nom d'un type de données défini par l'instruction **typedef** comporte le préfixe particulier **Typ** permettant de les identifier. Voici un exemple :

```
TypUpsModDonnees
```

Texte 13 – Exemple de dénomination d'un type – 1

L'alias qui suit les instructions **struct** et **union** est le nom du type écrit en minuscule.

L'alias qui suit la déclaration du type ou de l'union est le nom du type écrit normalement et il correspond toujours à une adresse de la définition.

Voici un exemple :

```
/*
typedef struct typupsmoddonnees
/* Objet : Interface de donnees de Ups Module.
/*
{
...
} *TypUpsModDonnees;
```

Texte 14 – Exemple de dénomination d'un type – 2

Voici un exemple d'emploi de la dénomination d'un type :

```
sizeof(struct TypUpsModDonnees)
TypUpsModDonnees Donnees;
```

Texte 15 – Exemple d'emploi de la dénomination d'un type

4.3 Déclaration des définitions

4.3.1 Variable

Il existe deux sortes de variable :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVm-000003-A Plan de programmation.doc		

- **Une variable propre à un composant.**
Elle est définie dans un composant et elle est uniquement utilisée dans ce composant.
En ce cas, elle est déclarée en statique sans préfixe particulier. Voici un exemple :

```
static TypUpsVmShort Essai;
```

Texte 16 – Exemple de déclaration d'une variable – 1

- **Une variable partagée par plusieurs composants.**
Elle est définie dans un composant, elle est utilisée dans ce composant ainsi que dans d'autres composants.
En ce cas, elle n'est pas déclarée en statique et elle est préfixée par le l'identifiant du module.
Voici un exemple :

```
TypUpsVmShort UpsMonMdlEssai;
```

Texte 17 – Exemple de déclaration d'une variable – 2

Les variables globales sont toujours initialisées à une valeur par défaut, le plus souvent 0 ou **NULL**, ce qui correspond à une initialisation physique réalisée par le système d'exploitation au lancement du processus.

M Pourtant, il ne faut pas considérer qu'elles ont été initialisées. Le programme doit commencer par effectuer une initialisation logique, éventuellement avec les mêmes valeurs par défaut. La raison est qu'il peut exister plusieurs instances du même module au sein du même processus parce que le code est réentrant.

4.3.2 Fonction

Il existe deux sortes de fonction :

- **Une fonction propre à un composant.**
Elle est définie dans un composant et elle est uniquement utilisée dans ce composant.
En ce cas, elle est déclarée en statique sans préfixe particulier. Voici un exemple :

```
static TypUpsVmShort Essai(...);
```

Texte 18 – Exemple de déclaration d'une fonction – 1

- **Une fonction partagée par plusieurs composants.**
Elle est définie dans un composant, elle est utilisée dans ce composant ainsi que dans d'autres composants.
En ce cas, elle n'est pas déclarée en statique et elle est préfixée par le l'identifiant du module.
Voici un exemple :

```
TypUpsVmShort UpsVmAPI UpsMonMdlEssai(...);
```

Texte 19 – Exemple de déclaration d'une fonction – 2

Le symbole **UpsVmAPI** écrit devant l'identifiant correspond à une directive de compilation obligatoire pour toutes les fonctions partagées par plusieurs composants.

M 4.3.2.1 Résultat de la fonction

Une fonction comporte toujours la déclaration de son résultat. Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVm-000003-A Plan de programmation.doc		

```
static TypUpsVmShort Essai(TypUpsVmShort A)
{
...
}
```

Texte 20 – Exemple de déclaration du résultat d'une fonction

Voici un contre-exemple

```
static Essai(TypUpsVmShort A)
{
...
}
```

Texte 21 – Contre-exemple de déclaration du résultat d'une fonction

Si une fonction ne retourne aucune valeur – nous parlons alors de procédure –, son résultat est alors déclaré **TypUpsVmVoid**. Voici un exemple :

```
static TypUpsVmVoid Essai(TypUpsVmShort A)
{
...
}
```

Texte 22 – Exemple de déclaration du résultat d'une procédure

4.3.2.2 Prototypage de la fonction

M

Une fonction est toujours déclarée avec son prototype complet comme en **C++** et non comme en standard **C**. Sinon le compilateur n'est pas capable de vérifier la cohérence des valeurs transmises aux paramètres.

Voici un exemple :

```
static TypUpsVmShort Essai(TypUpsVmShort A)
{
...
}
```

Texte 23 – Exemple de déclaration du prototype d'une fonction – 1

Voici un contre-exemple

```
static TypUpsVmShort Essai()
TypUpsVmShort A;
{
...
}
```

Texte 24 – Contre-exemple de déclaration du prototype d'une fonction

Si une fonction ne comporte aucun paramètre, son prototype est alors **TypUpVmVoid**. Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVm-000003-A Plan de programmation.doc		

```
static TypUpsVmShort Essai(TypUpsVmVoid)
{
    ...
}
```

Texte 25 – Exemple de déclaration du prototype d’une fonction – 2

4.4 Type de données

4.4.1 Types de base

Les types de données standards du langage **C++ / C** ne sont jamais utilisés dans un module sauf dans l’éventuelle bibliothèque regroupant les **Application Program Interfaces (API)** natives de celle-ci.

Il faut toujours utiliser les types de données standards d’**Up ! Virtual Technical Machine** qui sont identiques en terme de capacité de stockage quelle que soit la plate-forme d’exécution.

Voici les types de base d’**Up ! Virtual Technical Machine** :

Types standards	Nom littéral du module
TypUpsVmChar	Entier sur un octet de valeur appartenant à [-128, 127].
TypUpsVmUnsignedChar	Entier sur un octet de valeur appartenant à [0, 255].
TypUpsVmShort	Entier sur deux octets de valeur appartenant à [-32768, 32767].
TypUpsVmUnsignedShort	Entier sur deux octets de valeur appartenant à [0, 65536].
TypUpsVmLong	Entier sur quatre octets de valeur appartenant à [-2147483648, 2147483647].
TypUpsVmUnsignedLong	Entier sur quatre octets de valeur appartenant à [0, 4294967296].
TypUpsVmDouble	Réel sur huit octets.
TypUpsVmUnicode	Caractère sur deux octets.
TypUpsVmPointeurDonnees	Pointeur vers une interface de données indéfinie.
TypUpsVmPointeurTraitements	Pointeur vers une interface de traitements indéfinie.

Tableau 26 – Type de données d’Up ! Virtual Technical Machine

M

L’ordre des octets pour les types standards entiers est toujours celui attendu par le microprocesseur. Si celui lu dans un flux est différent de celui attendu par le microprocesseur, il faut permuter les octets en utilisant une des fonctions suivantes :

- **UpsSys.DecoderShort.**
- **UpsSys.DecoderUnsignedShort.**
- **UpsSys.DecoderLong.**
- **UpsSys.DecoderUnsignedLong.**

Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

if (BufferProtocoleUpsNet->EstBigEndian!=
UpsSysSUIntDonUpsMod->EstBigEndian)
Token= (*UpsSysSUIntTrtUpsSys->DecoderLong) (BufferProtocoleUpsNet
->Token) ;
else
Token=BufferProtocoleUpsNet->Token;

```

Texte 27 – Exemple de permutation d'octets

4.4.2 Types construits

M L'alignement des champs des types construits est obligatoirement quatre octets. Les modules *Up ! Kernel* et *Up ! Object Management System* réalisent des calculs d'adressage en tenant compte de cette convention.

Il est donc nécessaire de régler le compilateur en conséquence.

4.4.3 Types de fonction

Par souci de lisibilité, tous les pointeurs de fonction sont déclarés sous forme d'un type de fonction avant d'être utilisés et non déclarés directement. Voici un exemple :

```

typedef TypUpsVmVoid UpsVmAPI (*TypMonMdlAppel) (TypUpsVmVoid) ;
...
TypMonMdlAppel MonAppel ;
...
(*MonAppel) ();

```

Texte 28 – Exemple de type de fonction

Voici un contre-exemple :

```

TypUpsVmVoid UpsVmAPI (*MonAppel) (TypUpsVmVoid) ;
...
(*MonAppel) ();

```

Texte 29 – Contre-exemple de type de fonction

M Le pointeur de fonction pouvant correspondre à une fonction exportée, il est obligatoire d'écrire le symbole *UpsVmAPI* devant son identifiant.

4.5 Commentaires

4.5.1 En-tête d'un fichier

Tous les fichiers ont l'en-tête standard suivant :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

/*-----
Fichier      : upsvm.e
Objet       : Liste des definitions de Ups Virtual Machine

Module      : UpsVm
Auteur-Date : Jean-Pierre DUVAL - 20 novembre 2003.
-----
Observations

Corrections apportees : Q000001, Q000013.
----- */

```

Texte 30 – Exemple de commentaire pour un en-tête de fichier

Voici le contenu de chaque champ de cet en-tête :

- **Auteur.**
Nom de l'auteur de ce fichier. Celui-ci pourra être contacté afin de fournir des informations nécessaires à sa maintenance.
- **Date.**
Date de dernière modification de ce fichier.
- **Fichier.**
Nom du fichier avec son extension et sans son chemin d'accès.
- **Module.**
Nom du module dont le fichier fait partie.
- **Objet.**
Description synthétique de l'objet du fichier.
- **Observations.**
Liste des observations concernant ce fichier. Il y a en particulier les numéros des corrections suite aux dysfonctionnements signalés au support technique.

4.5.2 Déclaration

Toutes les déclarations de constantes, d'énumérés, de variables, de champ d'un type ont un commentaire ligne indenté de trois caractères **espace**. Voici un exemple :

```

#define CO_TailleNomFichier 256
/* Taille d'un nom de fichier. */

```

Texte 31 – Exemple de commentaire pour une déclaration

a Il existe un commentaire au sens particulier pour les variables intitulé « Variable de travail ». Il signifie que la variable est utilisée localement le temps d'un calcul intermédiaire. Aussi, elle peut être utilisée à plusieurs reprises dans le même corps de traitement sans risque d'écrasement.

4.5.3 Macro-algorithme

Les commentaires de macro-algorithme permettent de comprendre la compréhension globale du contenu du fichier en les parcourant.

Les commentaires de premier niveau, délimités par une suite de caractères **étoile ***, sont employés pour introduire :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- Une directive de compilation conditionnelle.
Les instructions **#if**, **#elif**, **#else** et **#endif**.
- La déclaration d'un énuméré.
- La déclaration d'une fonction.
- La déclaration d'un type.

Tous ces éléments doivent être commentés. Voici un exemple :

```

/*****/
TypUpsVmShort UpsVmAPI Ups5GLExpressionEstFactorisable(
    TypUpsVmSession *Session, TypUpsVmShort Numero,
    TypUpsVmShort NumeroPaquet, TypUps5GLExpression Expression,
    TypUps5GLExpression ExpressionAFactoriser)
/* Objet : Calcule si l'expression peut etre factorisee.          */
/*****/

```

Texte 32 – Exemple de commentaire pour un macro-algorithme – 1

Les commentaires de second niveau, délimités par une suite de caractères **moins -**, sont employés pour introduire une étape logique du macro-algorithme, en particulier les principaux débranchements.

```

/*-----*/
/* Construction de la negation de l'expression gauche.          */
/*-----*/

```

Texte 33 – Exemple de commentaire pour un macro-algorithme – 2

Les commentaires de troisième niveau correspondent aux dénominations employées pour les déclarations de constantes, d'énumérés, de variables, de types, de champs, de fonctions, etc.

M Contrairement à ce qu'on pourrait penser, trop de commentaire nuit à la lisibilité du programme. Seule les étapes logiques des macro-algorithmes, non évidents de part leur objet, ont besoin d'être commentés.

4.5.4 Corrections apportées au module

Lorsqu'une correction est apportée au module alors qu'il est en production – et non lorsque le module est en test suite à son développement –, la correction est explicitement tracée de la sorte :

- **Marqueur du début de la correction.**
Ce commentaire de premier niveau contient le nom de la personne réalisant la correction, la date de la correction, le numéro de dossier et la raison de la correction.
- **Ancien code en commentaire.**
Cela permet de se remémorer l'original.
- **Code corrigé.**
- **Marqueur de fin de la correction.**
Ce commentaire de premier niveau contient le numéro de dossier.

Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

/*****
/* Auteur - Date : DUVAL Jean-Pierre - 10nov. 2003 */
/* Dossier : Q000001. */
/* Objet : La taille de la chaine est mal calculee. */
/* Taille=(TypUpsVmShort)(2*CO_TailleBuffer); */
/*****
Taille=(TypUpsVmShort)(2*CO_TailleBuffer*sizeof(TypUpsVmUnicode));
/*****
/* Fin Dossier : Q000001. */
/*****

```

Texte 34 – Exemple de commentaire pour une correction apportée à un module

4.6 Codage

4.6.1 Identificateurs prédéfinis

4.6.1.1 Identification de la plate-forme cible

Les définitions suivantes permettent d'identifier la plate-forme cible de compilation :

- **PLATEFORMEMAC.**
Cette définition existe si la plate-forme cible est *Macintosh*.
- **PLATEFORMEOS390.**
Cette définition existe si la plate-forme cible est *Os 390*.
- **PLATEFORMEOS400.**
Cette définition existe si la plate-forme cible est *Os 400*.
- **PLATEFORMEUNIX.**
Cette définition existe si la plate-forme cible est *Unix*.
- **PLATEFORMEWINDOWS.**
Cette définition existe si la plate-forme cible est *Windows*.

M

Seules ces définitions doivent être utilisées dans les modules génériques pour éventuellement identifier la plate-forme d'exécution. Par contre, dans le module *Up! System*, les définitions plus spécifiques peuvent être employées telles *SOLARIS27*.

4.6.1.2 Identification des appels de service supportés

Les définitions suivantes permettent de connaître les normes d'appel de service supportées :

- **PLATEFORMEALE.**
Cette définition existe si la plate-forme cible supporte la norme d'appel de service *Ale / Rfc*.
- **PLATEFORMECORBA.**
Cette définition existe si la plate-forme cible supporte la norme d'appel de service *Corba*.
- **PLATEFORMEDCOM.**
Cette définition existe si la plate-forme cible supporte la norme d'appel de service *DCom*.
- **PLATEFORMEJAVA.**
Cette définition existe si la plate-forme cible supporte la norme d'appel de service *Java*.
- **PLATEFORMESOAP.**
Cette définition existe si la plate-forme cible supporte la norme d'appel de service *Soap*.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

4.6.1.3 Identification du potentiel de la plate-forme

Les définitions suivantes permettent de connaître le potentiel de la plate-forme :

- **PLATEFORMEGRAPHIQUE.**
Cette définition existe si la plate-forme cible supporte une interface graphique fenêtrée.

4.6.1.4 Identification de la version-révision de la machine virtuelle

Les constantes suivantes permettent d'identifier la version-révision de la machine virtuelle technique :

- **CO_Version.**
Niveau de version d'*Up ! Virtual Technical Machine*.
- **CO_Revision.**
Niveau de révision d'*Up ! Virtual Technical Machine*.
- **CO_Correction.**
Niveau de correction d'*Up ! Virtual Technical Machine*.

4.6.1.5 Identification du niveau de code

Les définitions suivantes permettent de connaître le niveau de code :

- **MISEAUPOINT.**
Le code de mise au point est ajouté aux exécutable, ce qui les rend plus robustes au détriment du temps d'exécution.

4.6.2 Dimensionnement des tableaux

Aucune taille de tableau ne doit être définie en dur par l'emploi direct d'une constante. Pour chaque tableau, il existe une variable ou un champ entier qui est l'index explicite de remplissage du tableau.

Voici un exemple :

```
#define CO_TailleTableau 100

static TypUpsVmShort MonTableau[CO_TailleTableau];
static TypUpsVmShort IndexTableauMax;
```

Texte 35 – Exemple de dimensionnement des tableaux

Il existe trois constantes prédéfinies pour dimensionner la taille utile des tampons de chaîne de caractères :

- **CO_Tailleldf.**
Pour un identificateur tel le nom d'un type ou d'un module.
- **CO_TailleNomFichier.**
Pour un nom de fichier ou de répertoire.
- **CO_TailleBuffer.**
Pour un temporaire de calcul.

En employant toujours l'une de ces trois valeurs ou une combinaison de ces trois, cela permet de redimensionner le logiciel complètement en changeant uniquement la constante appropriée.

Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

TypUpsVmUnicode Identifiant[CO_TailleIdf+1];
TypUpsVmUnicode NomFichier[CO_TailleNomFichier+1];
TypUpsVmUnicode Chaine1[CO_TailleBuffer+1];
TypUpsVmUnicode Chaine2[2*CO_TailleIdf+1];

```

Texte 36 – Exemple de formatage d'une expression

M

Il faut toujours ajouter **+1** à la taille utile pour le caractère de fin de chaîne.

