

**TGV**Allocate new space

**Table of Contents**

[1 Document Information 3](#_Toc363458225)

[1.1 Document History 3](#_Toc363458226)

[1.2 Attached Files 3](#_Toc363458227)

[1.3 Caption 3](#_Toc363458228)

# Document Information

## Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description of modifications** | **Name** | **Date** |
| 1.0 | Document creating | F Archambault | 08/05/2013 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Attached Files

|  |  |
| --- | --- |
| **Name** | **Description** |
|  |  |
|  |  |

## Caption

|  |  |
| --- | --- |
| **Image** | **Description** |
| C:\Users\delacourtie_c\Pictures\Microsoft Clip Organizer\j0431579.png | Desktop entry point to access the function |
| C:\Users\delacourtie_c\Pictures\Microsoft Clip Organizer\j0434828.png | Initial screen process |
| C:\Users\takeda_e\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\PHD22P7R\MCj04348050000[1].png | Allows defining a warning. |
| C:\Users\takeda_e\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\7G35DKB2\MCj04326170000[1].png | Set a tip. |
| [**Xxxxx**] | Designate un bouton |
|  | Example |

# Introduction

 TGV cannot shrink, only grow.

# Mecanisms

## What is put in TGV when saving a class?

When saving a class (i.e a class’s version), at least these objects are saved:

aClassdef

aClassImplem

aMethodDesc

aMethodImplem

aMethodType

aParamDesc

MotorStatus

Some Indexes

Some lists

(And more depending of the number of variables/methods in the class)

A class’s version takes at least 1600o.

1060o for data

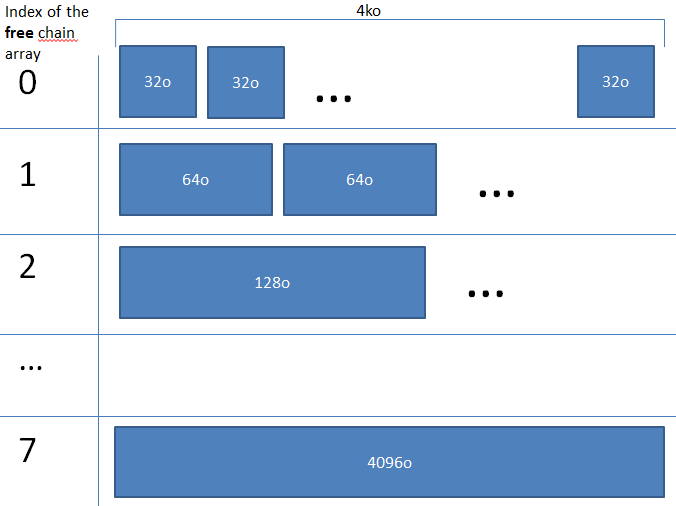
500o for index (especially NameAccessPlan)

## How it is put in TGV?

At each time that an object needs to be saved into TGV, it calls *NewSpace* (WTGV.CPP) method to allocate memory in TGV.

This method uses an array of pages (4ko each page) (called *FreeChains*) where each frame is decomposed into many homogeneous blocks like below:

Schema of *FreeChains* array



*Each block has a pointer (Header) which indicates:*

* *A pointer to the next block*
* *The size of the block (in number of Chunck (32ko))*
* *Some information by Booleans (busy,…)*

### Algorithm of NewSpace(Int4 thisSize)

Check if the TGV is not locked

Get the number of chunck corresponding to *thisSize*.

Get the index of *FreeChains* via *P2Over(Number of chuncks)*.

Get the first block of *FreeChains*.

Check if the block is free (otherwise allocate a new page of 4ko for this index).

Set the first block of FreeChains by the next block (*Header.Next*).

Return the block collected.

# Optimizations

|  |  |  |
| --- | --- | --- |
|  |  | Estimated gain |
| In **wInterface**, do not generate and store serialization classes when not necessary | In progress : 5jh | On Wynsure 5.1 Business Service: Gain 20Mo to 2.5Mo  Gain: /10  First time: 20 Mo  15 minutes to 17 minutes: slight loss of performance but it’s insignificant  2nd time :  2 minutes : major gain: /10 |
| In eWam : **indexes à supprimer** certain objet du metamodel dont on a pas besoin de l’AP Name :   * Local var desc * Method type * Parameder desc (on a dans MMBroser mais rarement utilisé) * Accesplan : on en a besoin dans MMBrowser et en plus gain faible | Assez facile  Faire l’etude (3jh) | Plus il y a du code dans les methodes.  A estimer en faisant un test en désactivant les acces plan sur ces classes et en generant des vrais classes et pas des classes vides.  3Jh d’etudes  (aTGVDBMgr::StoreAccessPlansFor(aFullObject & thisObject) |
| **Why 15 minutes**? | 3jh | Etude :   * Compiler wInterface 4 et mesurer le temps compilé * AMD Analyst : pie chart du temps passé :   + Generation de l’IR WSDL   + Parsing de classes   + Stockage TGV   + … * **Z:\Wyde Development\wInterface\Management\Internal\DistributionOfTimeDuringGeneration.xlsx** |
| Mettre un **Versionning** de 1  Comme cela est fait pour le DBDef | Facile. Mais risques à lever… | Versionning de 1 sur les classes avant de generer les classes de serialization.  Le stockage de la N+1 va liberer l’espace de N. Et donc la N+2 va réutiliser l’espace de N.  (Sinon , c’est un bug.)  Le restaurer après la generation.  A etudier :   * Gain taille TGV * Perte de perf a cause du Zap * Bonne gestion au niveau du In/out multi user * Bonne gestion au niveau des bundles (generation/installation. Test des upgrades) |
| Augementer **l’eventail de taille de cellules** possibles :  8 tailles de cellule : on est statistiquement à 25 % de blanc :  Surtout sur les tailles de debuts  32o  64o  96o  128o  …  On peut pas diminuer les 32o (relatif à la taille max du TGV)  Avoir 128 niveau : cout pour les petit TGV : ils vont etre plus gros car 128\*4k au lieu de 8 \* 4ko.  Ex de petit TGV : In MemTGV : (multi team + wReplicate/wExtract).  L’idee est plutôt d’introduire de nous niveaux.  Attention à la compatibilité du TGV. | At least 10jh | Gain: on joue sur les 25% d’espace libre.  La borne theorique superieur: si 128 niveaux : on gagne 70% des 25% = 16% |
| Compresser plus les objets stockés : **compression de type Zip**  Quand on zip le TGV : 1Go à 200Mo :   * 25% de 0 des bloc * Les 0 du name index * Compression autre que 0   Aujourd’hui :   * on coupe les 0 des string * on compresse les text   Voir impact des perf du zip : compromis | Complexité moyenne |  |
| **Reutiliser les blancs** !  Le danger : quand on libere on fragment de plus en plus le TGV   * A chaque ecriture et chaque suppression Perte de temps a decouper et mettre a jour les freechain (acces disk) * Fragmentation :sur un TGV où il n’y a que des suppression et ecriture il ne reste plus que des bloques de 32. | Jh elevé. Risque élevé. |  |
| **Zero Compress** des index | A premiere vue :compliqué |  |