The IBM VisualAge Generator Newsletter

# VisualAge® Generator

A Powerful New Vision of Programming™

Volume 3, Number 1
December 1997

## Contents

Industry Speaks About IBM's VisualAge Family 2

A Customer Success Story 3

VisualAge Generator V3.0—MSLs to ENVY 4

Everything You Had with MSLs and More! 8

Interoperability—Passing Data to 3GL Programs 13

TriviaGen 16

E-mail—One More Time! 16

A Question from Us to You 19

HAPPY NEW YEAR

# VisualAge Generator Version 3.0 Availability

*by Sandra Johnson, VisualAge Generator Product Manager*

VisualAge Generator Version 3.0 is currently available in English and will be available in translated languages on December 31, 1997. The product development team has worked very hard to make the Windows NT development environment available to you, our existing and potential customers. And, as usual, we have tried to provide you with articles in this issue of the newsletter that will help you get started with VisualAge Generator Version 3.0.

1997 has been an exciting year as we have seen our VisualAge Generator customer base double since 1996. We are delighted that you are our customers and hope that you continue to find useful information in our newsletters. We continue to solicit your input for the newsletter. We want to provide you with information that is most helpful to you. To do this, we need to know what topics are of most interest to you. So please use the Comment Form in this newsletter to provide us with questions you want us to repond to or topics for which you want us to provide helpful hints or tips. Also, if you have helpful hints or tips that you would like to share with other VisualAge Generator users, you can write an article to be published in the newsletter. You can fax the article to (919) 254-0206 or send it by e-mail to gators@us.ibm.com.

When you send articles to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you. In addition, keep in mind that all submissions might not be published.

We ‘re looking forward to 1998!

## *Industry Speaks About IBM’s VisualAge Family*

M. Blechar of the Gartner Group, a leading information technology research and analysis firm, reported in Gartner Group’s “Application Development and Management Strategies Research Note” dated July 16, 1997, “In 2000, the list of leading enterprising AD tool vendors with the best cross-category support will include IBM, Oracle, and Microsoft® with its alliance partners (0.9 portability).”

The VisualAge family is in the leadership category of 3 out of 4 categories—including “E-CASE and Traditional Mainframe Generators” where VisualAge® Generator was listed as one of the leaders.

Please contact the Gartner Group for the report.

# A Customer Success Story

*by Rusty Edmister, VisualAge Generator Sales and Technical Sales Support*

In late 1996, Danka Office Imaging acquired Kodak's facilities management business, sales, and marketing and equipment service operations. Timeliness of customer deliveries is critical to the company's image and success. Danka OI is better positioned to fulfill its promises and to reduce costly inventory overhead. The reason: a set of advanced client/server applications built to streamline the order fulfillment process and to minimize inventory.

The IS group faced many challenges including the need to:

- Reduce the cost structure of maintaining applications
- Pragmatically migrate from a legacy environment to a client/server environment
- Significantly reduce the cycle time to deliver solutions
- Effectively develop new IS skills while leveraging people's strong knowledge of the business

The solution chosen to address these challenges was to make the transition to client/server using an object-based development tool. The tool was VisualAge Generator.

Jim Cantin, formerly a Systems Architect at Kodak's Information Services, headed a group that started with order entry and then added a suite of Lotus Notes applications to support order management. "Like many companies, Kodak had years of equity invested in legacy code," says Cantin. "What is particularly unique about our approach is that we used an object-based tool to create client/server applications, while retaining our mainframe IMS application logic and data structures wherever possible."
Using VisualAge Generator, the first client/server application went from concept to production in just six weeks. The client/server application is used by customer service specialists to dynamically schedule product manufacturing, transportation, delivery, and installation, all based on the customer's preferred date. It has a Windows graphical user interface that taps into IMS applications via MQSeries middleware.

"The strength of VisualAge Generator's visual programming really lends itself to speeding up the development process and increasing the quality," says Cantin. "We estimate developer productivity increased 20-30 percent initially, and we expect it to continue to rise. Even more important, we're able to sit down with our users for the first time and design applications together. And, we don't have to throw away the prototype—we just keep adding to it to get the final product. With the VisualAge Generator environment, everything you need is within one environment, including the middleware," he adds. "There's a client piece and a host piece, and VisualAge Generator handles the conversation for you. That means our developers can focus on writing the application without having to be concerned about writing to the communication layers of the execution platform. And our users don't know how we've tied it together. They just know it does what they want it to do. Involving users early on shortens the cycle time and training required. And—most important—the users are happy with the end results they see."

The next step was to introduce Lotus Notes to simplify the flow of information even further. With a customized Lotus Notes suite of applications, users now have a simple but powerful front-end interface to monitor and manage the workflow associated with open orders for customers. The scheduling application looks and functions like a calendar. "That metaphor makes it intuitive and easy for users to learn and become productive," says Cantin.

As an order moves through the order fulfillment cycle, updates on its status are sent to a VisualAge Generator host-based application on the mainframe and then to Lotus Notes via MQSeries. "It's a changing view of your data that is updated constantly because it is linked to the back-end data," adds Cantin.