4.6.3 Formatage des expressions

Les expressions sont isolées entre parenthèses de la sorte à éviter toute ambiguïté d'évaluation et d'empêcher toute liberté d'évaluation au compilateur. Les calculs sont donc identiques quelle que soit la plate-forme. Ceci est particulièrement important pour :

- Les calculs booléens.
- Les calculs utilisant des nombres entiers et des nombres réels.

Si l'expression est trop longue pour être écrite sur une seule ligne, elle est coupée devant un foncteur et la ligne suivante est indentée d'un cran. De plus, à chaque mise entre parenthèse, le niveau d'indentation est augmenté d'un cran. Cela permet de repérer visuellement l'architecture de l'expression.

Voici un exemple :

```

if ( ( ExpressionGauche->Token==EX_OperateurBinaire )
      && ( ExpressionGauche->Selection.OperateurBinaire.Token==Token ) )
{
    ...
}

```

Texte 37 – Exemple de formatage d'une expression

Voici un contre-exemple :

```

if ( ExpressionGauche->Token==EX_OperateurBinaire && ExpressionGauche-
      >Selection.OperateurBinaire.Token==Token )
{
    ...
}

```

Texte 38 – Contre exemple de formatage d'une expression

M

Aucune affectation ne doit être réalisée dans une expression. Voici un contre-exemple :

```

TypUpsVmShort A;
TypUpsVmShort B;
...
A=B=1;

```

Texte 39 – Contre-exemple d'affectation dans une expression

4.6.4 Conversion de type

Les conversions de types sont systématiquement forcées pour éviter toute liberté au compilateur. Ceci est particulièrement important pour :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **Le changement de type pointé.**
Pour un héritage d'une interface.
- **La fiabilité dans les calculs.**
Quand des nombres entiers et réels sont mêlés.

Voici un exemple :

```
Resultat->Selection.Reel=(TypUpsVmDouble)Entier1+ObjetR->Selection.Reel
```

Texte 40 – Exemple de conversion de type – 1

Pour initialiser une valeur énumérée à zéro, il faut obligatoirement utiliser une conversion de type, sinon le compilateur renvoie une erreur.

```
return((enum EnuUpsVmOptionObjet)0);
```

Texte 41 – Exemple de conversion de type – 2

M Aucune conversion de pointeur ne doit être réalisée au sein d'une expression. Il faut utiliser une variable intermédiaire par souci de lisibilité. Voici un exemple

```
TypMonMdlB *F;
F=(TypMonMdlB *)C->D;
A=(TypUpsVmShort)(2*F->E+1);
```

Texte 42 – Exemple d'usage d'une expression sans conversion de type

Voici un contre-exemple :

```
A=(TypUpsVmShort)(2*((TypMonMdlB *)C->D)->E+1);
```

Texte 43 – Contre-exemple d'usage d'une conversion de type dans une expression

4.6.5 Fonctions standards

A l'exception de l'instruction *setjmp* et des opérateurs arithmétiques sur les nombres entiers et réels ou sur les booléens, aucune fonction standard du langage **C** n'est utilisée dans le cœur d'un module pour rendre les algorithmes indépendants de toutes plates-formes particulières.

Ces fonctions standards sont exposées ou redéfinies au sein des modules standards d'**Up ! Virtual Technical Machine**. Leurs noms sont généralement proches des fonctions standards du langage **C** et leurs prototypes sont adaptés de la manière suivante :

- **Usage de l'Unicode.**
Toutes les fonctions manipulant des chaînes de caractères sont encodées en **Unicode**, même si le système d'exploitation ne le supporte pas en théorie – par exemple **Windows Millenium**.
- **Harmonisation des codifications.**
Toutes les fonctions sont harmonisées pour utiliser les codifications standard d'**Up ! Virtual Technical Machine**.
- **Compatibilité multi-tâches.**
Toutes les fonctions sont encodées de la sorte à être employées dans un contexte multi-tâches, quel que soit le procédé utilisé pour la gestion des tâches et de la mémoire.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **Normes de sûreté.**
Toutes les fonctions sont encodées de la sorte à mettre en oeuvre les normes de sûreté de fonctionnement.

Les fonctions usuelles sont disponibles dans les modules suivants :

Ensemble de fonctions	Modules
Fonctions arithmétiques : <i>/</i> , <i>%</i> , etc.	Up ! Kernel.
Fonctions scientifiques : <i>abs</i> , <i>cos</i> , <i>tan</i> , etc.	Up ! Mathematical.
Fonctions de manipulation de chaîne de caractères : <i>strlen</i> , <i>strcmp</i> , <i>upper</i> , etc.	Up ! Natural Language Support.
Fonctions d'accès au réseau : <i>gethostbyaddr</i> , etc.	Up ! Network.
Fonctions d'accès aux ressources : <i>malloc</i> , <i>open</i> , etc.	Up ! System.

Texte 44 – Localisation des fonctions standards

Le détail des fonctions standards et les équivalences sont présentés en détail dans le chapitre **8** intitulé **Index des Application Program Interfaces** page 57.

4.6.6 Usage des pointeurs de fonction

Les pointeurs de fonction sont couramment utilisés étant donné l'architecture des modules et le nombre d'appels inter-modules. Pour être compilé sans ambiguïté, l'appel pointé, une fois désigné par le caractère **étoile ***, doit être entouré de parenthèses.

Voici un exemple :

```
A = (*MonPremierAppel) (B) ;
C = (*MonMdl->MonSecondAppel) (D) ;
```

Texte 45 – Exemple d'usage des pointeurs de fonction

Voici un contre-exemple :

```
A = *MonPremierAppel (B) ;
C = * (MonMdl->MonSecondAppel) (D) ;
```

Texte 46 – Contre-exemple d'usage d'une conversion de type dans une expression

4.6.7 Paramètres

Le codage d'une fonction ou d'une procédure respecte les conventions suivantes, mises en oeuvre dans l'ordre de leur énumération :

- **Initialisation des paramètres de sortie.**
A la valeur par défaut correspondant au type de données.
- **Vérification des paramètres obligatoires.**
Il s'agit de vérifier une contrainte de fonctionnement de la fonction ou de la procédure. En cas d'erreur détectée, l'exception **UpsKrn.ErreurInterne** est envoyée.
- **Initialisation des variables locales.**
A la valeur par défaut correspondant au type de données.
- **Traitements.**

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVm-000003-A Plan de programmation.doc		

- **Renvoi du résultat.**
Pour les fonctions.

D'une façon générale, il est préférable d'employer une fonction retournant un booléen traduisant son bon fonctionnement qu'une procédure aveugle en terme de compte-rendu. Par convention, **Vrai** traduit un bon fonctionnement.

Voici un exemple de contrôle des paramètres obligatoires :

```

/*****/
static TypUpsVmVoid UpsVmAPI FichierMth_8_4(TypUpsVmSession *Session,
    TypUpsVmAdresse *AdresseObjet, TypPrmUpsSys_12_8_4 *UpsPrm)
/* Procédure Ecrire(E:Nul Ou Entier); */
/*****/
{
    ...
if (!AdresseObjet->NumeroSegment || !UpsPrm->UpsVar1.NumeroSegment)
    {
        (*UpsSysIntTrtUpsKrn->EnvoyerExceptionStandard)(Session,
            (*UpsSysIntTrtUpsKrn->UpsException56_get)(Session,
                _T("UpsSys.Fichier.Ecrire"),
                _T("if (!AdresseObjet->NumeroSegment || !UpsPrm->UpsVar1.NumeroSegment)"),
                NULL, NULL, NULL);
        return;
    }
    ...
}

```

Texte 47 – Exemple de contrôle des paramètres obligatoires

4.7 Codifications

4.7.1 Booléens

Un booléen est représenté par le type **TypUpsVmShort**. La valeur 0 a pour sémantique **Faux** et la valeur 1 a pour sémantique **Vrai**.

Les autres valeurs sont interdites.

4.7.2 Caractère Unicode

Les 127 premiers caractères d'**Unicode** correspondent à la page de code **Us7-Ascii**. Les 255 premiers caractères d'**Unicode** correspondent à la page de code **Iso8859-P1**. Les codes **Unicode** de l'essentiel de ces caractères sont définis par un ensemble de constantes telle **UN_AMajuscule** pour **A**.

Cependant, les octets des caractères **Unicode** sont encodés dans la norme **Big endian** alors que le processeur n'accepte que des entiers à la forme soit **Big endian** ou soit **Little endian**.

Quand il s'agit d'employer un caractère sous sa forme de code **Unicode**, il est alors nécessaire d'effectuer éventuellement une conversion pour avoir le bon ordre des octets de l'entier. Pour simplifier ces manipulations, il faut utiliser la table **UpsNls.TableUnicode** comme cela :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

static TypUpsVmUnicode Bonjour[CO_TailleIdf+1];

