MEMO INTERNE

Version/Date :	**MSTE v1.01 : Rév. 6 en date du 31 janvier 2013**
Auteurs :	**H. MALAINGRE, JM. BERTHEAS**
Destinataires :	**H. NAUGUET**, **C. SIMON, G. LOOS**, **T. STEHLE, M. GIRAUD-GASNIER**,

Mise en place d'un encoder/decodeur objet trans-langage

Contexte et buts

Le développement de PlanitecWeb ainsi que des logiciels mobiles a fait apparaître un besoin important et complexe en termes d'échanges optimisés de grappes d'objets entre différents logiciels et web-services.

L'analyse des besoins immédiats fait transparaître un besoin d'encodage/décodage depuis et vers des des modules écrits en JavaScript (client ExtJS), Objective-C (MASH, XNet et LSMobile iOS), Java (LSMobile Androïd) et WinDev (Municipol).

Les moyens de sérialisation actuellement utilisés sont le JSON, l'Apple P-List (ancien format) ainsi que le XML. Puisqu'aucun de ces moyens de sérialisation ne permet nativement d'encoder des grappes cycliques d'objets et que le types d'objets transmis n'est pas assez précis, il avait été envisagé d'utiliser le format d'encodage HESSIAN.

HESSIAN est un format binaire, assez condensé, permettant l'échange de données structurées et prenant en charge les graphes cycliques d'objets transmis. Malheuresement Hessian n'existe ni en JavaScript (qui ne connaît pas les formats d'échange binaires) ni en WinDev. L'encodage binaire confère à l'encodage et au décodage en JavaScript d'HESSIAN une difficulté de réalisation importante avec, à la clé la perte du côté condensé des données HESSIAN qui devraient de toutes les façon être transmises en Base64 (ou autre moyen de codage ASCII) dans des requêtes de type AJAX.

Pour toutes ces raisons, et pour pouvoir maîtriser l'évolution à moyen et court terme de échanges inter-applications, il a été décidé de créer un nouvel encoder/decoder faisant partie du package open-source MicroStep fonctionnant en **Objective-C**, **JavaScript**, **WinDev**, **Delphi** et **Java** avec les contraintes suivantes :

- encodage dans une string (et non pas de buffer binaire) avec possibilité de transport sur base de 7 bits par octets sans uuencodage (utilisation exclusives de caractères ASCII 7 bits) ;
- encodage sous formes de tokens chaînes et/ou nombres en format array de type JSON ;
- encodage des classes sources avec possibilité pour le destinataire, s'il ne possède pas ces classes, de récupérer les objets sous forme de dictionnaire ;
- encodage natif de nombreux types systèmes (dits natifs) permettant une grande richesse sémantique des données transmises
- garantie d'une évolutivité maximale et simple
- non-redondance des données transmises avec la possibilité de gérer de manière unique les clefs de dictionnaires transmises

Description du format

Chaîne de caractère de support

Le support de codage est une chaîne de caractères codable en 7 bits et représentant un array JSON contenant des chaînes de caractères JSON et des nombres.

Exemple :

```
["MSTE0101",18,"CRCF3452BCD",2,"point","geopoint",3,"x","y","z",7,45,6,8,12,1,"ONE",2]
```

Chaque chaîne de caractères correspondant à un encodage commence donc par un `[` et se termine par un `]`. les tokens sont séparés par des `,` et les tokens étant des nombres et des chaînes JSON, ils doivent répondre aux spécification du JSON avec une contrainte en plus qui est qu'aucun caractère d'une chaîne ne doit être inférieur à 0x20 ou dépasser la valeur 0x7f.

Cela implique que tous les caractères des token-chaînes inférieurs à 32 (espace) ou supérieurs à 127 seront escapés, soit de manière spécifique conformément au format JSON (`\"`, `\\`, `\/`, `\b`, `\f`, `\n`, `\r`, `\t`) soit au format unicode `\uXXXX` ou `XXXX` est la valeur héxadécimale UTF16 d'un caractère.

