	Spécification technique du module	**Date rédaction : 14 mars 2004.**
	Diffusion restreinte	**Date validation :**
Référence : UpComp-UpsAna-000003-A Spécification technique du module UpsAna.doc		

Suivi des versions-révisions et des validations du document.			
Ce document annule et remplace tout document diffusé de version-révision antérieure. 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. 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.			
Version.	**Date.**	**Auteurs.**	**Création, modification ou validation.**
A	23 nov. 2003.	JPD.	Création.

1 Tables

1.1 Table des matières

1.2 Table des illustrations

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, abré.	**Définition du terme, du concept ou de l'abréviation.**
Paquet lexical	Voir page 8.
Règle d'analyse	Voir page 10.
Token	Voir page 8.
Grammaire	Voir page 10.

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 indicé par ***A1***, ***A2***, ***A3***, etc. Un document en référence est indicé par ***R1***, ***R2***, ***R3***, etc.

Index.	Nom du document.	Commentaire.
A1	UpComp-Plan Qualité-000005	Méthode documentaire.
A2	UpComp-Plan Qualité-000006	Processus de management de projet.
A3	UpComp-Plan Qualité-000046	Méthode de spécification technique d'un module.
A4	UpComp-UspAna-000002	Plan documentaire du projet.
A5	UpComp-UpsVm-000003	Plan de programmation.
A6	UpComp-UpsVm-000004	Programmation en ***C--***.
R7	http://www.up-comp.com	Site ***Internet*** d'***Up ! Application System***.
A8	UpComp-UpsKrn-000003	Plan d'écriture d'un module.

3 Introduction

3.1 Objet du document

L'objet de ce document est de décrire le contenu technique du module logiciel ***Up ! Analyzer*** pour le projet ***Up ! Application System***.

Ce document est rédigé et approuvé par la ***Maîtrise d'Oeuvre*** (**MOE**).

3.2 Audience

Ce document s'adresse aux :

- Directeurs de projets et chefs de projets.
  Pour la compréhension du module technique.
- Ingénieurs de développement.
  Pour savoir comment est conçu le module technique.

Pour aider ces personnes à remplir le document ***Spécification technique d'un module***, leur manager et la cellule de support projet se tiennent à leur disposition.

3.3 Pré-requis

Le pré-requis est la connaissance des documents suivants :

- **Méthode documentaire** [A1].
- **Processus de management de projet** [A2].
- **Méthode de spécification technique d'un module** [A3].

Nous rappelons que tous les documents applicables ou référencés pour le projet ***Up ! Application System*** sont tracés dans le **Plan documentaire** [A4].

4 Interaction avec l'environnement

4.1 Description

L'objet du module logiciel ***Up ! Analyzer*** est de d'analyser des flux en vue de reconnaître :

- Des éléments lexicaux – les mots d'une phrase.
  Conformément à un référentiel préétabli.
- Des éléments syntaxiques – les enchaînements des mots en une phrase.
  Conformément à une grammaire préétablie.

Plusieurs analyseurs lexicaux et syntaxiques peuvent coexister. Ils peuvent être indépendants ou interdépendants.

Les référentiels et les grammaires sont construits dynamiquement. Ils peuvent être également adaptés dynamiquement au cours de l'analyse.

4.2 Paramètres

Up ! Analyzer ne possède aucun paramètre.

4.3 Particularités

4.3.1 Compilation

Néant.

4.3.2 Exécution

Néant.

4.4 Application Program Interfaces

Néant.

5 Choix techniques

5.1 Principes

Un analyseur est un automate traitant des flux devant correspondre à des :

- Modèles appelés éléments lexicaux.
  Il s'agit par exemple de mots, de nombres, de signes de ponctuation, etc. dans une phrase.
- Enchaînements de modèles appelés éléments syntaxiques.
  Il s'agit par exemple de phrases indépendantes.
- Enchaînements de phrases appelés éléments sémantiques.
  Il s'agit par exemple de phrases conjointes.