Bonjour[0]=MonMdlIntDonUpsNls->TableUnicodes[UN_BMajuscule];
Bonjour[1]=MonMdlIntDonUpsNls->TableUnicodes[UN_O];
Bonjour[2]=MonMdlIntDonUpsNls->TableUnicodes[UN_N];
Bonjour[3]=MonMdlIntDonUpsNls->TableUnicodes[UN_J];
Bonjour[4]=MonMdlIntDonUpsNls->TableUnicodes[UN_O];
Bonjour[5]=MonMdlIntDonUpsNls->TableUnicodes[UN_U];
Bonjour[6]=MonMdlIntDonUpsNls->TableUnicodes[UN_R];
Bonjour[7]=0;
...
if (Bonjour[0]==MonMdlIntDonUpsNls->TableUnicodes[UN_BMajuscule])
{
...
}

```

Texte 48 – Exemple d'emploi des codifications Unicode

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

5 Architecture

5.1 Architecture d'un module

Un module est composé des bibliothèques suivantes :

- **Le cœur fonctionnel du module.**
Cette bibliothèque est obligatoire. Elle peut provenir de :
 - **Encodage natif manuel ou généré par *Up ! Compiler*.**
Elle contient le code du module écrit en **C--**. Il est indépendant de toute technologie, sauf du processeur cible choisi.
Ce module est soit une bibliothèque statique, soit une bibliothèque dynamique ou soit un programme exécutable.
Le fichier de cette bibliothèque ne comporte pas de suffixe particulier.
 - **Encodage pour le processeur *Up ! P32* généré par *Up ! Compiler*.**
Elle contient le code du module écrit en **C--**. Il est indépendant de toute technologie, même du processeur cible choisi. Il est interprété par **Up ! Engine**.
Ce module est soit une pseudo bibliothèque dynamique ou soit un pseudo programme exécutable.
Le fichier de cette bibliothèque ne comporte pas de suffixe particulier.
 - **Adaptateur client *Bapi*.**
Il est généré automatiquement par **Up ! Compiler** à partir du dictionnaire de **Sap**, pour créer un client d'appel de service à la norme **Bapi / Rfc**.
Ce module est une bibliothèque dynamique.
Le fichier de cette bibliothèque comporte le suffixe **_ca**.
 - **Adaptateur client *Corba*.**
Il est généré automatiquement par **Up ! Compiler** à partir soit du fichier **Interface Definition Language (IDL)** ou soit du contenu de l'**Interface Repository (IR)**, pour créer un client d'appel de service à la norme **Corba**.
Ce module est une bibliothèque dynamique.
Le fichier de cette bibliothèque comporte le suffixe **_cc**.
 - **Adaptateur client *DCom*.**
Il est généré automatiquement par **Up ! Compiler** à partir soit du fichier **Interface Definition Language (IDL)** ou soit du fichier **Type Library (TLB)**, pour créer un client d'appel de service à la norme **DCom**.
Ce module est une bibliothèque dynamique.
Le fichier de cette bibliothèque comporte le suffixe **_cd**.
 - **Adaptateur client *Java*.**
Il est généré automatiquement par **Up ! Compiler** à partir du fichier source **Java**, pour créer un client d'appel de service à la norme **Java Native Interface**.
Ce module est une bibliothèque dynamique.
Le fichier de cette bibliothèque comporte le suffixe **_cj**.
 - **Adaptateur client *Soap*.**
Il est généré automatiquement par **Up ! Compiler** à partir du fichier source **Xml**, pour créer un client d'appel de service à la norme **Soap / Http**.
Ce module est une bibliothèque dynamique.
Le fichier de cette bibliothèque comporte le suffixe **_ca**.
 - **Adaptateur client *Up ! Network*.**
Il est généré automatiquement par **Up ! Compiler** à partir du fichier source d'extension **upl**

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

pour créer un client d'appel de service à la norme **Up ! Network**.

Ce module est une bibliothèque dynamique.

Le fichier de cette bibliothèque comporte le suffixe **_cu**.

- **L'interface avec un logiciel de base.**
 Cette bibliothèque est obligatoire dès lors qu'un module utilise au moins une **Application Program Interface (API)** native.
 Ce module est une bibliothèque dynamique.
 Le fichier de cette bibliothèque comporte le suffixe **_nt**.
- **Le dictionnaire.**
 Cette bibliothèque est facultative. Elle mémorise la composition du module pour être interprétée par **Up ! Virtual Technical Machine**. Elle complète le dictionnaire d'**Up ! Object Management System**. Elle est générée automatiquement par **Up ! Compiler** à partir du fichier d'extension **upi**.
 Ce module est une bibliothèque dynamique.
 Le fichier de cette bibliothèque comporte le suffixe **_d**.
- **L'adaptateur serveur Ale.**
 Cette bibliothèque est facultative. Elle expose le contenu du module selon la norme d'appel de service **Ale / Rfc** de **SAP**. Elle est générée automatiquement par **Up ! Compiler** à partir du fichier d'extension **upi**.
 Ce module est une bibliothèque dynamique.
 Le fichier de cette bibliothèque comporte le suffixe **_sa**.
- **L'adaptateur serveur Corba.**
 Cette bibliothèque est facultative. Elle expose le contenu du module selon la norme d'appel de service **Corba**. Elle est générée automatiquement par **Up ! Compiler** à partir du fichier d'extension **upi**.
 Ce module est une bibliothèque dynamique.
 Le fichier de cette bibliothèque comporte le suffixe **_sc**.
- **L'adaptateur serveur DCom.**
 Cette bibliothèque est facultative. Elle expose le contenu du module selon la norme d'appel de service **DCom**. Elle est générée automatiquement par **Up ! Compiler** à partir du fichier d'extension **upi**.
 Ce module est une bibliothèque dynamique.
 Le fichier de cette bibliothèque comporte le suffixe **_sd**.
- **L'adaptateur serveur Java.**
 Cette bibliothèque est facultative. Elle expose le contenu du module selon la norme d'appel de service **Java Native Interface**. Elle est générée automatiquement par **Up ! Compiler** à partir du fichier d'extension **upi**.
 Ce module est une bibliothèque dynamique.
 Le fichier de cette bibliothèque comporte le suffixe **_sj**.
- **L'adaptateur serveur Soap.**
 Cette bibliothèque est facultative. Elle expose le contenu du module selon la norme d'appel de service **Soap / Http**. Elle est générée automatiquement par **Up ! Compiler** à partir du fichier d'extension **upi**.
 Ce module est une bibliothèque dynamique.
 Le fichier de cette bibliothèque comporte le suffixe **_so**.
- **L'adaptateur serveur Up ! Network.**
 Cette bibliothèque est facultative. Elle expose le contenu du module selon la norme d'appel de service **Up ! Network**. Elle est générée automatiquement par **Up ! Compiler** à partir du fichier d'extension **upi**.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Ce module est une bibliothèque dynamique.
Le fichier de cette bibliothèque comporte le suffixe **_su**.

Le code du module est indépendant du contexte culturel d'exécution, en particulier de la langue choisie par l'utilisateur. Le module est donc complété par les ressources linguistiques mémorisées dans des fichiers d'extension **nls**. Il y a un fichier par bibliothèque.

Voici un schéma résumant l'organisation logique d'un module :

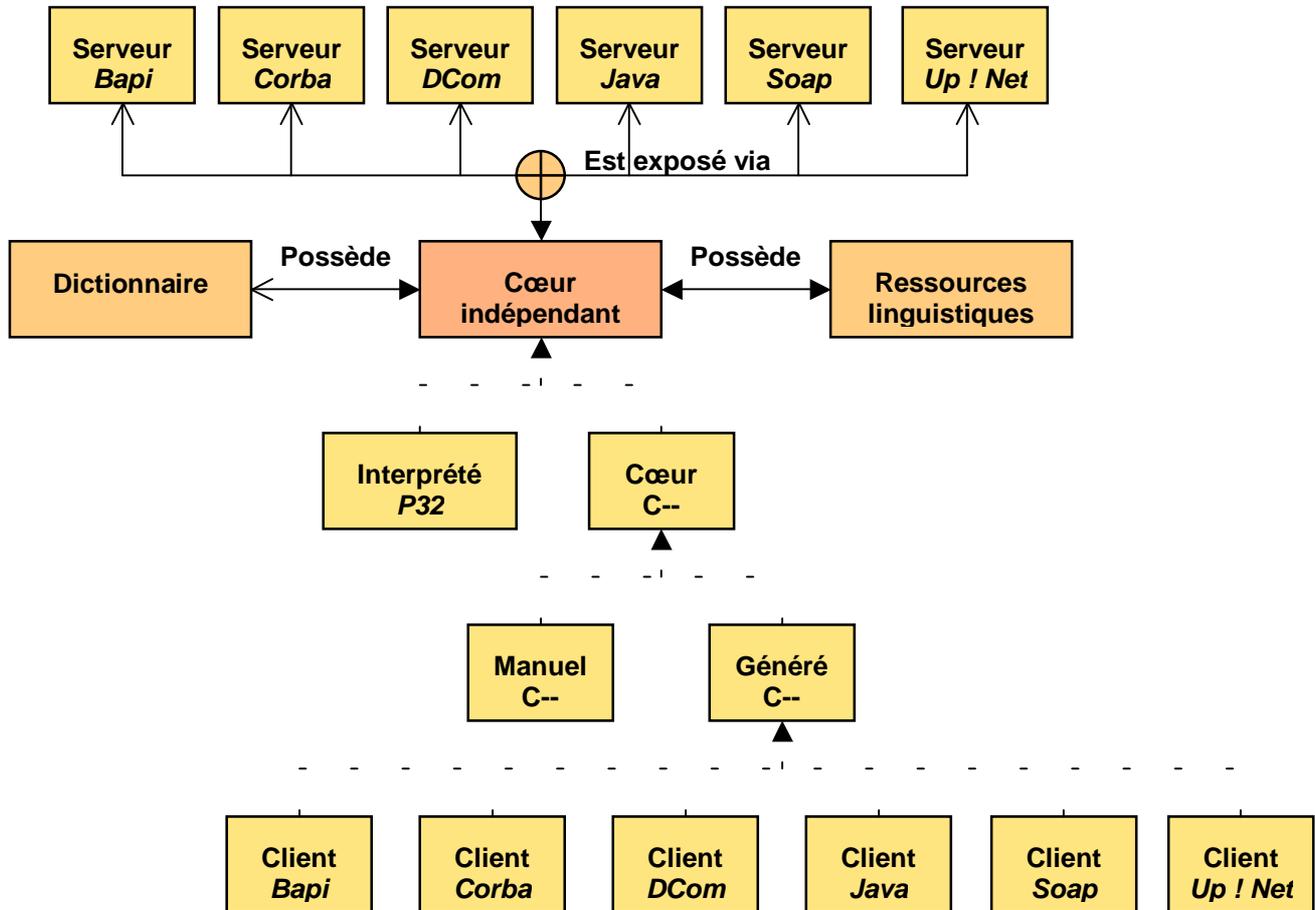


Figure 49 – Organisation logique d'un module

Un module est une unité logique de livraison. Toutes les bibliothèques du module porte le même niveau de numéro de version-révision et elles sont toutes livrées en cas de changement de numéro version-révision.

En cas de changement de numéro de correction, seules quelques bibliothèques peuvent être livrées.

5.2 Architecture d'une bibliothèque

Il existe deux architectures de bibliothèque :

- **L'architecture simplifiée.**
Elle est utilisée uniquement par la bibliothèque native d'interface avec les logiciels de base.
- **L'architecture complète.**
Elle est utilisée par toutes les autres bibliothèques.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

5.2.1 Composants d'une bibliothèque

Une bibliothèque est constituée de plusieurs composants. Les composants portent généralement le nom de la bibliothèque avec une séquence. Voici leur nature :

- **Composant d'interface avec *Up ! Module*.**
Ce composant comporte des fonctions spécifiques utilisées par *Up ! Module* pour piloter la bibliothèque et l'informer des changements d'instance d'exécution. Il est obligatoire. Par convention, il s'agit toujours du composant de numéro 0.
- **Composants cœur.**
Ces composants comportent le code du cœur fonctionnel et technique de la bibliothèque.
- **Composant d'interface avec le dictionnaire.**
Ce composant exécute les ordres d'interprétation provenant de la bibliothèque du dictionnaire du module. Il est obligatoire sauf pour l'architecture simplifiée. Par convention, il s'agit toujours du composant de numéro 99.

Voici un schéma résumant l'organisation logique d'une bibliothèque :

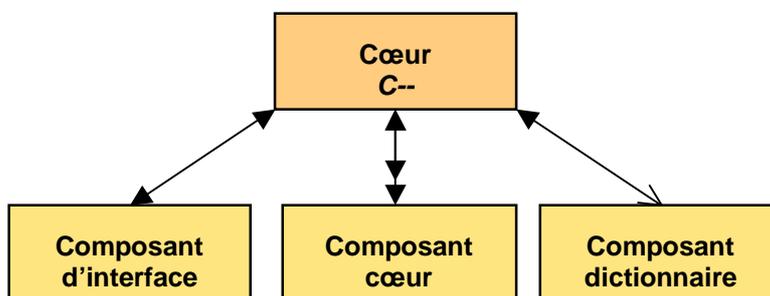


Figure 50 – Organisation logique d'une bibliothèque

5.2.2 Fichiers sources

5.2.2.1 Fichiers sources d'une bibliothèque

Les fichiers source d'une bibliothèque, autres que ceux de ses composants, sont les suivants :

- **En-tête des définitions publiques.**
Ce fichier porte le nom de la bibliothèque et a pour extension **e**.
Il regroupe l'ensemble des définitions publiques que la bibliothèque met à disposition des autres modules dans le cadre d'une relation client-fournisseur. Il y a notamment l'interface des données et l'interface des traitements.
- **En-tête des définitions privées.**
Ce fichier porte le nom de la bibliothèque et a pour extension **h**.
Il regroupe les inclusions :
 - Des fichiers d'inclusion des **Application Program Interfaces** des logiciels de base.
 - Des fichiers d'en-tête des définitions publiques des autres modules.
 - Du fichier d'en-tête des définitions publiques du module.
- **Fichier de ressources linguistiques.**
Ce fichier porte le nom de la bibliothèque et a pour extension **nls**.
Il regroupe l'ensemble des traductions des termes, des expressions et des phrases employées par la bibliothèque dans une langue donnée.

Voici un schéma résumant l'organisation physique d'une bibliothèque :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

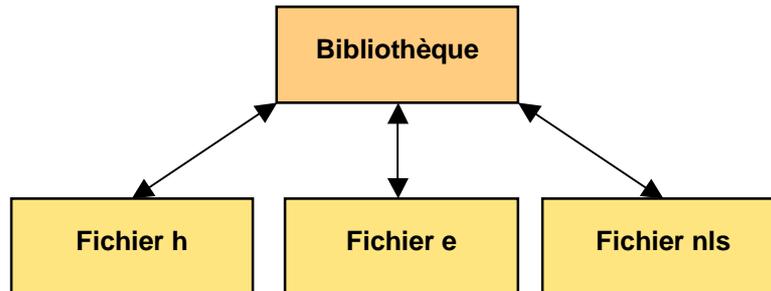


Figure 51 – Organisation physique d'une bibliothèque

5.2.2.2 Fichiers sources d'un composant

Les fichiers source d'un composant sont les suivants :

- **En-tête des définitions protégées.**
Ce fichier porte le nom du composant et a pour extension **e**.
Il regroupe l'ensemble des définitions protégées que le composant met à disposition des autres composants du module dans le cadre d'une relation client-fournisseur. Il y a notamment les définitions publiques qui sont exposées par le composant d'interface avec **Up ! Module**.
- **En-tête des définitions privées.**
Ce fichier porte le nom du composant et a pour extension **h**.
Il regroupe les inclusions :
 - Des fichiers des **Application Program Interfaces** des logiciels de base.
 - Du fichier d'en-tête des définitions publiques du module.
 - Des fichiers d'en-tête des définitions protégées des autres composants du module.
En particulier, le composant d'interface avec **Up ! Module** et le composant d'interface avec le dictionnaire.
- **Fichier de code.**
Ce fichier porte le nom de la bibliothèque et a pour extension **cpp**.
Il regroupe l'ensemble du code du composant écrit en **C--**.

Voici un schéma résumant l'organisation physique d'un composant :

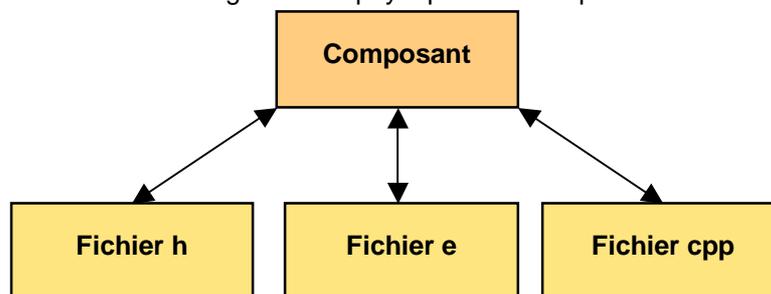


Figure 52 – Organisation physique d'un composant

5.2.3 Interface avec un logiciel de base

Isoler les **Application Program Interfaces** d'un logiciel de base dans une bibliothèque dédiée à l'interfaçage avec le logiciel de base est une nécessité pour les raisons suivantes :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **Convertir les types de données.**
Le cœur d'un module n'utilise que des informations dont les types sont ceux d'**Up ! Virtual Technical Machine**.
- **Aligner les champs.**
La convention d'alignement des champs d'une structure ou d'une union est fixée à quatre octets pour **Up ! Virtual Technical Machine** alors qu'elle est libre pour chaque système d'exploitation. Par exemple, l'alignement par défaut est de huit octets sous **Microsoft Windows**.
- **Gérer Unicode.**
La convention de stockage des chaînes de caractère est d'utiliser la norme **Unicode 2.0** pour **Up ! Virtual Technical Machine** alors qu'elle est libre pour chaque système d'exploitation.
- **Gérer le multitâche.**
La convention d'exécution d'**Up ! Virtual Technical Machine** est un environnement multitâche alors qu'elle est libre pour chaque système d'exploitation.
- **Simplifier les APIs.**
Nombreuses sont les **Application Program Interface (API)** qui sont plus puissantes que nécessaires, soit en terme de paramétrage soit en terme de codification. Cela permet de se restreindre au juste suffisant au regard de la visibilité d'emploi à venir.
- **Faciliter la maintenance.**
En cas de livraison d'un paquet de correction de la part de l'éditeur du logiciel de base, il suffit uniquement de recompiler cette bibliothèque et de la transmettre à nos clients.

M Il ne faut pas dupliquer les **Application Program Interfaces (API)** natives d'un module à l'autre.

La bonne pratique est d'avoir un module par thème – système d'exploitation, bases de données, interfaces homme-machine, réseau, etc. – et de les exposer sous forme de fonction de plus haut niveau au travers des interfaces de traitements de ce module.

Voici un exemple :

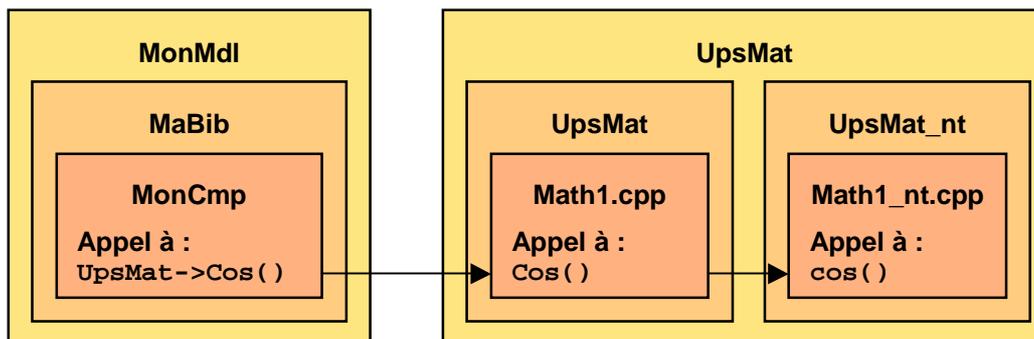


Figure 53 – Exemple d'appel inter-module pour utiliser une API native

5.3 Architecture d'un composant

5.3.1 Programmation orientée objet

Même si le code est écrit en **C-**, la programmation d'**Up ! Virtual Technical Machine** est orientée objet, dans un formalisme proche de celle de **C++**. En fait, **C++** n'a pu être retenu du fait du fonctionnement interne à **Up ! Module** et **Up ! Object Management System**.

Dans **Up ! Virtual Technical Machine**, tout est un objet – un programme, un module, un type, etc. –, avec une intime association entre les données et les traitements.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Il existe deux sortes d'objets :

- **Les objets cachés.**
Ils ne sont pas exploitables au travers d'**Up ! 5GL**, aussi la norme de codification des propriétés et des méthodes est relativement libre.
- **Les objets exposés.**
Ils sont exploitables au travers d'**Up ! 5GL**, aussi la norme de codification des propriétés et des méthodes est imposée de la sorte que :
 - **Up ! Compiler** puisse générer le code natif.
 - **Up ! Engine** puisse faire appel à ces définitions lors de l'interprétation du code.

Cette norme est présentée dans les documentations d'**Up ! Kernel** et d'**Up ! Management System**.

Il arrive que des objets exposés encapsulent des objets cachés. C'est le cas par exemple pour le type **Module**.

La définition d'un objet repose sur :

- **Une interface des traitements.**
Elle regroupe toutes les méthodes explicites de l'objet ainsi que les méthodes implicites provenant de l'encapsulation des données.
- **Une interface des données.**
Elle regroupe toutes les informations explicites et implicites de l'objet.

5.3.1.1 Interface des traitements

L'interface de traitements est partagée par tous les objets du même type et elle est publiée en différentes versions-révisions. Cela signifie qu'il existe physiquement plusieurs interfaces de traitements dont la liste des méthodes varie en fonction de l'évolution des versions-révisions.

L'exception est la bibliothèque d'interface native qui ne comporte que l'interface des traitements dans la dernière version-révision.

La structure d'une interface de traitements peut être imposée par un gestionnaire, par exemple **Up ! Module** le gestionnaire de modules. Le fournisseur, alimentant le gestionnaire selon la norme imposée, peut étendre cette définition en ajoutant à la suite ses propres méthodes. C'est la raison pour laquelle il existe des en-têtes de table de méthodes.

Pour les objets cachés, il y a des méthodes publiques mais cachées, uniquement utilisables par **Up ! Virtual Technical Machine**.

Cette interface de traitements est appelée **Virtual Table (VTBL)** en **C++**.

5.3.1.2 Interface des données

L'interface de données est dupliquée pour les objets du même type et elle est publiée uniquement dans la dernière versions-révisions, ce qui n'est pas gênant puisque l'accès aux propriétés de l'objet est réalisé au travers de méthodes implicites.

Pour les objets encapsulant une ressource, son identifiant– le **handle** du fichier par exemple – fait partie des informations mémorisées par l'interface de données. Pour les objets exposés encapsulant des objets cachés, l'interface des données est généralement réduite à sa plus simple expression, à savoir l'en-tête standard des données.

Concernant le cycle de vie des objets, la convention est la suivante :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **Type caché.**
Le cycle de vie et le contrôle d'accès aux objets doivent être gérés par le module qui définit le type dans son fichier interface d'extension **e**.
- **Type exposé.**
Le cycle de vie et le contrôle d'accès aux objets doivent être gérés par **Up ! Object Management System**. Le module ne fait que déclarer à **Up ! Object Management System** le type défini dans son fichier interface d'extension **e**.

5.3.1.3 Compatibilité avec le C++

Afin de pouvoir être inclus plusieurs fois sans engendrer des erreurs pour des définitions multiples, le contenu des fichiers d'en-têtes de définition doit être protégé par des instructions **#if / #endif**.

De même, comme les définitions sont en **C--**, il faut les déclarer comme tel pour que le compilateur **C++** puisse les assigner à ce langage.

Voici un exemple :

