

**How to Modify the EGL File Location  
Algorithm used by Stage 1 of the  
VisualAge Generator on Smalltalk to  
Enterprise Generation Language  
Migration Tool**

**By: Larry Hamann**

# Contents

Introduction.....	3
What is the current file placement algorithm?.....	3
Why do I want to change the file placement algorithm?.....	3
Where in the Smalltalk code is the file name determined?.....	3
An Example of changing the algorithm.....	4
Summary.....	6
Appendix. How do I use VisualAge Smalltalk to change the algorithm?.....	7

## Introduction

This paper is intended for customers who are migrating from VisualAge Generator (VAGen) on Smalltalk to Enterprise Generation Language (EGL) and are interested in customizing the algorithm that determines the file placement of the VAGen parts in the EGL workspace. It discusses what Smalltalk class needs to be changed, what methods need to be added, what methods need to be modified, and how the changes work. In addition, this paper provides a sample of the modified Smalltalk script that you can use as a guide to help you understand and create your own modifications.

## What is the current file placement algorithm?

During the migration from VAGen to EGL the location of where a part is placed in the EGL workspace must be determined. Stage 1 migration determines file placement based on the data specified in the Stage 1 migration preference file and the VAGen part type. For example, the default placement for a part that is determined to be common is in a file called *commonparts.egl* where *commonparts* is specified by the Stage 1 preferences. Unused or non-referenced parts are placed in a file called *unusedparts.egl* where *unusedparts* is specified by the Stage 1 preferences. A complete description of the file placement algorithm for each part type is described in the 'Placing parts in EGL files' section of the 'VisualAge Generator to EGL Migration Guide'.

## Why do I want to change the file placement algorithm?

There are a number of reasons you may want to change the algorithm that determines the location of a part being migrated. One big reason may be the size of the file. For a big VAGen application the number of parts placed into a particular file can get quite large. Large EGL files can be hard to work with because of the time it takes to parse, search, and compile them. Another reason may be the criteria you use for determining what parts are grouped together in a file. Record parts may need to be managed by your database department while a different department manages the business logic. Because your EGL files are usually managed by a library system, controls can be put into place that determine who can modify which parts. Having many part types in one file could limit the granularity needed for this type of part management.

## Where in the Smalltalk code is the file name determined?

The algorithm for determining what file name a part is assigned is located in the Smalltalk application *HptEglMigrationBaseApp*. The instance methods that determine the part type are grouped together under the Instance Category 'EGL File Paths' in class *VAGenToEGLMapper*. Table 1 lists the methods in this group and the part types for which they determine the EGL file location.

**Table 1. Methods in the EGL File Paths Category**

<b><u>Method Name</u></b>	<b><u>Description</u></b>
<b><i>eglFilePathForAdditionalPart</i></b>	Non-referenced and unused Parts
<b><i>eglFilePathForCommonParts</i></b>	All parts that are determined to be common
<b><i>eglFilePathForControlPart:</i></b>	Control Parts
<b><i>eglFilePathForTable:</i></b>	Table Parts
<b><i>eglFilePathForUniquePartIn:</i></b>	Unique parts such as Program parts

## **An Example of changing the algorithm**

Having too many parts in one EGL file can be a major problem. Large files are usually more difficult to manage and they take a greater amount of time to parse, compile, and search in the workspace. A typical problem during migration is for the number of common parts or unused parts in a single file to become quite large. The following example shows how the algorithm for the *VAGenToEGLMapper* class can be changed in order to further separate the common parts and unused parts into smaller files.

The default algorithm used by the migration tool places all the common parts in a file specified by the EGL Naming Preferences section of the Stage 1 Preferences file. In the example we want to change this algorithm so the parts are further separated into files based on their part type and the first letter of their part name. Therefore, if the Common Parts file name specified in the Stage 1 Preference file is 'Common' and the part type is 'Data Item', the modified logic places the parts in files with the names, 'CommonDataItemA', 'CommonDataItemB', ..., 'CommonDataItemZ'. The modified logic places data items with names that start with nonalphabetic characters in a file called CommonDataItem. For example, all data items with names starting with \$, #, or @ are placed in a file called CommonDataItem.

The instance method for *VAGenToEGLMapper* that determines the file name for a common part is *eglFilePathForCommonParts* in the *EGL File Paths* category. In order to add logic to assign the file name for the part based on its type and the first letter of its name, the part object must be passed into the method as an argument. In Smalltalk a colon at the end of a keyword in a method signifies the presents of an argument. Therefore a colon must be added to the method *eglFilePathForCommonParts*, which creates a new method named *eglFilePathForCommonParts:*. Figure 1 shows the logic for this method. The comments indicated by "Step *n*" where *n* is a number are explained after the figure.

**Figure 1. eglFilePathForCommonParts:**

```
eglFilePathForCommonParts: aPart
```

```
" Determine the EGL file name for this common part. "
```

```
" Return a file name that places each common part in a file based on its part type and  
the first letter of the its name. "
```

```
| endingFileNameSuffix partFileName |
```

```
" Step 1"
```

```
" Determine the ending file name Suffix "
```

```
( endingFileNameSuffix := ( aPart name at: 1 ) isLetter  
  ifFalse: [ endingFileNameSuffix := " . ]
```

```
" Step 2"
```

```
" Create the part name by combining the different parts of the name together. "
```

```
partFileName := (  
  (HptEglMigrationDriver preferences commonPartsFilename),  
  (aPart typeString),  
  (endingFileNameSuffix asString),  
  (self eglSuffix)
```

```
" Step 3"
```

```
) hptRemoveWhiteSpace.
```

```
" Step 4"
```

```
" Create the full file name for the part by adding the path to the front of the name. "  
^self standardEGLFilePathFor: partFileName.
```

Step 1 of the algorithm determines the ending file suffix. At Step 1 the first character of the part name is obtained. It is checked to see if it is a letter of the alphabet; if not the ending file suffix is set to a null string. This is done because some part names do not start with an alphabetic character.