Up ! Analyzer est un analyseur générique dynamique pour les flux ***Unicode 2.0*** pour les phases lexicales et syntaxiques. Quand un élément est reconnu, une fonction de rappel spécifique est appelée. Cette dernière communique à ***Up ! Analyzer*** le résultat de son traitement selon la convention définie par l'énuméré ***EnuUpsAnaAction*** :

- **AC_Succes**.
  L'analyse continue jusqu'à l'atteinte de la fin du flux.
- **AC_ArreterRegle**.
  Une erreur a été détectée, ce qui nécessite d'arrêter de traiter la règle de syntaxe en cours.
  Up ! Analyzer tentera de synchroniser l'analyse sur une autre règle en sautant une partie du flux.
- **AC_ArreterTout**.
  Une erreur a été détectée, ce qui nécessite d'arrêter l'analyse en cours.
- **AC_ReprendreSansErreur**.
  L'analyse devrait continuer jusqu'à l'atteinte de la fin du flux. Cependant, la fonction de rappel demande d'arrêter de traiter la règle de syntaxe en cours.
  Up ! Analyzer tentera de synchroniser l'analyse sur une autre règle en sautant une partie du flux.
- **AC_ArreterToutSansErreur**.
  L'analyse devrait continuer jusqu'à l'atteinte de la fin du flux. Cependant, la fonction de rappel demande d'arrêter l'analyse en cours.
- **AC_SuccesFinal**.
  L'analyse est terminée et la fin du flux doit être atteinte.

En cas d'erreur bénigne correspondant soit à **AC_ArreterRegle** ou soit à **AC_ReprendreSansErreur**, ***Up ! Analyzer*** tente de corriger l'erreur à la volée. Il communique les modifications apportées à une fonction de rappel spécifique, pour laquelle la convention de communication est définie par l'énuméré ***EnuUpsAnaCorrection*** :

- **AC_Ajouter**.
  Ajoute le modèle lexical manquant.
- **AC_AjouterEntier**.
  Ajoute le nombre entier manquant.
- **AC_AjouterReel**.
  Ajoute le nombre réel manquant.
- **AC_AjouterChaineGuillemets**.
  Ajoute la chaîne de caractères entre caractère **guillemet "** manquante.

- **AC_AjouterChaineApostrophes**.
  Ajoute la chaîne de caractères entre caractère **apostrophe '** manquante.
- **AC_Modifier**.
  Modifie le modèle lexical incorrect.
- **AC_Supprimer**.
  Supprime le modèle lexical inutile.

5.2 Analyse lexicale

Cette section décrit comment les analyseurs lexicaux sont gérés par ***Up ! Analyzer***.

Les différentes alternatives possibles sont les suivantes :

- Utiliser un analyseur lexical externe.
  Par exemple, ***Lex***.
- Utiliser un analyseur lexical propriétaire.

La seconde alternative a été retenue pour la raison suivante :

- Besoin de plusieurs analyseurs en simultané.
  Le code en langage ***C*** produit par ***Lex*** ne le permet pas. Il faut le transformer manuellement.
- Besoin d'analyseurs dynamiques.
  Pour ***Up ! 5GL*** qui est extensible et auto-défini. Le code en langage ***C*** produit par ***Lex*** ne le permet pas.
- Fonctionnement en multi-thread.
  Le code en langage ***C*** produit par ***Lex*** ne le permet pas.
- Portabilité complète.
  Le code en langage ***C*** produit par ***Lex*** l'est difficilement puisqu'il fait des calculs d'adressage.

Compte-tenu de ces raisons, il n'est pas possible de changer de choix.

5.2.1 Tokens

&

Un analyseur lexical reconnaît des modèles auxquels des identifiants numériques sont assignés, appelés ***tokens***. Il y a un identifiant différent par modèle.

Un ***token*** a une position dans une ligne donné par l'énuméré ***EnuUpsAnaPositionToken*** :

- **PT_DebutLigne**.
  Le modèle doit être placé en début de ligne.
- **PT_FinLigne**.
  Le modèle doit être placé en fin de ligne.
- **PT_Libre**.
  Le modèle peut être placé n'importe où dans la ligne.

5.2.2 Paquets lexicaux

Au fur et à mesure de la progression de l'analyse, de nouveaux ***tokens*** sont créés correspondant aux nouvelles définitions posées. Cependant, ces définitions peuvent n'être valides que dans un contexte particulier – par exemple une variable locale à une procédure à une fonction.

&

Afin de faciliter la gestion des contextes, les définitions sont regroupées dans des **paquets lexicaux** qui peuvent être sélectionnés ou non. Quand ils sont sélectionnés, les définitions qu'ils regroupent sont reconnues.