```

/*****/
#if !defined(PasseModuleUpsSysNat)
/*****/
/*****/
#if defined(__cplusplus)
/*****/
extern "C"
{
/****/
#endif
/****/
...

/*****/
#if defined(__cplusplus)
/*****/
}
/****/
#endif
/****/
/****/
#endif
/****/

```

Texte 54 – Exemple de protection contre l'inclusion multiple et de déclaration du langage cible

5.3.2 Architecture simplifiée

M

Une bibliothèque en architecture simplifiée ne peut pas utiliser les fonctionnalités d'une autre bibliothèque. Il doit être autonome dans son fonctionnement.

La conversion de code d'**Unicode 2.0** à la page de code du logiciel de base doit être réalisée par l'appelant.

5.3.2.1 En-tête des définitions publiques de la bibliothèque

Ce fichier contient :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- La définition des types des paramètres complexes.
Afin de respecter les règles énoncées dans le paragraphe **5.2.3** intitulé **Interface avec un logiciel de base** page 30.

Voici un exemple de structure de paramètre complexe :

```

/*****/
typedef struct typupsysnatlocaltime
/* Objet: Parametre de localtime(). */
/*****/
{
TypUpsVmShort tm_sec;
TypUpsVmShort tm_min;
TypUpsVmShort tm_hour;
TypUpsVmShort tm_mday;
TypUpsVmShort tm_mon;
TypUpsVmShort tm_year;
TypUpsVmShort tm_wday;
TypUpsVmShort tm_yday;
TypUpsVmShort tm_isdst;
} TypUpsSysNatLocalTime;

```

Texte 55 – Exemple de paramètre complexe – Architecture simplifiée

- L'interface de traitements de la bibliothèque.
Il s'agit d'une structure au nom imposé défini par le nom du module suivi de **Nat** suivi de **Traitements**.
Le contenu est la liste des appels.

Voici un exemple d'interface de traitements :

```

/*****/
typedef struct typupssysnattraitements
/* Objet: Interface des traitements de Ups Sys Nat. */
/*****/
{
TypUpsVmVoid UpsVmAPI>(*Malloc)(TypUpsVmLong Taille);
/* Objet : malloc(). */
TypUpsVmVoid UpsVmAPI>(*Realloc)(TypUpsVmPointeurDonnees Adresse,
TypUpsVmLong Taille);
/* Objet : realloc(). */
TypUpsVmVoid UpsVmAPI(*Free)(TypUpsVmPointeurDonnees Adresse);
/* Objet : free(). */
...
} *TypUpsSysNatTraitements;

```

Texte 56 – Exemple d'interface de traitements – Architecture simplifiée

- L'exportation de la fonction de démarrage.
Il s'agit d'une fonction au nom imposé défini par le nom du module suivi de **Nat** suivi de **DemarrerModule**.
Son contenu est imposé. Il est présenté dans les documentations d'**Up ! Module**.

Voici un exemple d'exportation de la fonction de démarrage :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVm-000003-A Plan de programmation.doc		

```
extern DllExport TypUpsSysNatTraitements UpsVmAPI
UpsSysNatDemarrerModule(TypUpsVmVoid);
/* Objet : Retourne la description de Ups Sys Nat. */
```

Texte 57 – Exemple d'exportation de la fonction de démarrage – Architecture simplifiée

5.3.2.2 En-tête des définitions protégées des composants

Ce fichier contient la liste des prototypes des **Application Program Interfaces (API)** encapsulées. Voici un exemple :

```
extern TypUpsVmVoid UpsVmAPI *UpsSysNatMalloc(TypUpsVmLong Taille);
/* Objet : malloc(). */
extern TypUpsVmVoid UpsVmAPI *UpsSysNatRealloc(TypUpsVmPointeurDonnees
Adresse, TypUpsVmLong Taille);
/* Objet : realloc(). */
extern TypUpsVmVoid UpsVmAPI UpsSysNatFree(TypUpsVmPointeurDonnees
Adresse);
/* Objet : free(). */
```

Texte 58 – Exemple de prototypes des API encapsulées – Architecture simplifiée

5.3.2.3 Fichier de code du composant d'interface avec Up! Module

Ce fichier contient le code de la fonction de démarrage au nom imposé défini par le nom du module suivi de **Nat** suivi de **DemarrerModule**. Son objet est de remplir la table des méthodes et de retourner son adresse.

Voici un exemple :

```
static struct typupssysnattraitements Traitements;
/* Interface de traitement. */

/*****
DllExport TypUpsSysNatTraitements UpsVmAPI
UpsSysNatDemarrerModule(TypUpsVmVoid)
/* Objet : Retourne la description de Ups Sys Nat. */
*****/
{
Traitements.Malloc=UpsSysNatMalloc;
Traitements.Realloc=UpsSysNatRealloc;
Traitements.Free=UpsSysNatFree;
...
return(&Traitements);
}
```

Texte 59 – Exemple d'un composant d'interface avec Up! Module – Architecture simplifiée

5.3.2.4 Fichier de code du composant coeur

Ce fichier contient le code du composant cœur réalisant l'encapsulation de chaque **Application Program Interface (API)** du logiciel de base utilisée.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

a La **convention d'appel** est que l'appelé réalise la conversion en début d'appel pour les paramètres d'entrée et en fin d'appel pour les paramètres de sortie en utilisant un temporaire dans le type natif pour chaque conversion non triviale.

Voici un exemple :

```

/*****/
TypUpsVmChar UpsVmAPI *UpsSysNatFcvt(TypUpsVmDouble Double,
    TypUpsVmShort TaillePartieReelle, TypUpsVmShort *Decimale,
    TypUpsVmShort *Signe)
/* Objet : Fcvt(). */
/*****/
{
TypUpsVmChar *Resultat;
    /* Variable de travail. */
int Decimale2;
    /* Pour la conversion de type. */
int Signe2;
    /* Pour la conversion de type. */

Resultat=(TypUpsVmChar *)fcvt(Double, TaillePartieReelle, &Decimale2,
    &Signe2);
if (Decimale)
    *Decimale=(TypUpsVmShort)Decimale2;
if (Signe)
    *Signe=(TypUpsVmShort)Signe2;
return(Resultat);
}

```

Texte 60 – Exemple d'encapsulation d'une API – Architecture simplifiée

5.3.3 Architecture complète

5.3.3.1 En-tête des définitions publiques de la bibliothèque

Ce fichier contient :

- La définition des constantes.
Au travers de l'instruction **#define** du pré-processeur. Il y a également les macros-fonctions.
- La définition des énumérés.
Au travers de l'instruction **enum** du langage **C--**.
- La définition des interfaces des données des types.
Conformément à ce qui est défini dans les documentations d'**Up ! Object Management System**.
- La définition des interfaces des traitements des types.
Conformément à ce qui est défini dans les documentations d'**Up ! Object Management System**.
- La définition de l'interface des données de la bibliothèque.
Il s'agit d'une structure au nom imposé défini par le nom du module, suivi de **Données**, suivi par le suffixe de la bibliothèque.
Cette interface étend l'en-tête obligatoire **EnteteDonneesModule**.

Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

/***** /
typedef struct typupssysdonnees
/* Objet: Interface des donnees de Ups Sys. */
/***** /
{
TypUpsVmEnteteDonneesModule EnteteDonneesModule;
/* Entete generique du module. */

TypUpsVmUnicode UpsUser[CO_TailleIdf+1];
/* Valeur de UPS_USER. */
TypUpsVmUnicode UpsProjet[CO_TailleIdf+1];
/* Valeur de UPS_Projet. */
TypUpsVmUnicode NomProgramme[CO_TailleIdf+1];
/* Nom du programme. */
...
} *TypUpsSysDonnees;

```

Texte 61 – Exemple d’interface de données d’une bibliothèque – Architecture complète

M

L’interface de données d’une bibliothèque contient directement les rares informations stables. En effet, en cas de changement dans sa définition, il faudrait recompiler entièrement le logiciel et diffuser la nouvelle version-révision à tous les clients.

- **La définition des interfaces des traitements de la bibliothèque.**
Il s’agit d’une structure au nom imposé défini par le nom du module, suivi de **Traitements**, suivi par le suffixe de la bibliothèque, suivi du numéro de version-révision-correction. Il y a une interface par version-révision-correction du module. Ces interfaces étendent chacune l’en-tête obligatoire **EnteteMethodesModule**.

Voici un exemple :

```

/***** /
typedef struct typupssystraitements_1_0_0
/* Objet: Interface des traitements de Ups Sys. */
/***** /
{
TypUpsVmEnteteMethodesModule EnteteMethodesModule;
/* Methodes generiques au module. */

TypUpsVmVoid UpsVmAPI (*Trt_1_1)(TypUpsVmSession *Session,
TypPrmUpsSys_1_1 *UpsPrm);
/* Objet : Procedure CreerFichier(NomFichier:Nul Ou Caractere). */
...
} *TypUpsSysTraitements_1_0_0;

```

Texte 62 – Exemple d’interface de traitements d’une bibliothèque – Architecture complète

- **L’alias vers l’interface des traitements de la bibliothèque courante.**
Il ne porte pas le numéro de version-révision-correction de la bibliothèque.

Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

/*-----*/
/* La version courante est la 1.0.0. */
/*-----*/
typedef TypUpsSysTraitements_1_0_0 TypUpsSysTraitements;

```

Texte 63 – Exemple d’alias sur l’interface de traitements d’une bibliothèque – Architecture complète

5.3.3.2 En-tête des définitions protégées des composants

Ce fichier contient la liste des prototypes des fonctions exportées. Voici un exemple :

```

extern TypUpsVmUnicode UpsVmAPI *UpsSysCalculerRepertoireIni(
    TypUpsVmUnicode *Reponse, TypUpsVmLong TailleReponse);
/* Objet : Calcule le repertoire dans lequel sont ranges les fichiers
ini de l'utilisateur. */
extern TypUpsVmShort UpsVmAPI UpsSysPartagerUpsIni(TypUpsVmVoid);
/* Objet : Recherche si ups.ini est partage. */

```

Texte 64 – Exemple de prototypes des fonctions exportées – Architecture complète

5.3.3.3 Fichier de code du composant d’interface avec Up ! Module

Ce fichier contient :

- **Les définitions fournies par le gestionnaire de module *Up ! Module*.**
Les pointeurs vers les interfaces de données et de traitements de la bibliothèque principale des modules importés pour la version-révision demandée.
- **Les liaisons vers les autres bibliothèques du module.**
Vers sa bibliothèque dictionnaire et vers ses bibliothèques d’adaptateur serveur.
- **Les définitions à fournir obligatoirement à *Up ! Module*.**
La gestion de la persistance, la gestion des interfaces, la gestion des interfaces, l’initialisation du module, la terminaison du module, le démarrage du module.

Ces fonctions sont détaillées dans la documentation d’*Up ! Module* et d’*Up ! Object Management System*.

Voici un exemple :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

```

TypUpsModTraitements UpsSysIntTrtUpsMod=NULL;
/* Interface de traitements de Ups Mod. */
TypUpsModDonnees UpsSysIntDonUpsMod=NULL;
/* Interface de traitements de Ups Mod. */
...