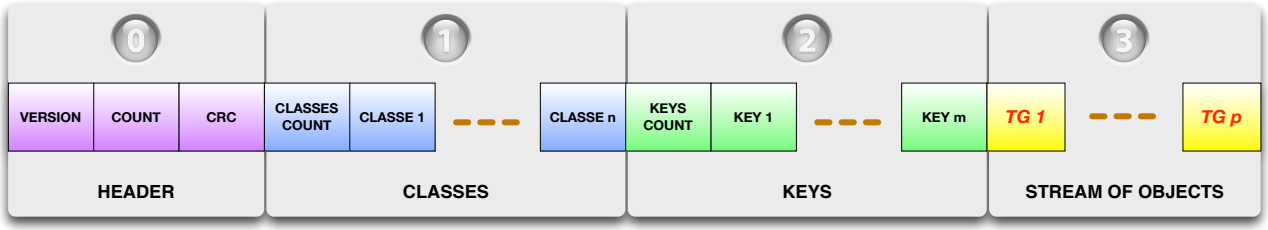
Comme dans l'exemple, il est dans la mesure du possible souhaité de limiter les blancs inutiles avant et après les virgules séparant les tokens (attention l'exemple n'est pas juste pour ce qui est de la partie stream d'objets).

Organisation des tokens

Un encodage est une suite de tokens (des nombres ou des chaînes de caractères) qui commence toujours de la même façon et dans cet ordre :

- une chaîne de 8 caractères de long comprenant les quatre lettres MSTE suivi du code hexadécimal de la version et de la sous version de l'encodage (un octet chacun). Exemple : `"MSTE0101"`;
- le nombre de tokens total (y compris la chaîne précédente). Exemple : `18` ;
- une chaîne de caractères de 11 caractères comprenant les 3 lettres CRC suivi, si il a été calculé, du CRC32 en hexadécimal (calculé avec la chaîne "CRC00000000" en guise de CRC initial) de l'ensemble de la chaîne support. Exemple : `"CRCF3452BCD"` ;
- le nombre de classes spécifiques encodées dans la grappe (toutes les classes qui ne sont pas des objets systèmes connues par l'encodeur). Exemple : `2`. Si le nombre est 0, aucune classe spécifique n'est définie.
- autant de chaînes de caractères qu'il y a de classes spécifiques encodées. Chaque chaîne de caractères est un nom ou un code représentatif de la classe en question. Exemple : `"point","geopoint"`.
- le nombre de clefs de dictionnaires utilisées lors de l'encodage: Exemple : `3`. Si le nombre est 0, aucune clé n'est définie.
- autant de chaînes de caractères qu'il y a de clefs de dictionnaires. Exemple : `"x","y","z"`.

Le token qui suit est le code de l'objet racine de la grappe d'objet encodée. Tous les tokens qui suivent jusqu'à épuisement des tokens décrivent la grappe encodée. Le schéma suivant résume l'organisation des tokens :



Stream des objets et des valeurs

Le stream des objets correspond à la partie ③ du schéma précédent.

C'est une suite de token organisé en séquences. Chaque séquence commence avec un numéro qui défit le type d'information stockée puis une série de tokens adapté à l'information en question. Par exemple **2** est le code d'un nombre entier signé et le code suivant est le nombre en lui-même.

Le tableau suivant présente la liste des codes de séquences acceptées avec les token attendus derrière chaque code ainsi que l'indication si l'objet qu'il représente peut être référencé ou non.