5.2.3 Gestion des analyseurs lexicaux

Un nouvel analyseur lexical est créé par l'appel à l'***Application Program Interface*** (**API**) ***AjouterAnalyseurLexical***. Un analyseur lexical existant est détruit par l'appel à l'***Application Program Interface*** (**API**) ***SupprimerAnalyseurLexical***.

Voici les ***Application Program Interface*** (**API**) pour gérer les ***tokens*** :

- ***AjouterSeparateurLexical***.
  Ajoute un nouveau séparateur lexical. Il peut être supprimé ultérieurement par l'appel à ***SupprimerSeparateurLexical***.
- ***AjouterToken***.
  Ajoute un nouveau modèle lexical. Il peut être supprimé ultérieurement par l'appel à ***SupprimerToken***.
- ***UtiliserToken***.
  Spécifie qu'un ***token*** va être utilisé ou non. Il ne peut être supprimé quand il est utilisé.

5.2.4 Déclenchement de l'analyse

Le début de l'analyse via un analyseur particulier s'effectue par l'appel à l'***Application Program Interface*** (**API**) ***DebuterAnalyseLexicale***. Elle se termine par l'appel à l'***Application Program Interface*** (**API**) ***TerminerAnalyseLexicale***.

En cours d'analyse, il est possible de :

- Lire le ***token*** suivant.
  Via l'appel à ***LireToken***.
- Lire le numéro de ligne et de caractère courant.
  Via l'appel à ***LireNumeroLigneColonne***.
- Lire les commentaires.
  Via l'appel à ***ChangerCommentaireToken***. Il s'agit des commentaires collectés depuis le dernier appel à cette ***Application Program Interface*** (**API**).
- Lire la valeur du contexte lexical.
  Via l'appel à ***ValeurLexicale***. La valeur est encodée selon la convention définie par l'énuméré ***EnuUpsAnaValeurLexicale*** :
  - **VL_SymboleCourant**.
    Il s'agit d'un symbole dont le libellé est le champ ***SymboleCourant*** du type ***ValeurLexicale***.
  - **VL_Chaine**.
    Il s'agit d'un symbole dont le libellé est le champ ***Chaine*** du type ***ValeurLexicale***.

    La chaîne de caractères est allouée dynamiquement par ***Up ! Analyzer***. Si elle est récupérée par l'appelant, alors le champ ***Chaine*** doit être affecté à ***NULL***.
  - **VL_Entier**.
    Il s'agit d'un symbole dont le libellé est le champ ***Entier*** du type ***ValeurLexicale***.
  - **VL_Reel**.
    Il s'agit d'un symbole dont le libellé est le champ ***Reel*** du type ***ValeurLexicale***.
- Lire directement dans le flux en contournant l'analyseur lexical.
  Via l'appel à ***LireCaractere***.
- Rejeter un caractère directement dans le flux en contournant l'analyseur lexical.
  Via l'appel à ***RejeterCaractere***.

M

- Sélectionner ou non un paquet lexical.
  Via l'appel à ***SelectionnerPaquetLexical***.
- Chercher un ***token*** correspondant à un modèle.
  Via l'appel à ***ChercherTokenParLibelle***.
- Chercher un ***token*** correspondant à un modèle.
  Via l'appel à ***SupprimerTokenParLibelle***.

5.3 Analyse syntaxique

Cette section décrit comment les analyseurs syntaxiques sont gérés par ***Up ! Analyzer***.

Les différentes alternatives possibles sont les suivantes :

- Utiliser un analyseur syntaxique externe.
  Par exemple, ***Yacc***.
- Utiliser un analyseur syntaxique propriétaire.

La seconde alternative a été retenue pour la raison suivante :

- Besoin de plusieurs analyseurs en simultané.
  Le code en langage ***C*** produit par ***Yacc*** ne le permet pas. Il faut le transformer manuellement.
- Besoin d'analyseurs dynamiques.
  Pour ***Up ! 5GL*** qui est extensible et auto-défini. Le code en langage ***C*** produit par ***Yacc*** ne le permet pas.
- Fonctionnement en multi-thread.
  Le code en langage ***C*** produit par ***Yacc*** ne le permet pas.
- Portabilité complète.
  Le code en langage ***C*** produit par ***Yacc*** l'est difficilement puisqu'il fait des calculs d'adressage.

  Compte-tenu de ces raisons, il n'est pas possible de changer de choix.