/*****
static TypUpsVmVoid UpsVmAPI *LireEnteteDonneesModule(void)
/* Objet : Lit l'entete de l'interface de donnees du module. */
*****/
{
...
}

/*****
static TypUpsVmEnteteMethodesModule UpsVmAPI *LireInterfaceModule(
TypUpsVmShort Version, TypUpsVmShort Revision, TypUpsVmShort Correction)
/* Objet : Lit l'interface des traitements du module et transmet */
/* la au serveur de modules. */
*****/
{
...
}

/*****
static TypUpsVmShort UpsVmAPI InitialiserModule(TypUpsModTraitements IntTrtUpsMod,
TypUpsModDonnees IntDonUpsMod)
/* Objet : Initialise le module Ups Sys. */
*****/
{
...
}

/*****
static TypUpsVmVoid UpsVmAPI TerminerModule(TypUpsVmVoid)
/* Objet : Termine le module Ups Sys. */
*****/
{
...
}

/*****
static TypUpsVmVoid UpsVmAPI ChangerInstance(TypUpsVmChar *NomModule,
TypUpsVmEnteteDonneesModule *Interface)
/* Objet : Previens le module d'un changement d'instance. */
*****/
{
...
}

/*****
DllExport TypUpsModDemarrer UpsVmAPI UpsSysDemarrerModule(TypUpsVmShort Numero)
/* Objet : Retourne la description de Ups Sys. */
*****/
{
...
}

```

Texte 65 – Exemple d'un composant d'interface avec Up ! Module – Architecture complète

5.3.3.4 Fichier de code du composant coeur

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Ce fichier contient le code du cœur de composants ainsi que trois fonctions obligatoires :

- **L'initialisation du composant.**
Son nom est libre.
- **La terminaison du composant.**
Son nom est libre.
- **Le démarrage du composant.**
Son nom est libre.

Voici un exemple :

```

static TypUpsSysVarGlo3 UpsSysVarGlo3=NULL;
/* Variables globales statiques. */

...

/*****
TypUpsVmShort UpsVmAPI UpsSysInitGroupe(TypUpsVmVoid)
/* Objet : Initialisation du composant. */
/*****
{
...
return(1);
}

/*****
TypUpsVmVoid UpsVmAPI UpsSysTerminerGroupe(TypUpsVmVoid)
/* Objet : Terminaison du composant. */
/*****
{
...
}

/*****
TypUpsVmVoid UpsVmAPI UpsSysDemarrerGroupe(TypUpsModSegmentsDonnees
*SegmentsDonnees)
/* Objet : Demarrage du composant. */
/*****
{
...
}

```

Texte 66 – Exemple d'un composant coeur – Architecture complète

5.3.3.5 Fichier de code du composant d'interface avec dictionnaire

Ce fichier contient les fonctions d'interface avec la bibliothèque du dictionnaire. Elles sont les suivantes :

- **Sur les énumérés.**
Enumération des énumérés et de leurs valeurs.
- **Sur les entrepôts.**
Enumération des entrepôts.
- **Sur les types.**
Enumération des types.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **Sur les variables.**
Enumération des variables et opération sur celles-ci.
 - **Sur les exceptions.**
Enumération des exceptions.
 - **Sur les appels.**
Enumération des appels.
- Ces fonctions sont détaillées dans la documentation d'*Up ! Kernel*.

5.4 Interface avec la machine

& A chaque fois qu'un module nécessite des **Application Program Interfaces (API)** pour exploiter les ressources d'un logiciel de base, une **interface avec la machine** est utilisée. Il s'agit d'une interface de traitements ne manipulant que des concepts de haut niveau encodée indépendamment de la plate-forme cible i.e. n'utilisant que les types de données du langage **C--**.

Le cœur du module n'emploie que ces **Applications Program Interfaces (API)** qui sont renseignées par un composant spécialisé par famille de plate-forme. Ce dernier porte le nom de l'abrégié du module concaténé à l'abrégié de la famille de la plate-forme. Par exemple **netmac**, **net390**, **net400**, **netunx** et **netw32** pour l'adaptateur d'*Up ! Network* respectivement pour **Macintosh**, **Os 390**, **Os 400**, **Unix** et **Windows**.

Ces composants mettent en oeuvre chacune de ces fonctions, en faisant appel aux **Applications Program Interfaces (API)** natives du logiciel de base qui sont encapsulées dans la bibliothèque native. Pour cela, une version simplifiée des ces **APIs** est utilisée, le temps de faire les conversions de type et de basculer d'**Unicode 2.0** à la page de code système.

Voici un schéma résumant cette architecture :

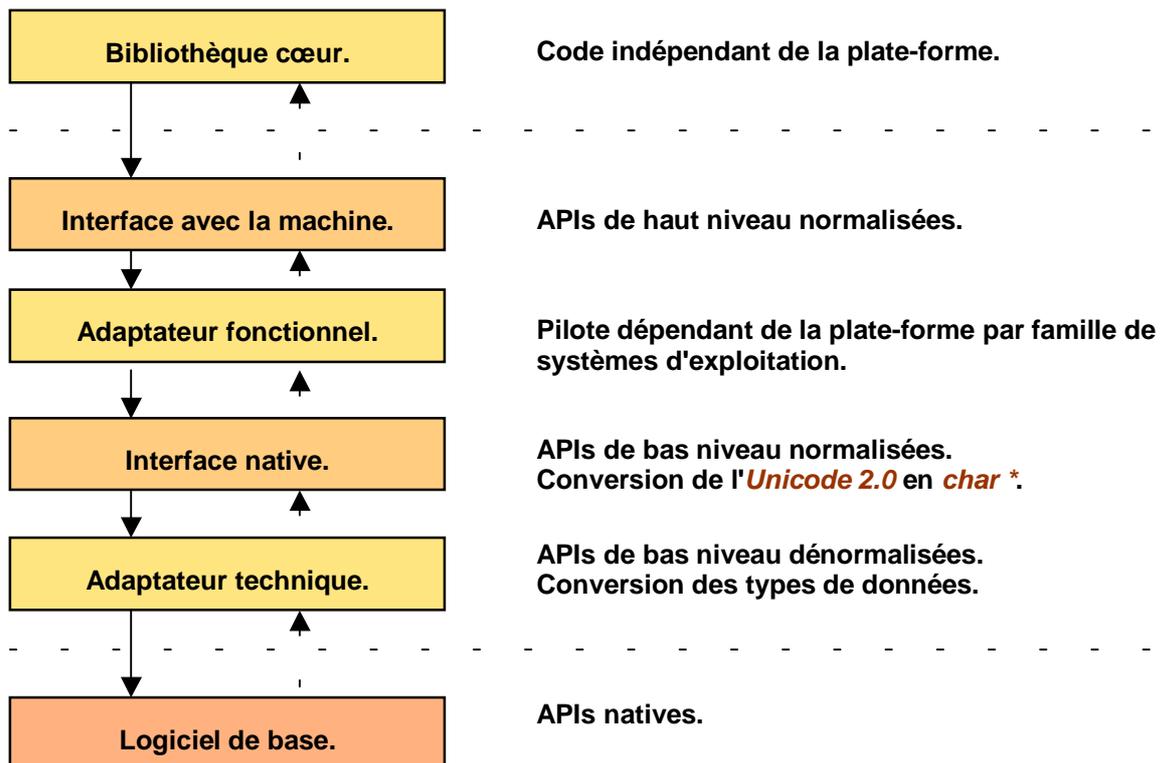


Diagramme 67 – Exemple d'usage d'une interface avec la machine

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVm-000003-A Plan de programmation.doc		

Voici un exemple d'une interface machine :

```

/*****
typedef struct typupsmodmachineinterface
/* Objet : Interface avec la machine pour Up ! Module. */
/*****
{
TypUpsVmVoid UpsVmAPI (*AjouterExtensionBibliotheque)(TypUpsVmChar
*NomBibliotheque);
/* Objet : Ajoute l'extension du nom de la bibliotheque dynamique. */
TypUpsVmVoid UpsVmAPI (*CalculerBibliothequeNative)(TypUpsVmChar
*RepertoireDeBase, TypUpsVmChar *NomModule, TypUpsVmChar *Bibliotheque);
/* Objet : Calcule le nom physique de la bibliotheque native. */
TypUpsVmVoid UpsVmAPI (*CalculerBibliotheque)(TypUpsVmChar *NomModule,
TypUpsVmChar *Bibliotheque);
/* Objet : Calcule le nom physique de la bibliotheque. */
TypUpsVmPointeurDonnees UpsVmAPI (*ChargerBibliotheque)(TypUpsVmChar
*NomBibliotheque);
/* Objet : Charge une bibliotheque dynamique. */
TypUpsVmVoid UpsVmAPI (*DechargerBibliotheque)(TypUpsVmPointeurDonnees
HandleBibliotheque);
/* Objet : Decharge une bibliotheque dynamique. */
TypUpsVmPointeurTraitements UpsVmAPI (*LireAPI)(TypUpsVmPointeurDonnees
HandleBibliotheque, TypUpsVmChar *NomApi);
/* Objet : Retrouve une adresse de fonction dans l'instance d'apres son
nom. */
TypUpsVmShort UpsVmAPI (*IlexisteFichier)(TypUpsVmChar *NomFichier);
/* Objet : Teste s'il existe un fichier. */
TypUpsVmPointeurDonnees UpsVmAPI (*AllouerMemoire)(TypUpsVmLong Taille);
/* Objet : Alloue de la memoire privatee au processus. */
TypUpsVmPointeurDonnees UpsVmAPI (*ReallouerMemoire)(
TypUpsVmPointeurDonnees Adresse, TypUpsVmLong Taille);
/* Objet : Realloue de la memoire privatee au processus. */
TypUpsVmVoid UpsVmAPI (*LibererMemoire)(TypUpsVmPointeurDonnees Adresse);
/* Objet : Effectue une desallocation. */
TypUpsVmPointeurDonnees UpsVmAPI (*AllouerMemoirePartagee)(TypUpsVmChar
*NomMemoire, TypUpsVmLong Taille,
TypUpsVmLong *NumeroMemoire, TypUpsVmLong Cle);
/* Objet : Cree ou retrouve une memoire partagee. */
TypUpsVmVoid UpsVmAPI (*LibererMemoirePartagee)(TypUpsVmPointeurDonnees
Adresse, TypUpsVmLong NumeroMemoire);
/* Objet : Libere une memoire partagee. */
TypUpsVmVoid UpsVmAPI (*EcrireEcran)(TypUpsVmChar *Message);
/* Objet : Ecrit un message a la console. */
TypUpsVmVoid UpsVmAPI (*ArreterProgramme)(TypUpsVmShort CodeRetour);
/* Objet : Arrete proprement le programme. */
TypUpsVmVoid UpsVmAPI (*GererSignaux)(TypUpsVmVoid);
/* Objet : Genere les signaux recus par le thread. */
TypUpsVmChar UpsVmAPI (*LireEnvironnement)(TypUpsVmChar *NomVariable);
/* Objet : Lit une variable environnement. */
} *TypUpsModMachineInterface;

```

Texte 68 – Exemple d'interface avec la machine

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

a

Afin de ne pas dupliquer les interfaces machines, le module qui en définit une publie sa mise en oeuvre au travers d'une **API** dans son interface de traitements. Ceci permet de centraliser les accès aux logiciels de base, ce qui facilite la maintenance.

Voici un exemple :

