

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

**By: Doumit Ishak**

# 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 Java code is the file name determined?.....	3
An Example of changing the algorithm .....	4
Summary .....	8
Appendix. How do I use VisualAge Java to change the algorithm? .....	8

## Introduction

This paper is intended for customers who are migrating from VisualAge Generator (VAGen) on Java 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 Java 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 Java code 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 Java code is the file name determined?

The algorithm for determining what file name is assigned, is located in the “IBM VisualAge Generator EGL Migration” project, under the “com.ibm.vgj.mig” package, in the “VAGenToEGLMapper” java class. Table 1 lists the methods that we are interested in editing to implement the needed changes.

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

<u>Method Name</u>	<u>Description</u>
<i>getEGLPathForPart</i>	Determines the part type and calls appropriate method
<i>getEGLFilePathForCommonParts</i>	All parts that are determined to be common
<i>getEGLFilePathForUnusedParts</i>	All parts that are determined to be unused

## 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 method for *VAGenToEGLMapper* that determines the file name for a common part is *getEGLFilePathForCommonParts*. 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 name and part type must be passed into the method as arguments. Figure 1 shows the original logic for method *getEGLFilePathForCommonParts*.

**Figure 1: Old logic for *getEGLFilePathForCommonParts()***

```
private String getEGLFilePathForCommonParts() {  
    return (standardEGLFilePath(commonFilePref, ".egl"));  
}
```

The *getEGLFilePathForCommonParts* method is modified to fit our need as shown in Figure 2. The original code is shown in blue and the new code is shown in red. Figure 2 contains notes that explain the new logic. The notes are in green and indicated by "Step *n*", where *n* is a number.

**Figure 2: New logic for getEGLFilePathForCommonParts(String, String) method**

```
/** Step 1
 * Edit the method to accept two strings as parameters
 * "partName" : name of the part being migrated
 * "partType" : type of the part being migrated (PSBs, Records, Functions, Data Items)
 */
private String getEGLFilePathForCommonParts(String partName, String partType) {
    // Step 2
    // Get first letter of the part name, and make it upper case
    String firstLetter = partName.substring(0,1).toUpperCase();

    // Step 3
    // In case the part is a "Data Item", delete the space in "Data Item"
    // in order to append it in the new derived name
    if(partType.equals("Data Item")){
        partType = "DataItem";
    }

    // Step 4
    // Create a new empty string to later hold the derived name
    String newName = "";

    /** Step 5
     * Check to see if the first letter of the part name is one of the following: "@, #, $"
     * If it is, append the file preference and the part type only.
     * If it is not, append the file preference, the part type, and the first letter of the
     * part name in the order they were listed.
     */
    if(firstLetter.equals("@")
        || firstLetter.equals("#")
        || firstLetter.equals("$")){
        newName = commonFilePref + partType; // ex: CommonDataItem
    } else{
        newName = commonFilePref + partType + firstLetter;
        // ex: CommonDataItemA
    }

    // Step 6
    // Pass new name to standartEGIFilePath method
    return (standardEGLFilePath(newName, ".egl"));
}
```

The changes just described for a common part need to be done for unused or non-referenced parts as well. Figure 3 shows the original logic for method *getEGLFilePathForUnusedParts*.

**Figure 3: Old logic for getEGLFilePathForUnusedParts()**

```
private String getEGLFilePathForUnusedParts() {
    return (standardEGLFilePath(unusedFilePref, ".egl"));
}
```

Figure 4 shows the modified *getEGLFilePathForUnusedParts()*. The logic is identical to Figure 2 except that the *newName* is created using the file name preference for unused parts rather than the preference for common parts.