5.3.1 Règles de grammaire

&
&

Un analyseur syntaxique reconnaît un enchaînement de ***tokens*** qui peut être récurrent. Une partie linéaire de cet enchaînement est déclarée sous forme d'une **règle d'analyse**. L'ensemble des règles d'analyse forme la **grammaire**.

- Une règle d'analyse est composée d'une succession de :
  - Terminaux.
    Il s'agit soit de :
    - Un caractère, une constante entière, une constante réelle, une constante chaîne de caractères.
      Ils font partie du flux en tant que tel. Ils sont écrits en **vert et gras** dans l'exemple ci-après.
      Quand il ne s'agit pas d'un simple caractère, le terminal est identifié par un ***token*** défini par ***Up ! Analyzer***.
    - Un champ contenant une information échangée.
      L'information fait partie du flux et elle est reconnue parce qu'elle correspond à un modèle identifié par un ***token*** défini par l'usager d'***Up ! Analyzer***. Elle est écrite en ***marron, gras et italique*** dans l'exemple ci-après.
  - Non-terminaux.
    Ils se dérivent en une ou plusieurs règles. Ils sont écrits en *italique* dans l'exemple ci-après.

- Deux règles dérivant un même non terminal sont séparées par le caractère **tuyau |** et la dernière règle se termine par le caractère **point-virgule ;** dans l'exemple ci-après.
  Cette convention est non applicable pour ***Up ! Analyzer***.

Voici un exemple de grammaire pour analyser le flux de l'échange de données ***Chargement des contacts*** :

DebutFlux :

ListeDEnregistrements

;

ListeDEnregistrements :

Enregistrement

| *ListeDEnregistrements Enregistrement*

;

Enregistrement :

" Nom " , " Prénom " , Téléphone , " e-mail " , " Raison sociale " , Siren , " Batiment " , Numéro , " Rue " , Code postal , " Ville " \n

;

Texte 1 – Exemple de syntaxe d'un flux d'un l'échange de données

Voici comment se lit cet exemple :

- Le flux est composé d'une liste d'enregistrements.
- Une liste d'enregistrements est soit :
  - Un enregistrement.
  - Une liste d'enregistrements suivie d'un enregistrement.
- Un enregistrement commence par le champ ***Nom*** écrit entre caractères **guillemet "**, suivi par le champ ***Prénom*** écrit entre caractères **guillemet "**, etc., suivi par le champ ***Ville*** écrit entre caractères **guillemet "** et suivi par un saut de ligne.

Texte 2 – Lecture de l'exemple de syntaxe d'un flux d'un l'échange de données

Une règle peut comporter :

- Des paramètres – Comme un appel de procédure ou de fonction.
  La taille de la zone des paramètres est définie par ***TailleValeurParametre***. Elle peut être agrandie par l'appel à l'***Application Program Interface*** (**API**) avant de débuter l'analyse, par l'appel à l'***Application Program Interface*** (**API**) ***ModifierTailleParametreEtape***.
- Un résultat – Comme une fonction.
  La taille de la zone du résultat est définie par ***TailleValeurEtape***. Elle ne peut pas être agrandie.

5.3.2 Gestion des analyseurs syntaxiques

Un nouvel analyseur syntaxique est créé par l'appel à l'***Application Program Interface*** (**API**) ***AjouterAnalyseurSyntaxique***. Un analyseur lexical existant est détruit par l'appel à l'***Application Program Interface*** (**API**) ***SupprimerAnalyseurSyntaxique***.

Une fois l'analyseur créé, il est nécessaire de déclarer tous les non-terminaux par des appels successifs à ***Application Program Interface*** (**API**)***AjouterNonTerminal***.

Voici les ***Application Program Interface*** (**API**) pour gérer les règles de grammaire :

- ***AjouterRegle***.
  Ajoute une nouvelle règle syntaxique. Elle peut être masquée ultérieurement par l'appel à ***ChangerEtatRegle***.
- ***AjouterEtapeRegle***.
  Ajoute une nouvelle étape à la règle à la suite de l'étape précédente. A défaut, il s'agit de la première étape.
- ***ChangerPrioriteRegle***.
  Spécifique le niveau de priorité de la règle pour résoudre les ambiguïtés. Plus la valeur de la priorité est élevée, plus la règle est prioritaire.

5.3.3 Déclenchement de l'analyse

Le début de l'analyse via un analyseur particulier s'effectue par l'appel à l'***Application Program Interface*** (**API**) ***AnalyserSyntaxiquement***. Elle se termine automatiquement, soit parce que le flux a été correctement analysé ou soit parce qu'il y a eu une erreur de syntaxe fatale correspondant à **AC_ArreterTout** ou à **AC_ArreterToutSansErreur**.