```

TypUpsModMachineInterface UpsVmAPI (*LireMachineInterface)(TypUpsVmVoid);
/* Objet : Retrouve l'interface machine pour Ups Module. */

```

Texte 69 – Exemple de publication d'une interface avec la machine

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

6 Organisation des versions-révisions

Une version-révision d'**Up ! Application System** comporte de nombreux éléments qu'il est nécessaire de conserver cohérents. Il y a notamment :

- **Les documentations.**
Les documentations externes publiées sur le site **Internet** et les documentations techniques internes.
- **Les fichiers sources.**
En **C--**, en **C++**, en **Java** ou en **Up ! 5GL**.
- **Les fichiers de compilation.**
Pour produire le code exécutable des plates-formes cibles.
- **Les fichiers de test.**
Pour vérifier le bon fonctionnement d'**Up ! Application System**.
- **Les fichiers d'exemples.**
Pour illustrer l'emploi d'**Up ! Application System**.

L'organisation retenue est la suivante :

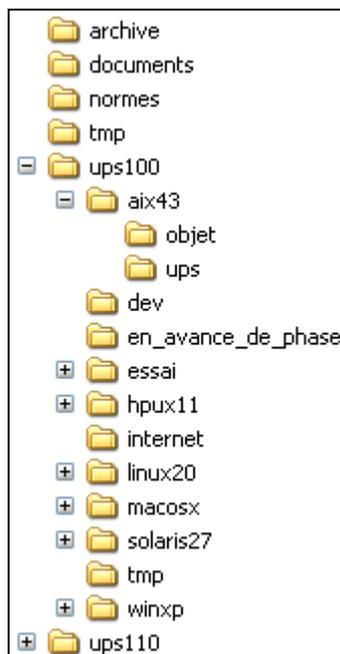


Diagramme 70 – Organisation des versions-révisions

6.1 Répertoire Archive

Ce répertoire contient les fichiers archives :

- **Des versions d'Up ! Application System en production.**
Les fichiers ne sont plus disponibles en ligne. Il faut d'abord les désarchiver avant de pouvoir les exploiter.
Ces archives sont également conservées sur des supports magnétiques.
Un fichier d'une archive ne doit jamais être modifié puisque la version-révision a été figée. Il ne peut être que consulté.

M

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **Des sauvegardes des versions courantes d'Up ! Application System.**
Il s'agit des sauvegardes quotidiennes, hebdomadaires et mensuelles. Elles sont également conservées sur des supports magnétiques.

6.2 Répertoire Documents

Ce répertoire contient les documents de la société. Il y a notamment le répertoire **Production** qui contient tous les documents internes relatifs à **Up ! Application System**.

Ces documents sont regroupés par version-révision pour lequel il existe un répertoire. Il y a ensuite un projet par module donc un sous-répertoire par module. Tous les documents sont gérés conformément à la **Méthode documentaire** [A1].

L'organisation est la suivante :

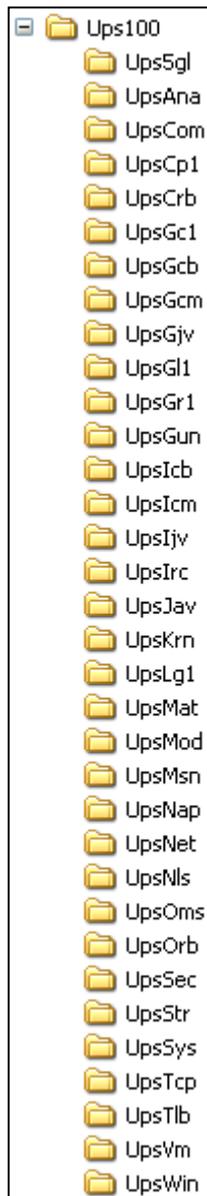


Diagramme 71 – Organisation des documents internes

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

6.3 Répertoire Normes

Ce répertoire contient les documents officiels produits par les constructeurs et les éditeurs de logiciels.

Ce sont les documents qui permettent d'interfacer **Up ! Application System** avec les logiciels de base et de respecter les standards des organisations internationales.

6.4 Répertoire Tmp

Ce répertoire est un répertoire de travail permettant par exemple de descendre une version-révision archivée.

6.5 Répertoire « UpsVRC »

Ce répertoire contient une version / révision / correction complète d'**Up ! Application Système**. L'organisation est la suivante :

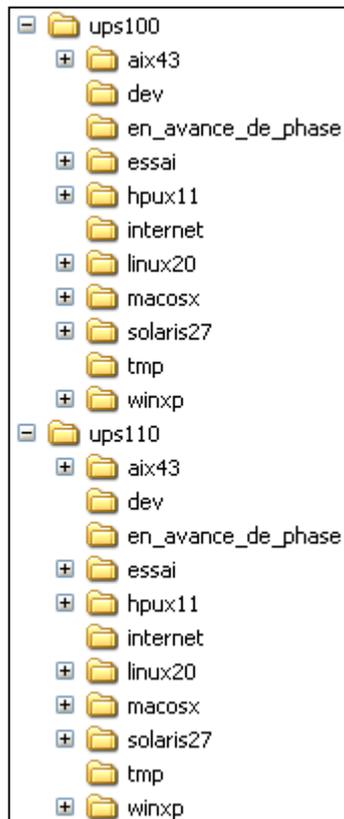


Diagramme 72 – Organisation des versions-révisions d'Up ! Application System

6.5.1 Répertoire « Plate-forme »

Il y a un répertoire par plate-forme cible de compilation :

- **Aix 43.**
- **HpUx 11.**
- **Linux 20.**

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **Mac Os X.**
- **Solaris 2.7.**
- **Windows XP.**
- Etc.

Voici l'organisation :

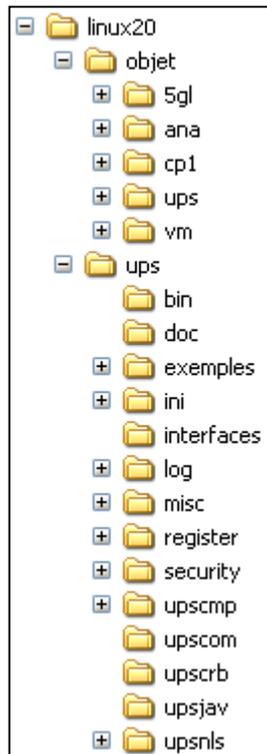


Diagramme 73 – Organisation du répertoire d'une plate-forme

6.5.1.1 Répertoire Objet

Ce répertoire contient les répertoires de compilation des modules pour la plate-forme. Chacun d'eux contient un sous-répertoire par bibliothèque.

Voici l'organisation :

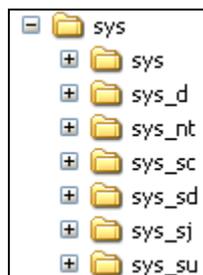


Diagramme 74 – Organisation du répertoire objet

6.5.1.2 Répertoire Ups

Ce répertoire contient la version courante d'**Up ! Application System** pour la plate-forme.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

6.5.2 Répertoire Dev

Ce répertoire contient la liste des fichiers sources d'*Up ! Application System*. Voici l'organisation :

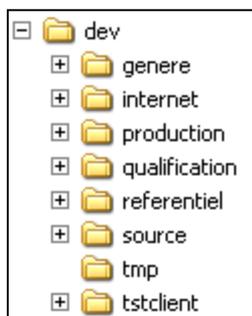


Diagramme 75 – Organisation du répertoire des développements

6.5.2.1 Répertoire Genere

Ce répertoire contient les fichiers sources des modules générés automatiquement. Il y a un sous-répertoire par module qui contient un sous-répertoire par bibliothèque.

Voici l'organisation :

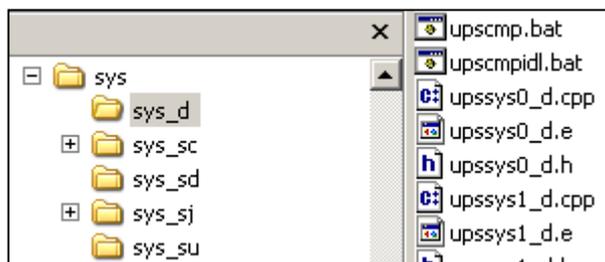


Diagramme 76 – Organisation du répertoire des sources générés automatiquement

6.5.2.2 Répertoire Internet

Ce répertoire contient les fichiers sources des programmes d'*Up ! Community*.

6.5.2.3 Répertoire Production

Ce répertoire contient les fichiers d'intégration produits automatiquement par *Up ! Compiler* lors de la génération des adaptateurs client et serveur pour les modules d'*Up ! Virtual Technical Machine*.

Voici l'organisation :

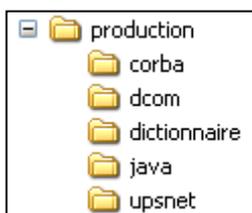


Diagramme 77 – Organisation du répertoire de production

Il contient également les scripts suivants :

- **livrer.sh** ou **livrer.bat** pour la livraison des interfaces des modules.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

- **profile.sh** pour définir les variables environnement.
- **profcmp.bat** pour profiler **Up ! Compiler**.
- **profvtm.bat** pour profiler **Up ! Virtual Technical Machine**.

6.5.2.4 Répertoire Qualification

Ce répertoire contient les fichiers de test employés pour la qualification pour les tests automatisés d'**Up ! Virtual Technical Machine**.

Voici l'organisation :

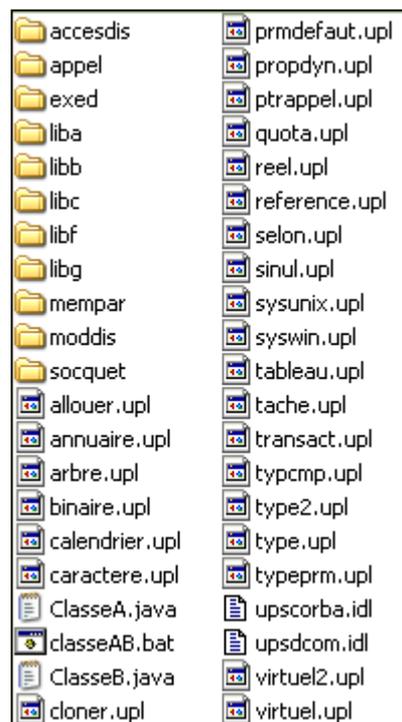


Diagramme 78 – Organisation du répertoire de qualification

6.5.2.5 Répertoire Référentiel

Ce répertoire contient les fichiers communs à toutes distributions d'**Up ! Application System**. Il s'agit de fichiers texte de référence tels :

- Les fichiers d'initialisation standards.
- Les fichiers messages.
- Les fichiers d'exemples.

6.5.2.6 Répertoire Source

Ce répertoire contient les fichiers sources des modules écrits manuellement. Il y a un sous-répertoire par module qui contient un sous-répertoire par bibliothèque.

Voici l'organisation :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

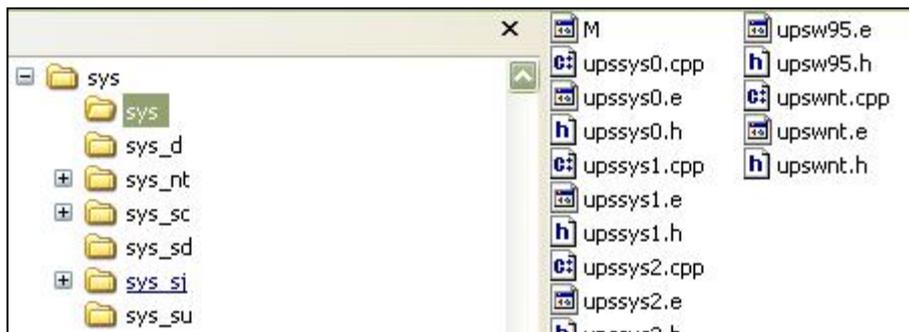


Diagramme 79 – Organisation du répertoire des sources écrits manuellement

6.5.3 Répertoire En_avance_de_phase

Ce répertoire contient les essais informels effectués dans le cadre de :

- **La veille technologique.**
Que proposent les concurrents ? Quelles sont les dernières technologies ?
- **La Recherche & Développement.**
L'objectif est d'anticiper les nouveautés à venir dans la conception des fonctionnalités du moment de la sorte à éviter de :
 - **Refondre l'architecture des modules.**
Cela est du temps et de l'énergie perdus.
 - **Mécontenter les clients.**
Par manque de compatibilité lors de la montée de version.

6.5.4 Répertoire Essai

Ce répertoire contient un sous-répertoire par ingénieur lui permettant de travailler sur la version courante.

Voici l'organisation :

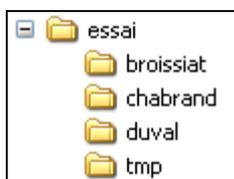


Diagramme 80 – Organisation du répertoire de travail

6.5.5 Répertoire Internet

Ce répertoire contient l'original du **Internet** de la version-révision d'**Up ! Application System** ainsi que les programmes d'**Up ! Community**.

Voici l'organisation :

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		



Diagramme 81 – Organisation du répertoire du site Internet

6.5.6 Répertoire Tmp

Ce répertoire est un répertoire de travail permettant par exemple de copier les fichiers source d'un module.

6.6 Fichiers propres à une plate-forme

Les fichiers propres à une plate-forme sont rangés dans le répertoire de développement. Ils sont les suivants :

- **Macintosh, Os 390 et Unix.**
Up ! Application System est compilé avec **Gnu C Compiler**. Le fichier de **makefile** est **ups.mak**.
- **Os 400.**
Aucun fichier n'est défini à ce jour.
- **Windows.**
Up ! Application System est compilé avec **Microsoft Visual Studio**. Il y a un espace de travail pour tous les modules de nom **ups.dsw** et un projet par module d'extension **dsp**.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

7 Normes de sûreté de fonctionnement

Les normes de sûreté de fonctionnement décrite dans ce chapitre sont à mettre en oeuvre sans réserve pour l'écriture d'un module directement en **C--**.

En cas d'anomalie détectée, soit :

- Une exception est envoyée.
Pour les fonctions exposées sous forme d'une **Application Program Interface (API)** d'**Up ! Virtual Technical Machine**. Par exemple, **FluxNonOuvertEnEcriture** est envoyée.
- Un code retour désignant la non-exécution de la fonction est envoyé.
Pour les fonctions non exposées sous forme d'une **Application Program Interface (API)** d'**Up ! Virtual Technical Machine**.

7.1 Gestion des ressources

Par convention, chaque module, qui alloue une ressource, est chargé de la libérer au travers d'une fonction appropriée.

Au pire, la réallocation des ressources s'effectue dans une des fonctions **Terminer** du composant du module en charge de sa gestion.

7.2 Initialisation

Chaque composant ne doit pas supposer que les variables globales ou locales sont pré-initialisées.

Pour les variables globales, leur initialisation doit se réaliser dans la fonction **Initialiser** de chaque composant et leur éventuelle réallocation doit se réaliser dans la fonction **Terminer** de chaque composant.

7.3 Protection des unions

Chaque union est protégée par un champ de type énuméré permettant de signifier quelle branche sélectionner. Ce principe permet d'éviter d'employer une mauvaise branche.

Voici un exemple :



Plan de programmation

Date rédaction :

21 avril 2004.

Diffusion restreinte

Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

```
enum EnuMonMdl
{
    EN_Entier=1,
    EN_Reel=2,
    EN_Objet=3
};

/*****/
typedef struct typmonmdlmontype
/* Objet : Exemple d'union protegee. */
/*****/
{
    TypUpsVmShort Type;
    /* Type de la valeur. */
    union
    {
        struct
        {
            ...
        } Entier;
        /* Pour EN_Entier. */
        struct
        {
            ...
        } Reel;
        /* Pour EN_Reel. */
        struct
        {
            ...
        } Objet;
        /* Pour EN_Objet. */
    } Selection;
    /* Selection en fonction du type. */
} TypMonMdlMonType;
```

Texte 82 – Exemple de protection des unions

M Cela signifie qu'aucune union n'est déclarée en directe. Elles sont toutes encapsulées dans une structure.

7.4 Contrôle des adresses

7.4.1 Ressources

Up! System vérifie chaque ressource avant de l'employer en vue de s'assurer qu'elle est dans un état compatible avec l'**Application Program Interface (API)** native de la sorte à ne pas provoquer un dysfonctionnement fatal.

Par exemple :

- Avant d'écrire dans un fichier, il est vérifié que celui-ci est bien ouvert en écriture.
- Avant de réallouer une zone mémoire, il est vérifié qu'elle a bien été allouée.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

7.4.2 Objets

Up ! Object Management System vérifie l'adresse logique de chaque objet avant de calculer l'adresse physique. En cas d'anomalie détectée, l'exception **AdresseInvalide** est envoyée.

La vérification des adresses peut être renforcée si l'option d'exécution **verifieradresse** est activée. En ce cas, les débordements d'index sont détectés.

Up ! Object Management System vérifie qu'un verrou a bien été alloué en écriture avant de valider les modifications à l'adresse physique transmises. En cas d'anomalie détectée, l'exception **VerrouInvalide** est envoyée.

7.5 Contrôle des codes retour

Le code retour de chaque **Application Program Interface (API)** native est testé en vue de détecter un dysfonctionnement au plutôt et d'apporter le diagnostic le plus précis.

7.6 Contrôle de la table des méthodes

Suite au remplissage d'une table de méthodes, il est obligatoire de faire vérifier l'intégrité de cette table de la sorte à ne pas avoir un oubli dans le remplissage, ce qui aurait pour effet d'engendrer un dysfonctionnement fatal.

De ce fait, une méthode ne peut être renseignée à **NULL**, même si elle est optionnelle. Elle doit être renseignée par **UpsKrn->AppellImpossibleApiNull** avec la conversion de prototype appropriée.

Le test de remplissage s'effectue de la manière suivante :

- Pour la table des méthodes d'une bibliothèque autre que native d'un module.
Par l'appel à **UpsKrn->VerifierInterfaceTraitements**.
- Pour la table des méthodes d'une bibliothèque native d'un module.
Par l'appel à **UpsKrn->VerifierInterfaceTraitementsNat**.
- Pour la table des méthodes d'un type.
Par l'appel à **UpsKrn->VerifierInterfaceMethodes**.

Si une méthode est mal renseignée, l'exception **AppellImpossible** est envoyée.

7.7 Contrôle de la taille des tampons

Tous les tampons de calcul ne correspondant pas à un scalaire, qui sont transmis par paramètre de sortie, sont accompagnés de leur taille. Ce principe permet à l'appelé de contrôler un éventuel débordement lors du remplissage

Voici un exemple :

```

/*****
TypUpsVmVoid UpsVmAPI UpsNlsStrCpy(TypUpsVmUnicode *ChaineArrivee,
    TypUpsVmLong TailleChaineArrivee, TypUpsVmUnicode *ChaineDepart)
/* Objet : Copie d'une chaine dans une autre. */
/*****
{
...
}

```

Texte 83 – Exemple de contrôle de la taille des tampons

Si un débordement est détecté pour une chaîne de caractères, l'erreur **UpsNls-100** est envoyée.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

7.8 Contrôle des paramètres obligatoires

En début de chaque fonction, les paramètres obligatoires sont vérifiés de la sorte à éviter un dysfonctionnement fatal. En cas d'anomalie détectée, l'exception à envoyer est **ErreurInterne**.

Voici un exemple :

```

if (!AnaCourant)
{
    (*Ups5GLIntTrtUpsKrn->EnvoyerExceptionStandard)(Session,
        (*Ups5GLIntTrtUpsKrn->UpsException56_get)(Session),
        _T("Ups5GL.AllouerConstante"),
        _T("if (!AnaCourant)"), NULL, NULL, NULL);
    return(NULL);
}

```

Texte 84 – Exemple de contrôle d'un paramètre obligatoire

7.9 Contrôle des cas imprévus

Les inconsistances de situation sont systématiquement détectées, notamment lors de l'emploi de l'instruction **switch ... case**. L'objectif est de parer aux dysfonctionnements engendrés par les combinaisons de cas a priori imprévues que l'utilisateur arrive tout de même à produire.

En cas d'anomalie détectée, l'exception à envoyer est **ErreurInterne**. Voici un exemple :