**Figure 4: New logic for *getEGLFilePathForUnusedParts(String, String)* method**

```
/** Step 1
 * Edit the method to accept two strings as parameters
 * "partName" : name of the part being migrated
 * "partType" : type of the part being migrated (PSBs, Records, Functions, Data Items)
 */
private String getEGLFilePathForUnusedParts(String partName, String partType) {
    // Step 2
    // Get first letter of the part name, and make it upper case
    String firstLetter = partName.substring(0,1).toUpperCase();

    // Step 3
    // In case the part is a "Data Item", delete the space in "Data Item"
    // in order to append it in the new derived name
    if(partType.equals("Data Item")){
        partType = "DataItem";
    }

    // Step 4
    // Create a new empty string to later hold the derived name
    String newName = "";

    /** Step 5
     * Check to see if the first letter of the part name is one of the following: "@, #, $"
     * If it is, append the file preference and the part type only.
     * If it is not, append the file preference, the part type, and the first letter of the
     * part name in the order they were listed.
     */
    if(firstLetter.equals("@")
        || firstLetter.equals("#")
        || firstLetter.equals("$")){
        newName = unusedFilePref + partType; // ex: UnusedFunction
    } else{
        newName = unusedFilePref + partType + firstLetter;
        // ex: UnusedFunctionD
    }

    // Step 6
    // Pass new name to standartEGIFilePath method
    return (standardEGLFilePath(newName, ".egl"));
}
```

The final method to change is also in the class *VAGenToEGLMapper*. The method is *getEGLPathForPart*. This method determines the part type and calls the appropriate method to create the file name. Therefore, a few 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 shown below in Figure 5. The original code is shown in blue and the new or modified code is shown in red. Figure 5 contains notes that explain the new logic. The notes are in green and indicated by “Step *n*”, where *n* is a number.

**Figure 5: Modified getEGLPathForPart(VAGenPartInformation)**

```
public String getEGLPathForPart(VAGenPartInformation part){
    String defaultPath = "unresolved.egl";
    VAGenPartInformation hit;
    com.ibm.ivj.util.base.Package pkg;

    // Step 1 - Determine the part name and part type
    String partName = part.getName();
    String partType = part.getTypeString();

    try {
        pkg = (com.ibm.ivj.util.base.Package)part.getPackage();
        setWorkingPackage(pkg);
    }
    catch (Exception ex) {
        return ("unresolvedPKG.egl");
    }
    // do the easy guys first ... their part type determines the path "
    if ( isSelfDeterminatePart(part) ) {
        if (isControlPartType(part.getType()))
            return (getEGLFilePathForControlPart(part));
        else
            return (getEGLFilePathForUniquePart(part));
    }

    if (sortedContext.hasCommonPart(part) )
        // Step 2 - Pass the part name and type in the parameters of the function
        return(getEGLFilePathForCommonParts(partName, partType));

    if (sortedContext.hasUnusedPart(part) )
        // Step 3 - Pass the part name and type in the parameters of the function
        return (getEGLFilePathForUnusedParts(partName, partType));

    hit = sortedContext.programPartFor(part);
    if (hit != null )
        return (getEGLFilePathForUniquePart(hit));

    return (defaultPath);
} //getEGLPathForPart()
```

## 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 Java to change the algorithm?

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

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

- From the VisualAge workspace, click on the **Projects** tab.
- Expand the “**IBM VisualAge Generator EGL Migration**” project.
- Expand “**com.ibm.vgj.mig**” package
- Select and expand “**VAGenToEGLMapper**” class to see the list of methods.
- If you prefer to work in a separate window, double-click the class, and a new window should open up displaying class
- If you prefer to work in a single window, select the method you want to edit. The source code appears in the Source pane.

To create a new method, do the following:

- Select and right click on “**VAGenToEGLMapper**” class.
- Select **Add -> Method... -> Create a new method**.
- Click on **Finish**
- Replace the template source code with your new code (for example, the code from Figure 2 or Figure 4).
- Select **File -> Save** to save this new method.

To change an existing method, do the following:

- Select the method you want to edit. The source code appears in the Source pane.
- Replace the lines of source code you want to change with the new source code (for example, the code from Figure 5).
- Select **File -> Save** to save your changes.

After you have made the code changes, remember to version the “IBM VisualAge Generator EGL Migration” project. Be sure to use a meaningful name because you will need to reference this version in the following situations:

- If you delete and then add the Stage 1 migration tool for any reason, you must reload your version of the tool.
- If you install a new version of the Stage 1 migration feature, you will need to apply your changes to the new version of the migration tool.