Step 2 of the algorithm creates the file name by concatenating together the Common Parts file name from the Stage 1 preferences file, the part type, the ending file name suffix determined in Step 1 and the ending '.egl' suffix. (Note: Smalltalk uses the comma as a binary message to concatenate strings together.)

Step 3 of the algorithm removes any white spaces (blanks) in the file name. This is necessary because there is a blank in the "Data Item" part type.

Step 4 adds the path of where the file is to be located and returns the completed file name to the sender.

The changes just described for a common part need to be done for unused or non-referenced parts as well. The new method for this part type is:

**Figure 2. eglFilePathForAdditionalPart:**

```
eglFilePathForAdditionalPart: aPart

    "Create the file name for this non-references part."

    " Return a file name that places each non-references part in a file based on its part type and
    the first letter of the its name. "

    | endingFileNameSuffix partFileName |

    " Determine the ending file name Suffix "
    ( endingFileNameSuffix := ( aPart name at: 1 ) isLetter
      ifFalse: [ endingFileNameSuffix := ". ].

    " Create the part name by combining the different parts of the name together. "
    partFileName := (
      (HptEglMigrationDriver preferences unusedPartsFilename),
      (aPart typeString),
      (endingFileNameSuffix asString),
      (self eglSuffix)
    ) hptRemoveWhiteSpace.

    " Create the full name of the part by adding the path to the front of the name. "
    ^self standardEGLFilePathFor: partFileName.
```

The final method to change is also in the class *VAGenToEGLMapper*. The method is *getEGLPathForPart*: in the *API* category. This method determines the part type and calls the appropriate method to create the file name. Therefore, two lines need to be changed in this method to call the new methods created for a common part and an unused part. The lines to change are:

**Figure 3. Old code to replace in getEGLPathForPart:**

```
" look for it in my categorized caches "
(self sortedParts isCommonPart: aPart) ifTrue: [ ^self eglFilePathForCommonParts. ].
(self sortedParts isAdditionalPart: aPart) ifTrue: [ ^self eglFilePathForAdditionalPart. ].
```

The lines in Figure 3 are modified to call the new methods as shown in Figure 4.

**Figure 4. New code to use in getEGLPathForPart:**

```
" look for it in my categorized caches "
(self sortedParts isCommonPart: aPart) ifTrue: [ ^self eglFilePathForCommonParts: aPart. ].
(self sortedParts isAdditionalPart: aPart) ifTrue: [ ^self eglFilePathForAdditionalPart: aPart. ].
```

## Summary

This paper has shown how to modify the EGL file placement algorithm for VAGen parts. Your requirements may be different from what is described in the paper. Feel free to modify the code based on your particular requirements. The goal is to have the migration tool place converted VAGen parts in EGL files that can be easily managed and understood by your organization.

## Appendix. How do I use VisualAge Smalltalk to change the algorithm?

If you have not worked with Smalltalk before, this appendix provides some details that might be helpful in making the modifications for the migration tool in VisualAge Smalltalk.

To find the methods you need to change, do the following:

- From the VisualAge Organizer window, select **Applications -> View -> Show All Applications**.
- Select **Options -> Change User**. From the Selection Required window, change to be the Library Supervisor.
- In the Applications pane, select *HptEGLMigrationBaseApp*. Select **Applications -> New Edition**. This enables you to make changes to the application.
- In the Parts pane, select *VAGenToEGLMapper*. Select **Parts -> Open -> Class Browser** to see the source code for this class.
- From the *VAGenToEGLMapper* window, toggle the **instance** or **class** button so that it is set to **instance**. Toggle the **public** or **private** button so that it is set to **public**. In the lower right corner of the window, toggle the **source**, **comments**, or **notes** button so that it is set to **source**.
- You should see both the *EGL File Paths* and the *API* category in the upper left pane.

To create a new method, do the following:

- From the *VAGenToEGLMapper* window, in the upper left pane, select the category in which you wish to place the new method. Then select **Method -> New Method Template**.
- Replace the template source code with your new code (for example, the code from Figure 1 or Figure 2).
- Select **File -> Save** to save this new method.

To change an existing method, do the following:

- From the *VAGenToEGLMapper* window, in the upper left pane, select the category in which the method is located. Then select the method in the upper right pane.
- Replace the lines of source code you want to change with the new source code (for example, the code from Figure 4).
- Select **File -> Save** to save your changes.

After you have made the code changes, remember to version application *HptEGLMigrationBaseApp*. Be sure to use a meaningful name because you will need to reference this version in the following situations:

- If you reload the Stage 1 migration tool for any reason, you must reload your version of *HptEGLMigrationBaseApp*. For example, if you scrub your image, you must reload the Stage 1 migration tool and then reload your version of *HptEGLMigrationBaseApp*.
- If you load a new version of the Stage 1 migration tool, you will need to apply your changes to the new version of the migration tool.