```

switch (Token)
{
    ...

    default :
        (*Ups5GLIntTrtUpsKrn->EnvoyerExceptionStandard)(Session,
            (*Ups5GLIntTrtUpsKrn->UpsException56_get)(Session),
            _T("Ups5GL.AllouerExpression2"),
            _T("switch (Token)"), NULL, NULL, NULL);
        return(0);
        break;
}

```

Texte 85 – Exemple de contrôle des cas imprévus

7.10 Formats d'enregistrement

Les formats d'enregistrement des applications sont verbeux, en utilisant au maximum les instructions d'**Up! 5GL**, quitte à les étendre, au détriment des encodages binaires indéchiffrables par un non initié.

Ce principe permet de réparer aisément un flux en l'éditant lorsque celui est endommagé suite à un dysfonctionnement, en minimisant la perte d'informations qu'il contenait.

7.11 Redondance de code

Lorsqu'il est possible de faire des calculs croisés ou de calculer des vérifications – **checksum** en anglais –, cela doit être effectué quand cela n'est pas trop coûteux.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

8 Index des Application Program Interfaces

8.1 Par module

Le préfixe **MI** signifie que l'**Application Program Interface (API)** est accessible via l'interface avec la machine.

Le préfixe **SFI** signifie que l'**Application Program Interface (API)** est accessible via l'interface avec le système de fichiers.

8.1.1 Up ! Kernel

API.	Sémantique.
DiviserEntier.	Division entière contrôlée, équivalente à $/$.
DiviserReel.	Division réelle contrôlée, équivalente à $/$.
ModulerEntier.	Reste de la division entière contrôlée, équivalente à $\%$.
ModulerReel.	Reste de la division réelle contrôlée, équivalente à $\%$.

Texte 87 – API d'Up ! Kernel

8.1.2 Up ! Mathematical

API.	Sémantique.
MI.Abs.	Valeur absolue.
MI.ACos.	Arc-cosinus trigonométrique.
MI.Aleatoire.	Tirage d'un nombre aléatoire.
MI.ASin.	Arc-sinus trigonométrique.
MI.ATan.	Arc-tangente trigonométrique.
MI.ATanH.	Arc-tangente hyperbolique.
MI.Cos.	Cosinus trigonométrique.
MI.CosH.	Cosinus hyperbolique.
MI.Double2Long.	Conversion d'un nombre réel en un nombre entier.
MI.DoubleDigit.	Nombre de chiffres dans la mantisse d'un nombre réel.
MI.ErreurDouble.	Marge de précision des nombres réels.
MI.Exp.	Exponentielle de base e .
MI.FMod.	Reste de la division réelle.
MI.FormaterDouble.	Formatage d'un nombre réel en une chaîne de caractères équivalente.
MI.Log.	Logarithme népérien de base e .
MI.Log10.	Logarithme base 10.
MI.Long2Double.	Conversion d'un nombre entier en un nombre réel.
MI.PartieEntiere.	Partie entière d'un nombre réel.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

API.	Sémantique.
MI.Pow.	Puissance d'un nombre réel par un second nombre réel.
MI.Sqrt.	Racine carrée.
MI.Sin.	Sinus trigonométrique.
MI.SinH.	Sinus hyperbolique.
MI.Tan.	Tangente trigonométrique.
MI.TanH.	Tangente hyperbolique.

Texte 88 – API d'Up ! Mathematical

8.1.3 Up ! Natural Language Support

API.	Sémantique.
Ascii2Double.	Conversion d'une chaîne de caractère en sa valeur réelle équivalente.
Ascii2Long.	Conversion d'une chaîne de caractère en sa valeur entière équivalente.
Ascii2UnsignedLong.	Conversion d'une chaîne de caractère en sa valeur entière équivalente non signée.
Double2Ascii.	Conversion d'un nombre réel en sa chaîne de caractère équivalente.
IntegerValue.	Valeur entière représentée par le caractère.
IsAlpha.	Vrai si la chaîne de caractères n'est composée que de caractères alphabétiques.
IsAlphaNum.	Vrai si la chaîne de caractères n'est composée que de caractères alphabétiques ou de chiffres.
IsDigit.	Vrai si la chaîne de caractères n'est composée que de caractères alphabétiques en majuscule.
IsLower.	Vrai si la chaîne de caractères n'est composée que de caractères alphabétiques en minuscule.
IsMark.	Vrai si la chaîne de caractères n'est composée que de signes de ponctuation.
IsNul.	Vrai si la chaîne de caractères est nulle.
IsSpace.	Vrai si la chaîne de caractères n'est composée que de séparateurs.
IsUpper.	Vrai si la chaîne de caractères n'est composée que de caractères alphabétiques en majuscule.
Long2Ascii.	Conversion d'un nombre entier en sa chaîne de caractère équivalente.
Lower.	Passage de la chaîne de caractère en minuscule.
RealValue.	Valeur réelle représentée par le caractère.
StrCat.	Concaténation d'une chaîne de caractères.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

API.	Sémantique.
StrCmp.	Comparaison de deux chaînes de caractères.
StrCpy.	Copie d'une chaîne de caractères.
StrICmp.	Comparaison de deux chaînes de caractères en ignorant la casse.
StrLen.	Longueur d'une chaîne de caractères en nombre de caractères.
StrLenB.	Longueur d'une chaîne de caractères en nombre d'octets.
StrNCat.	Concaténation partielle d'une chaîne de caractères.
StrNCmp.	Comparaison partielle de deux chaînes de caractères.
StrNCpy.	Copie partielle d'une chaîne de caractères.
StrNICmp.	Comparaison partielle de deux chaînes de caractères.
UnsignedLong2Ascii.	Conversion d'un nombre entier non signé en sa chaîne de caractère équivalente.
UpLower.	Passage de la chaîne de caractère en majuscule, sauf l'initiale qui est en majuscule.
Upper.	Passage de la chaîne de caractère en majuscule.

Texte 89 – API d'Up ! Natural Language Support

8.1.4 Up ! Network

API.	Sémantique.
MI.ConvertirAdresseTcpIpEnNom.	Convertit une adresse Ip en un nom de machine.
MI.ConvertirNomEnAdresseTcpIp.	Convertit un nom en une adresse Ip .
MI.ConvertirNomEnPortTcpIp.	Convertit un nom en une adresse Ip .
MI.ConvertirPortTcpIpEnNom.	Convertit une adresse Ip en nom de port.

Texte 90 – API d'Up ! Network

8.1.5 Up ! System

API.	Sémantique.
AjouterAlarme.	Ajoute une alarme.
AjouterEntreeContexte.	Ajoute une entrée au contexte à lire.
AnalyserChaine.	Décodage d'une chaîne de caractère.
BornerRepertoire.	Borne le chemin d'accès.
CalculerRepertoireIni.	Calcule le répertoire où sont conservés les fichiers d'initialisation des utilisateurs.
ChangerInstanceOption.	Change l'interface de données d'une extension d'un module.
ChargerOptions.	Charge les dénominations des extensions d'un module.
Contexte.	Lit le contexte de l'application.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

API.	Sémantique.
ConvertirVariablesEnvironnement.	Convertit une chaîne de caractères employant des variables environnement.
EcrireJournal.	Ecrit dans le journal où sont inscrits les comptes-rendus.
EtendreTableau.	Etend un tableau dynamique.
FermerJournal.	Ferme le journal où sont inscrits les comptes-rendus.
FluxAnalyserChaine.	Décodage d'une chaîne de caractères lue dans un flux.
FluxDeplacerPointeur.	Déplace la position du pointeur de lecture et d'écriture dans un flux.
FluxEcrireCaractere.	Ecriture d'un caractère dans un flux.
FluxEcrireCaracteres.	Ecriture d'un bloc de caractères dans un flux.
FluxEcrireOctet.	Ecriture d'un octet dans un flux.
FluxEcrireOctets.	Ecriture d'un bloc d'octets dans un flux.
FluxFermer.	Fermeture d'un flux.
FluxFormaterChaine.	Ecriture d'une chaîne de caractères dans un flux.
FluxLireCaracteres.	Lecture d'un bloc de caractères dans un flux.
FluxLireOctets.	Lecture d'un bloc d'octets dans un flux.
FluxLirePositionPointeur.	Lit la position du pointeur de lecture et d'écriture dans un flux.
FluxLireUnCaractere.	Lecture d'un caractère dans un flux.
FluxLireUneLigne.	Lecture d'une ligne dans un flux.
FluxLireUnOctet.	Lecture d'un octet dans un flux.
FluxLireUnOctetText.	Lecture d'un octet en mode texte dans un flux.
FluxRejeterUnCaractere.	Remise dans le tampon du flux d'un caractère.
FluxRejeterUnOctet.	Remise dans le tampon du flux d'un octet.
FluxVider.	Vidage du tampon d'écriture d'un flux.
FormaterChaine.	Formatage d'une chaîne de caractères.
IlYAUUnAccesDistant.	Retourne Vrai s'il y a un accès distant dans le nom du fichier ou du répertoire.
InitialiserContexte.	Initialise la lecture du contexte de l'application.
MI.AllouerMemoirePartagee.	Alloue une mémoire partagée.
MI.AllouerMutex.	Alloue un mutex.
MI.AllouerSemaphore.	Alloue un sémaphore.
MI.ArreterProcessus.	Arrête un processus.
MI.ArreterProgramme.	Arrête le programme.
MI.ArreterService.	Arrête un service.
MI.ArreterSysteme.	Arrêt le système.
MI.ArreterThread.	Arrête l'exécution d'un thread.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

API.	Sémantique.
MI.Attendre.	Attends un certain nombre de secondes.
MI.ChangerEnvironnement.	Changement de la valeur d'une variable environnement.
MI.ChangerPrioriteProcessus.	Change la priorité d'un processus.
MI.ChercherService.	Retourne Vrai si un service est en cours d'exécution.
MI.CommuterThreads.	Commute l'exécution du thread courant à un autre.
MI.CreerThread.	Crée un nouveau thread.
MI.DateEnSecondes.	Nombre de secondes écoulées depuis le 1 ^{er} janvier 1970.
MI.DateEtHeure.	Date et heure système.
MI.DebuterSectionCritique.	Entre dans une section critique.
MI.DecoderLong.	Permute les octets d'un TypUpsVmLong .
MI.DecoderShort.	Permute les octets d'un TypUpsVmShort .
MI.DecoderUnsignedLong.	Permute les octets d'un TypUpsVmUnsignedLong .
MI.DecoderUnsignedShort.	Permute les octets d'un TypUpsVmUnsignedShort .
MI.DemarrerService.	Démarre un service.
MI.EffacerEcran.	Efface l'écran de la console.
MI.EffacerMemoire.	Initialisation d'une zone mémoire.
MI.Executer.	Exécute un nouveau programme.
MI.FermerSession.	Ferme la session de l'utilisateur.
MI.GererSignaux.	Gère la réception des signaux.
MI.Imprimer.	Imprime un fichier.
MI.LibererMemoirePartagee.	Libère une mémoire partagée.
MI.LibererMutex.	Libère un mutex.
MI.LibererSemaphore.	Libère un sémaphore.
MI.LireCodeErreur.	Lit le code erreur du dernier appel système.
MI.LireCodeRetourProcessus.	Lit le code retour d'un processus lancé par Executer .
MI.LireEntreeStandard.	Lit la sortie standard.
MI.LireEnvironnement.	Lecture d'une variable environnement.
MI.LireErreurStandard.	Lit la sortie standard.
MI.LireEtatSuspensionThread.	Lit l'état de suspension d'un thread.
MI.LireInformationsSystemeDExploitation.	Les informations caractérisant le système d'exploitation.
MI.LireNomGroupe.	Lit le nom du groupe de travail de l'utilisateur.
MI.LireNomUtilisateur.	Lit le nom de l'utilisateur.
MI.LireNumeroGroupe.	Lit le numéro du groupe de travail de l'utilisateur.



Plan de programmation

Date rédaction :

21 avril 2004.

Diffusion restreinte

Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

API.	Sémantique.
MI.LireNumeroProcessus.	Lit le numéro du processus courant.
MI.LireNumeroThread.	Lire le numéro du thread.
MI.LireNumeroUtilisateur.	Lit le numéro de l'utilisateur.
MI.LirePageDeCodeSysteme.	Lit la page de codes du système d'exploitation.
MI.LirePrioriteProcessus.	Lit la priorité d'un processus.
MI.LireSortieStandard.	Lit la sortie standard.
MI.LireTempsProcessus.	Lit les temps d'exécution du processus courant.
MI.ListerProcessus.	Liste les processus s'exécutant sur la machine.
MI.PrendrejetonSemaphore.	Prends le jeton du sémaphore.
MI.PrendreMutex.	Prends le mutex en début de section critique.
MI.RelancerProgramme.	Relance le programme.
MI.RelancerSysteme.	Relance le système.
MI.RemettreJetonSemaphore.	Remet le jeton du sémaphore.
MI.RemettreMutex.	Remet le mutex en fin de section critique.
MI.RepertoireInstallationSysteme.	Répertoire d'installation du système d'exploitation.
MI.ReprendreExecutionThread.	Reprend l'exécution d'un thread.
MI.RetourInterAppel.	Saut au point de retour posé par UpsVmSetJmp .
MI.SuspendreExecutionThread.	Suspend l'exécution d'un thread.
MI.TerminerSectionCritique.	Sort d'une section critique.
MI.ThreadEstEnVie.	Retourne Vrai si le thread est en vie.
OuvrirJournal.	Ouvre le journal où sont inscrits les comptes-rendus.
RechercherFichierDansChemin.	Recherche un fichier dans une liste de chemins.
ReferenceExtension.	Référence l'interface de données d'un module en tant qu'extension d'un autre.
SFI.ChangerDroitsDAcces.	Change les droits d'accès d'un fichier ou d'un répertoire.
SFI.ChangerRepertoireCourant.	Change le répertoire de travail du thread courant.
SFI.CreerRepertoire.	Crée un répertoire.
SFI.CreerUnRaccourci.	Crée un raccourci vers un fichier ou un répertoire.