En cours d'analyse, dans une fonction de rappel associée aux étapes de la règle, il est possible de :

- Lire la valeur lexicale correspondant à une étape.
  Via l'appel à ***LireZoneLexicale*** ou via l'appel à ***LireZoneValeur***.
- Lire la valeur des paramètres d'une étape.
  Via l'appel à ***LireZoneParametre***.
- Lire la zone de la valeur de résultat d'une étape.
  Via l'appel à ***LireZoneResultat***.
- Lire le numéro de ligne et de caractère courant.
  Via l'appel à ***LireLigneColonneSource***.
- Lire les commentaires.
  Via l'appel à ***LireCommentaire***.
- Envoyer une erreur de sémantique.
  Via l'appel à ***EnvoyerErreur***.

Suite à l'analyse, il est possible de connaître le nombre d'erreurs de syntaxique détectées par l'appel à l'***Application Program Interface*** (**API**) ***NbErreursDeSyntaxe***.

6 Modèle de données

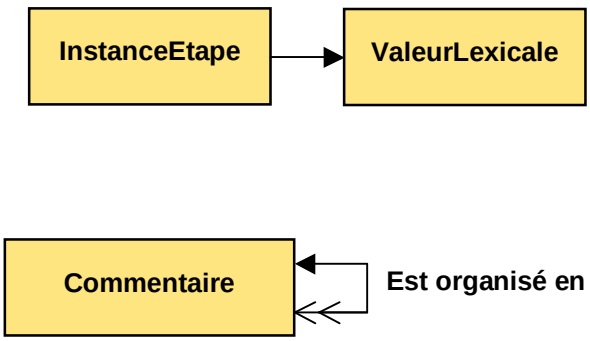

Diagramme 3 – Modèle physique des données publiques du module *Up ! Analyzer*

M Le modèle de données utilise des allocateurs et des tables, à des fins d'optimisation, qui ne sont pas représentés sur le diagramme.

Toutes les entités suivantes sont décrites dans le fichier **uspana.e** :

Entité.	Description.
Commentaire.	Commentaire sur un élément lexical.
InstanceEtape.	Instance d'une étape d'analyse syntaxique.
ValeurLexicale.	Valeur de l'élément lexical reconnu à une étape d'analyse sémantique.

Tableau 4 – Glossaire du modèle physique des données publiques du module *Up ! Analyzer*

7 Composants techniques

Le module ***Up ! Analyzer*** pour le projet ***Up ! Application System*** est constitué des composants suivants :

Fichiers du module.	
• Fichier **uspana.e** – Définition de ***Up ! Analyzer***. • Fichier **uspana.h** – En-tête privé de ***Up ! Analyzer***. • Fichier **uspana.def** – Exportation des symboles pour ***Windows***.	
Composants.	**uspana0**.
Description.	
Interface entre ***Up ! Analyzer*** et ***Up ! Module***.	
Fichiers.	
• Fichier **uspana0.cpp** – Fichier source. • Fichier **uspana0.h** – En-tête privé de **uspana0.cpp**. • Fichier **uspana0.e** – En-tête protégé de **uspana0.cpp**.	
Composants.	**uspana1**.
Description.	
Moteur d'analyse lexicale générique.	
Fichiers.	
• Fichier **uspana1.cpp** – Fichier source. • Fichier **uspana1.h** – En-tête privé de **uspana1.cpp**. • Fichier **uspana1.e** – En-tête protégé de **uspana1.cpp**.	
Composants.	**uspana2**.
Description.	
Moteur d'analyse syntaxique générique.	
Fichiers.	
• Fichier **uspana2.cpp** – Fichier source. • Fichier **uspana2.h** – En-tête privé de **uspana2.cpp**. • Fichier **uspana2.e** – En-tête protégé de **uspana2.cpp**.	
Composants.	**uspana99**.
Description.	
Interface entre ***Up ! Analyzer*** et ***Up ! Kernel***.	
Fichiers.	
• Fichier **uspana99.cpp** – Fichier source. • Fichier **uspana99.h** – En-tête privé de **uspana99.cpp**. • Fichier **uspana99.e** – En-tête protégé de **uspana99.cpp**.	

Tableau 5 – Composants techniques du module

Fin de document