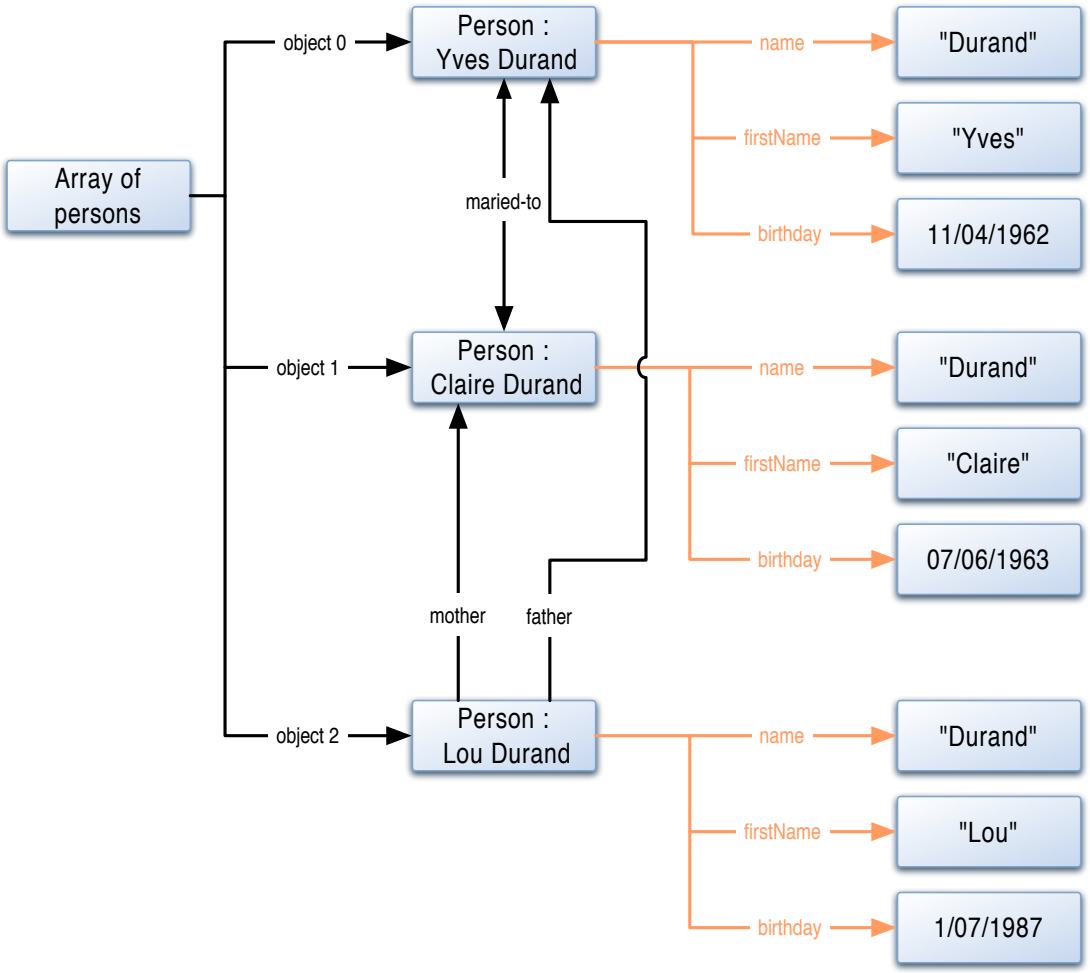
Type	**Code**	**Suite attendue**	**Ref ?**	**Retenu**
Objet null (ou nil)	0	Aucune	Non	N/A
Valeur "vraie" (true)	1	Aucune	Non	N/A
Valeur "fausse" (false)	2	Aucune	Non	N/A
Valeur entière	3	1 seul token contenant un nombre entier a / a ∈ ℤ, sans limite de stockage	Oui	Oui
Valeur réelle	4	1 seul token contenant un nombre réel r / r ∈ ℝ, sans limite de stockage	Oui	Oui
Chaîne de caractères	5	1 seul token comprenant la chaîne de caractères entre double-quotes	Oui	Oui
Date	6	1 seul token contenant le nombre entier (positif ou négatif) de seconde écoulées ou à venir depuis ou vers le **01/01/1970** qui est la référence (**Unix Epoch**).	Oui	Oui
Couleur	7	1 seul token (nombre entier positif ou nul) contenant la représentation en 24 ou 32 bits de la couleur. 24 bits = RRGGBB. 32 bits = TTRRGGBB ou TT est la transparence (0x00 = opaque, 0xFF = totalement transparent)	Oui	Oui
Dictionnaire	8	1 token contenant (nombre entier positif ou nul) le nombre de couples clé-valeur) du dictionnaire. Suivent ensuite toutes les séquences clé-valeurs du dictionnaire. A noter que les clés du dictionnaire sont forcément des chaînes de caractères	Oui	Oui
Référence **FORTE** vers un objet déjà encodé/décodé	9	1 seul token (nombre entier positif ou nul) contenant l'index de la chaîne-clé dans le tableau des objets déjà décodés. L'objet peut-être un dictionnaire, une chaîne de caractères, un tableau, un tableau d'entiers naturels, un couple, une data, une couleur, une date, une valeur entière/réelle.	N/A	**Oui**
Char	10	1 seul token : le nombre compris entre -128 et +127	Non	N/A
Unsigned char	11	1 seul token : le nombre compris entre 0 et 255	Non	N/A
Short	12	1 seul token : le nombre compris entre -32768 et +32767	Non	N/A
Unsigned short	13	1 seul token : le nombre compris entre 0 et 65535	Non	N/A
Int 32	14	1 seul token : le nombre compris entre -(2^31) et (2^31)-1	Non	N/A
Unsigned int 32	15	1 seul token : le nombre compris entre 0 et (2^32)-1	Non	N/A
Int 64	16	1 seul token : le nombre compris entre -(2^63) et (2^63)-1	Non	N/A
Unsigned int 64	17	1 seul token : le nombre compris entre 0 et (2^64)-1	Non	N/A
Valeur flotante	18	1 seul token comprenant un nombre flottant interprétable en simple précision	Non	N/A
Valeur double	19	1 seul token comprenant un nombre flottant interprétable en double précision	Non	N/A