SFI.DemonterUnVolumeAmovible.	Démonte un volume amovible tel un Cd-Rrom .
SFI.EffacerFichier.	Efface un fichier.
SFI.EffacerRepertoire.	Efface un répertoire.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

API.	Sémantique.
SFI.IlexisteFichier.	Retourne Vrai s'il existe un fichier.
SFI.IlexisteRepertoire.	Retourne Vrai s'il existe un répertoire.
SFI.LireDroitsDAcces.	Lit les droits d'accès à un fichier ou à un répertoire.
SFI.LireInformations.	Lit les informations caractérisant un fichier ou un répertoire.
SFI.LireRepertoireCourant.	Lit le répertoire de travail du thread courant.
SFI.ListerFichiers.	Liste les fichiers contenus dans un répertoire.
SFI.ListerUnites.	Liste les unités de stockages de la machine.
SFI.MonterUnVolumeAmovible.	Monte un volume amovible tel un Cd-Rrom .
SFI.Ouvrir.	Ouverture d'un fichier.
SFI.RenommerFichier.	Renomme un fichier.
SFI.RenommerRepertoire.	Renomme un répertoire.
SFI.Synchroniser.	Synchronisation des tampons d'écriture des disques.
SFI.TraduireFichier.	Convertir les séparateurs de répertoires d'un nom de fichier ou de répertoire.

Texte 91 – API d'Up ! System

8.1.6 Up ! Virtual Technical Machine

API.	Sémantique.
MI.AllouerMemoire.	Allocation dynamique d'une zone mémoire.
MI.ComparerMemoire.	Comparaison des octets de deux zones mémoires.
MI.CopierMemoire.	Copie d'une zone mémoire.
MI.LibererMemoire.	Libération d'une zone mémoire allouée dynamiquement.
MI.ReallouerMemoire.	Ré allocation dynamique d'une zone mémoire.

Texte 92 – API d'Up ! Virtual Technical Machine

8.2 Par dénomination

8.2.1 Via le nom de l'API standard Up ! Virtual Technical Machine

API.	Module
Abs.	Up ! Mathematical.
ACos.	Up ! Mathematical.
AjouterAlarme.	Up ! System.
AjouterEntreeContexte.	Up ! System.
Aleatoire.	Up ! Mathematical.
AllouerMemoire.	Up ! Virtual Technical Machine.
AllouerMemoirePartagee.	Up ! System.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

API.	Module
AllouerMutex.	<i>Up ! System.</i>
AllouerSemaphore.	<i>Up ! System.</i>
AnalyserChaine.	<i>Up ! System.</i>
ArreterProcessus.	<i>Up ! System.</i>
ArreterProgramme.	<i>Up ! System.</i>
ArreterService.	<i>Up ! System.</i>
ArreterSysteme.	<i>Up ! System.</i>
ArreterThread.	<i>Up ! System.</i>
Ascii2Double.	<i>Up ! Natural Language Support.</i>
Ascii2Long.	<i>Up ! Natural Language Support.</i>
Ascii2UnsignedLong.	<i>Up ! Natural Language Support.</i>
ASin.	<i>Up ! Mathematical.</i>
ATan.	<i>Up ! Mathematical.</i>
Attendre.	<i>Up ! System.</i>
BornerRepertoire.	<i>Up ! System.</i>
CalculerRepertoireIni.	<i>Up ! System.</i>
ChangerDroitsDAcces.	<i>Up ! System.</i>
ChangerEnvironnement.	<i>Up ! System.</i>
ChangerInstanceOption.	<i>Up ! System.</i>
ChangerPrioriteProcessus.	<i>Up ! System.</i>
ChangerRepertoireCourant.	<i>Up ! System.</i>
ChargerOptions.	<i>Up ! System.</i>
ChercherService.	<i>Up ! System.</i>
CommuterThreads.	<i>Up ! System.</i>
ComparerMemoire.	<i>Up ! Virtual Technical Machine.</i>
Contexte.	<i>Up ! System.</i>
ConvertirAdresseTcpIpEnNom.	<i>Up ! Network.</i>
ConvertirNomEnAdresseTcpIp.	<i>Up ! Network.</i>
ConvertirNomEnPortTcpIp.	<i>Up ! Network.</i>
ConvertirPortTcpIpEnNom.	<i>Up ! Network.</i>
ConvertirVariablesEnvironnement.	<i>Up ! System.</i>
CopierMemoire.	<i>Up ! Virtual Technical Machine.</i>
Cos.	<i>Up ! Mathematical.</i>
CosH.	<i>Up ! Mathematical.</i>
CreerRepertoire.	<i>Up ! System.</i>



Plan de programmation

Date rédaction :

21 avril 2004.

Diffusion restreinte

Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

API.	Module
CreerThread.	<i>Up ! System.</i>
CreerUnRaccourci.	<i>Up ! System.</i>
DateEnSecondes.	<i>Up ! System.</i>
DateEtHeure.	<i>Up ! System.</i>
DebuterSectionCritique.	<i>Up ! System.</i>
DecoderLong.	<i>Up ! System.</i>
DecoderShort.	<i>Up ! System.</i>
DecoderUnsignedLong.	<i>Up ! System.</i>
DecoderUnsignedShort.	<i>Up ! System.</i>
DemarrerService.	<i>Up ! System.</i>
DemonterUnVolumeAmovible.	<i>Up ! System.</i>
DiviserEntier.	<i>Up ! Kernel.</i>
DiviserReel.	<i>Up ! Kernel.</i>
Double2Ascii.	<i>Up ! Natural Language Support.</i>
Double2Long.	<i>Up ! Mathematical.</i>
DoubleDigit.	<i>Up ! Mathematical.</i>
EcrireJournal.	<i>Up ! System.</i>
EffacerEcran.	<i>Up ! System.</i>
EffacerFichier.	<i>Up ! System.</i>
EffacerMemoire.	<i>Up ! Virtual Technical Machine.</i>
EffacerRepertoire.	<i>Up ! System.</i>
ErreurDouble.	<i>Up ! Mathematical.</i>
EtendreTableau.	<i>Up ! System.</i>
Executer.	<i>Up ! System.</i>
Exp.	<i>Up ! Mathematical.</i>
FermerJournal.	<i>Up ! System.</i>
FermerSession.	<i>Up ! System.</i>
FluxAnalyserChaine.	<i>Up ! System.</i>
FluxDeplacerPointeur.	<i>Up ! System.</i>
FluxEcrireCaractere.	<i>Up ! System.</i>
FluxEcrireCaracteres.	<i>Up ! System.</i>
FluxEcrireOctet.	<i>Up ! System.</i>
FluxEcrireOctets.	<i>Up ! System.</i>
FluxFermer.	<i>Up ! System.</i>
FluxFormaterChaine.	<i>Up ! System.</i>



Plan de programmation

Date rédaction :

21 avril 2004.

Diffusion restreinte

Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

API.	Module
FluxLireCaracteres.	<i>Up ! System.</i>
FluxLireOctets.	<i>Up ! System.</i>
FluxLirePositionPointeur.	<i>Up ! System.</i>
FluxLireUnCaractere.	<i>Up ! System.</i>
FluxLireUneLigne.	<i>Up ! System.</i>
FluxLireUnOctet.	<i>Up ! System.</i>
FluxLireUnOctetText.	<i>Up ! System.</i>
FluxRejeterUnCaractere.	<i>Up ! System.</i>
FluxRejeterUnOctet.	<i>Up ! System.</i>
FluxVider.	<i>Up ! System.</i>
FMod.	<i>Up ! Mathematical.</i>
FormaterChaine.	<i>Up ! System.</i>
FormaterDouble.	<i>Up ! Mathematical.</i>
GererSignaux.	<i>Up ! System.</i>
IlExisteFichier.	<i>Up ! System.</i>
IlExisteRepertoire.	<i>Up ! System.</i>
IlYAUUnAccesDistant.	<i>Up ! System.</i>
Imprimer.	<i>Up ! System.</i>
InitialiserContexte.	<i>Up ! System.</i>
IntegerValue.	<i>Up ! Natural Language Support.</i>
IsAlpha.	<i>Up ! Natural Language Support.</i>
IsAlphaNum.	<i>Up ! Natural Language Support.</i>
IsDigit.	<i>Up ! Natural Language Support.</i>
IsLower.	<i>Up ! Natural Language Support.</i>
IsMark.	<i>Up ! Natural Language Support.</i>
IsNull.	<i>Up ! Natural Language Support.</i>
IsSpace.	<i>Up ! Natural Language Support.</i>
IsUpper.	<i>Up ! Natural Language Support.</i>
LibererMemoire.	<i>Up ! Virtual Technical Machine.</i>
LibererMemoirePartagee.	<i>Up ! System.</i>
LibererMutex.	<i>Up ! System.</i>
LibererSemaphore.	<i>Up ! System.</i>
LireCodeErreur.	<i>Up ! System.</i>
LireCodeRetourProcessus.	<i>Up ! System.</i>
LireDroitsDAcces.	<i>Up ! System.</i>



Plan de programmation

Date rédaction :

21 avril 2004.

Diffusion restreinte

Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

API.	Module
LireEntreeStandard.	<i>Up ! System.</i>
LireEnvironnement.	<i>Up ! System.</i>
LireErreurStandard.	<i>Up ! System.</i>
LireEtatSuspensionThread.	<i>Up ! System.</i>
LireInformations.	<i>Up ! System.</i>
LireInformationsSystemeDExploitation.	<i>Up ! System.</i>
LireNomGroupe.	<i>Up ! System.</i>
LireNomUtilisateur.	<i>Up ! System.</i>
LireNumeroGroupe.	<i>Up ! System.</i>
LireNumeroProcessus.	<i>Up ! System.</i>
LireNumeroThread.	<i>Up ! System.</i>
LireNumeroUtilisateur.	<i>Up ! System.</i>
LirePageDeCodeSysteme.	<i>Up ! System.</i>
LirePrioriteProcessus.	<i>Up ! System.</i>
LireRepertoireCourant.	<i>Up ! System.</i>
LireSortieStandard.	<i>Up ! System.</i>
LireTempsProcessus.	<i>Up ! System.</i>
ListerFichiers.	<i>Up ! System.</i>
ListerProcessus.	<i>Up ! System.</i>
ListerUnites.	<i>Up ! System.</i>
Log.	<i>Up ! Mathematical.</i>
Log10.	<i>Up ! Mathematical.</i>
Long2Ascii.	<i>Up ! Natural Language Support.</i>
Long2Double.	<i>Up ! Mathematical.</i>
Lower.	<i>Up ! Natural Language Support.</i>
ModulerEntier.	<i>Up ! Kernel.</i>
ModulerReel.	<i>Up ! Kernel.</i>
MonterUnVolumeAmovible.	<i>Up ! System.</i>
Ouvrir.	<i>Up ! System.</i>
OuvrirJournal.	<i>Up ! System.</i>
PartieEntiere.	<i>Up ! Mathematical.</i>
Pow.	<i>Up ! Mathematical.</i>
PrendrejetonSemaphore.	<i>Up ! System.</i>
PrendreMutex.	<i>Up ! System.</i>
ReallouerMemoire.	<i>Up ! Virtual Technical Machine.</i>



Plan de programmation

Date rédaction :

21 avril 2004.

Diffusion restreinte

Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

API.	Module
Realvalue.	<i>Up ! Natural Language Support.</i>
RechercherFichierDansChemin.	<i>Up ! System.</i>
ReferenceExtension.	<i>Up ! System.</i>
RelancerProgramme.	<i>Up ! System.</i>
RelancerSysteme.	<i>Up ! System.</i>
RemettreJetonSemaphore.	<i>Up ! System.</i>
RemettreMutex.	<i>Up ! System.</i>
RenommerFichier.	<i>Up ! System.</i>
RenommerRepertoire.	<i>Up ! System.</i>
RepertoireInstallationSysteme.	<i>Up ! System.</i>
ReprendreExecutionThread.	<i>Up ! System.</i>
RetourInterAppel.	<i>Up ! System.</i>
Sin.	<i>Up ! Mathematical.</i>
SinH.	<i>Up ! Mathematical.</i>
Sqrt.	<i>Up ! Mathematical.</i>
StrCat.	<i>Up ! Natural Language Support.</i>
StrCmp.	<i>Up ! Natural Language Support.</i>
StrCpy.	<i>Up ! Natural Language Support.</i>
StrICmp.	<i>Up ! Natural Language Support.</i>
StrLen.	<i>Up ! Natural Language Support.</i>
StrLenB.	<i>Up ! Natural Language Support.</i>
StrNCat.	<i>Up ! Natural Language Support.</i>
StrNCmp.	<i>Up ! Natural Language Support.</i>
StrNCpy.	<i>Up ! Natural Language Support.</i>
StrNICmp.	<i>Up ! Natural Language Support.</i>
SuspendreExecutionThread.	<i>Up ! System.</i>
Synchroniser.	<i>Up ! System.</i>
Tan.	<i>Up ! Mathematical.</i>
TanH.	<i>Up ! Mathematical.</i>
TerminerSectionCritique.	<i>Up ! System.</i>
ThreadEstEnVie.	<i>Up ! System.</i>
TraduireFichier.	<i>Up ! System.</i>
UnsignedLong2Ascii.	<i>Up ! Natural Language Support.</i>
UpLower.	<i>Up ! Natural Language Support.</i>
Upper.	<i>Up ! Natural Language Support.</i>

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

Texte 93 – Index général des APIs par leur nom selon le standard Up ! Virtual Technical Machine

8.2.2 Via le nom de l'API standard C Posix

API standard C.	API Up ! Virtual Technical Machine
%.	ModulerEntier, ModulerReel.
/.	DiviserEntier, DiviserReel.
abs.	Abs.
acos.	ACos.
acosh.	ACosH.
alarm.	AjouterAlarme.
asin.	ASinH.
atan.	ATan.
atof.	Ascii2Double.
atoi.	Ascii2Long, Ascii2UnsignedLong.
cos.	Cos.
cosh.	CosH.
exit.	ArreterProgramme.
exp.	Exp.
fclose.	FluxFermer.
fcvt.	FormaterDouble.
feof.	Témoin dans toutes les APIs de lecture / écriture.
fflush.	FluxVider.
fgetc.	FluxLireUnOctet.
fmod.	FMod.
fprintf.	FluxFormaterChaine.
fputc.	FluxEcrireUnOctet.
fread.	FluxLireOctets.
free.	LibererMemoire.
fscanf.	FluxAnalyserChaine.
fseek.	FluxDeplacerPointeur.
ftell.	FluxLirePositionPointeur.
ftol.	Double2Long.
fwrite.	FluxEcrireOctets.
getc.	LireEntreeStandard.FluxLireUnOctet
getenv.	LireEnvironnement.
gethostbyaddr.	ConvertirAdresseTcpIpEnNom.
gethostbyname.	ConvertirNomEnAdresseTcpIp.



Plan de programmation

Date rédaction :

21 avril 2004.

Diffusion restreinte

Date validation :

Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc

API standard C.	API Up ! Virtual Technical Machine
getpid.	LireNumeroProcessus.
getprotobyname.	ConvertirNomEnPortTcpIp.
getprotobynumber.	ConvertirPortTcpIpEnNom.
isalpha.	IsAlpha.
isalnum.	IsAlphaNum.
isdigit.	IsDigit.
islower.	IsLower.
log.	Log.
log10.	Log10.
longjmp.	LongJump.
lower.	Lower.
malloc.	AllouerMemoire.
memcmp.	ComparerMemoire.
memcpy.	CopierMemoire.
memset.	EffacerMemoire.
pow.	Pow.
printf.	Printf.
rand.	Aleatoire.
realloc.	Realloc.
setenv.	ChangerEnvironnement.
sin.	Sin.
sinh.	SinH.
sleep.	Attendre.
sprintf.	FormaterChaine.
sqrt.	Sqrt.
sscanf.	AnalyserChaine.
strcat.	StrCat.
strcmp.	StrCmp.
strcpy.	StrCpy.
stricmp.	StrICmp.
strlen.	StrLen.
strncat.	StrNCat.
strncmp.	StrNCmp.
strncpy.	StrNCpy.
sync.	Synchroniser.

	Plan de programmation	Date rédaction : 21 avril 2004.
	Diffusion restreinte	Date validation :
Référence : UpComp-UpsVtm-000003-A Plan de programmation.doc		

API standard C.	API Up ! Virtual Technical Machine
tan.	Tan.
tanh.	TanH.
localtime.	DateEtHeure.
time.	DateEnSecondes.
upper.	Upper.

Texte 94 – Index général des APIs par leur nom selon le standard C Posix

Fin de document