Type	Code	Suite attendue	Ref ?	Retenu
Array	20	1 token contenant (nombre entier positif ou nul) le nombre d'éléments du tableau. Suivent ensuite toutes les séquences correspondants aux éléments du tableau.	Oui	Oui
Natural array	21	N+1 tokens où N est le nombre d'entiers naturels positif sur 32 bits contenus dans le tableau. Le premier token est le count du tableau, les N suivants sont les nombres en question (sans code préalable par nombre, juste les nombres)	Oui	Oui
Couple	22	2 séquences de tokens correspondant au premier et au second membre du couple d'objets.	Oui	Oui
Data (base 64)	23	2 séquences de tokens correspondant à la longueur originale en octets de la donnée transmise, suivi d'un token comprenant une chaîne de caractères entre double-quotes et encodée en Base64.	Oui	Oui
Distant past	24	Date infinie dans le passé	Non	N/A
Distant future	25	Date infinie dans le futur	Non	N/A
Chaîne de caractères vide	26	Aucune	Non	N/A
Référence **FAIBLE** vers un objet déjà encodé/décodé	27	1 seul token (nombre entier positif ou nul) contenant l'index de la chaîne-clé dans le tableau des objets déjà décodés. **L'objet ne peut-être qu'un objet de classe «user class». (Gestion des références cycliques / libération mémoire en fonction du langage utilisé).**	N/A	**Non**
Objet de classe spécifique **retenu** (user class)	50+2.n	1 token contenant (nombre entier positif ou nul) le nombre de couples clé-valeur du dictionnaire représentant les membres de l'objet. Suivent ensuite toutes les séquences clé-valeurs du dictionnaire des membres. A noter que les clés du dictionnaire sont forcément des chaînes de caractères. L'indice n correspond à l'index de la classe spécifique listée dans la section ①	Oui	**Oui**
Objet de classe spécifique **non retenu** (user class)	51+2.n	1 token contenant (nombre entier positif ou nul) le nombre de couples clé-valeur du dictionnaire représentant les membres de l'objet. Suivent ensuite toutes les séquences clé-valeurs du dictionnaire des membres. A noter que les clés du dictionnaire sont forcément des chaînes de caractères. L'indice n correspond à l'index de la classe spécifique listée dans la section ①	Oui	**Non**

Tous les codes de 0 à 49 sont considérés comme des codes systèmes (pour l'instant, seuls ceux de 0 à 27 sont utilisés) et à partir de 50 et au-delà, les codes N correspondent à des objets "utilisateurs" dont les classes sont décrites dans la section CLASSES de l'en-tête et dont le classe se retrouve grâce à l'index = partie entière de (N-50)/2.

Exemple

Prenons comme grappe d'objets à encoder le système suivant décrit sur le schéma suivant :



Pour cet exemple, nous prenons comme hypothèse que la chaîne de caractères "Durand" utilisée dans les trois objets de classe Person est la même (a physiquement la même empreinte mémoire).

Après encodage, nous aurons la chaîne de caractères suivante :

```
["MSTE0101",59,"CRCC41DBEF3",1,"Person",6,"name","firstName","birthday","maried-to","father","mother",20,3,50,4,0,5,"Durand",1,5,"Yves",2,6,-1222131600,3,51,4,0,9,2,1,5,"Claire",2,6,-1185667200,3,27,1,9,5,50,5,0,9,2,1,5,"Lou",2,6,-426214800,4,9,1,5,9,5]
```

Décomposons les tokens de cette chaîne :

séquence	Découpage	Description	Objets référencés

`"MSTE0101"`	`MSTE = magic word`
	`01 = version (hexa)`
	`01 = release (hexa)`
		Token de versionning	–

`59`	`59 = nombre total de tokens`	Nombre de contrôle	–
`"CRCC41DBEF3"`	`CRC = magic word CRC` `C41DBEF3 = CRC32 en hexadécimal`	CRC de contrôle d'intégrité de la chaîne complète	–
`1,"Person"`	`1 = nombre de classe` `"Person" = une classe user`	La liste de classe utilsateur est ici réduite à un élément unique	–

`6,"name","firstName", "birthday","maried-to","father","mother"`	`6 = nombre de clés`
	`"name" = clé 0`
	`"firstName" = clé 1`
	`"birthday" = clé 2`
	`"maried-to" = clé 3`
	`"father" = clé 4`
	`"mother" = clé 5`
		L'ensemble des clés utilisées dans les dictionnaires ou les classes utilisateur	–

`20,3`	`20 = code "tableau"` `3 = count du tableau racine`	L'objet racine du stream encodé est un tableau contenant trois objets.	0: root array
`50,4`	`50 = 1er objet du tableau (de la classe Person)` `4 = nb de couples clé/valeur`	Début du stream du 1er objet Person du tableau	1: Person Yves Durand

`0,5,"Durand"`	`1er couple clé/valeur`
	`0 = clé 0 = name`
	`5 = objet string`
	`"Durand" = valeur`
		name = "Durand"	2: string "Durand"

`1,5,"Yves"`	`2e couple clé/valeur`
	`1 = clé 1 = firstName`
	`5 = objet string`
	`"Yves" = valeur`
		firstName = "Yves"	3: string "Yves"

`2,6,-1222131600`	`3e couple clé/valeur`
	`2 = clé 2 = birthday`
	`6 = objet date`
	`-1222131600 = valeur`
		birthday = "11/04/1962"	4: date 11/04/1962

`3,51,4`	`4e couple clé/valeur`
	`3 = clé 3 = maried-to`
	`50 = un nouvel objet Person`
	`4 = nb de couples clé/valeur`
		maried-to = un objet personne qui va être immédiatement défini dans la suite du stream (non retenu)	5: Person Claire Durand

`0,9,2`	`1er couple clé/valeur du nouvel objet`
	`0 = clé 0 = name`
	`9 = référence à un objet connu`
	`2 = référence à "Durand"`
		name = "Durand"	–

`1,5,"Claire"`	`2e couple clé/valeur du nouvel objet`
	`1 = clé 1 = firstName`
	`5 = objet string`
	`"Claire" = valeur`
		firstName = "Claire"	6: string "Claire"

`2,6,-1185667200`	`3e couple clé/valeur du nouvel objet`
	`2 = clé 2 = birthday`
	`6 = objet date`
	`-1185667200 = valeur`
		birthday = "07/06/1963"	7: date 07/06/1963

`3,27,1`	`4e couple clé/valeur du nouvel objet`
	`3 = clé 3 = maried-to`
	`9 = référence`
	`1 = index objet de référence`
		maried-to = référence FAIBLE vers l'objet Person "Yves Durand"	–

9,5	2e objet du tableau racine 9 = référence 5 = index objet de référence	le 2e objet du tableau racine est l'objet Person "Claire Durand" déjà définie dans l'encodage de la variable d'instance "maried-to" du 1er objet du tableau	-
50,5	50 = 3e objet du tableau racine (de la classe Person) 5 = nb de couples clé/valeur	le 3e objet du tableau racine est une personne nouvelle qui va être définie juste après	8: Person Lou Durand
0,9,2	1er couple clé/valeur 0 = clé 0 = name 9 = référence à un objet connu 2 = index de "Durand"	name = "Durand"	–
1,5,"Lou"	2e couple clé/valeur 1 = clé 1 = firstName 5 = objet string "Lou" = valeur	firstName = "Lou"	9: string "Lou"
2,6,-426214800	3e couple clé/valeur 2 = clé 2 = birthday 6 = objet date -426214800 = valeur	birthday = "01/07/1987"	10: date 01/07/1987
4,9,1	4e couple clé/valeur 4 = clé 4 = father 9 = référence 1 = index objet de référence	father = référence vers l'objet Person "Yves Durand"	-
5,9,5	5e couple clé/valeur 5 = clé 5 = mother 9 = référence 1 = index objet de référence	mother = référence vers l'objet Person "Claire Durand"	-

Contrôle de l'encodage et du décodage en Javascript

Expliquer pour le JS la notion de whiteList et de blackList des clefs et les restriction sur les clés ainsi que le pb du null non encodé dans les dictionnaires.