Danka's customers have been impacted as well. In the first six months, customer satisfaction has increased six percent within the processes supported by these applications. Danka has seen a 12 percent increase in customer satisfaction related to improved delivery dates.

At the same time, Danka OI continues to rely on VisualAge Generator to develop new client/server applications. "We started with 42 users and now have 120 people on the system following the merger. I see no reason why this won't scale up to any number of users," states Cantin. What's more, the cost of migration to the object-based environment has long been recovered. Significant gains have been made that wouldn't have been possible in a traditional programming environment. "We wouldn't even have been able to propose these client/server solutions, especially the Lotus Notes interface, without VisualAge Generator," adds Cantin. "There's a lot of excitement in IS because we're accomplishing business objectives while enhancing our careers. We're developing skills that are highly valued by the customer today and that will help improve our competitive position tomorrow."

# VisualAge Generator V3.0–MSLs to ENVY

*by Jeri Petersen, VisualAge Generator Consulting Services*

This article is the first of two articles that describes the new VisualAge Generator V3.0 library management system (ENVY) and the library management system (MSLs) used in previous releases of VisualAge Generator. The second article follows this article. Both articles describe, from a slightly different angle, the new concepts and terminology in V3.0 and provide insight on how to prepare for the move from MSLs to the new library management system.

In both Cross Systen Product (CSP) and previous releases of VisualAge Generator, code was written in small pieces called members. Members were stored in Member Specification Libraries (MSLs). In VisualAge Generator V3.0, the code must be stored in ENVY, a library management system. If you plan to use the code from CSP and previous releases of VisualAge Generator, you must migrate this code from the MSLs into ENVY. This article provides an insight to VisualAge Generator V3.0, and how it differs from previous releases of VisualAge Generator and CSP, and what you can do to start preparing for these changes. The following are covered:

- ENVY Characteristics
- Comparison of MSLs and ENVY
- The Migration Assistance Tool

## *ENVY Characteristics*

Using the ENVY library manager to store information is a major change for VisualAge Generator V3.0. There are new terms that are important for the ENVY environment. Because ENVY was originally developed for use with Smalltalk, some explanations of the terms include Smalltalk information to help relate the terminology to Smalltalk. The new terms are:

VAGen part class

Each 4GL member type (all member types except GUIs) becomes a *VAGen part class*. A *VAGen part class* is an extension of a class in Smalltalk. The *VAGen part classes* appear in the **VAGen Parts** pane of the **VisualAge Organizer** window. The *VAGen parts classes* created for the member types are prefixed by *VAGen* (for example, *VAGen Records*).

There are 5 additional VAGen part classes that are used to contain control information that was stored outside the MSL in previous releases of VisualAge Generator. These *VAGen part classes* are for linkage table, resource association, generation options, bind, and linkage editor information that is required during test and generation.

VAGen part

Each 4GL member is now stored as a *VAGen part.* A *VAGen part* is associated with a Smalltalk *method* in an extension of its *VAGen part class*. The VAGen parts appear in the **VAGen Parts** pane of the **VisualAge Organizer** window and in the VAGen Parts Browser window.

View

Each GUI is now stored as a *view*, which is a *visual part*. A view is a *class* in Smalltalk. The views appear in the **VAGen Parts** pane of the **VisualAge Organizer** window. Views do not appear on the **VisualAge Generator Parts Browser** in ENVY.

Program

The application member type has been changed to *program* to distinguish it from an ENVY application.

ENVY application

An ENVY application is a group of *classes* and *methods* that are closely related in function. An ENVY application can include *VAGen part classes* and *VAGen parts*. An ENVY application is also called an *application*.

Configuration Map

A *configuration map* is a group of application editions that should be loaded together into a developer's image.

The following ENVY concepts are new for VisualAge Generator V3.0:

Functional organization

ENVY enables you to group parts into aplications. These applications can (and should) be organized by function.

Ownership

Each configuration map and each application have an assigned manager who is responsible for the integrity of the code that is placed in the configuration map or application.

Each part (class) or VAGen part class has an assigned owner who is responsible for the integrity of the code that is placed in the class. For 4GL parts, this means that the owner of the class *VAGenRecords* within application XYZ is responsible for the integrity of *all* record definitions stored as

part of application XYZ. Because each GUI becomes a separate view (visual part), each GUI within application XYZ can have a different owner. Fourth-generation language (4GL) parts used within the GUI become VAGen parts.

**Note:** In previous releases of VisualAge Generator, the closest concept to ownership was write-protecting the staging, test, or production MSLs and only giving a team leader the authority to advance into the MSLs.

Edition

Each change that is made to a 4GL VAGen part results in a new edition of the VAGen part being stored in the ENVY library. Editions of parts, applications, and configuration maps are also stored in the ENVY library.

Version

Editions can be frozen to prevent further changes to that level of code. The frozen edition is called a *version*. After a part, application, or configuration map is versioned, the only way to make changes is to open a new edition.

Image

An image is the developer's current view of the ENVY library. It contains the version or edition of the configuration maps, applications, and parts that the developer wants to work on. Only one copy of a VAGen part can be loaded into the image at one time.

# *Comparison of MSLs and ENVY*

The following sections provide information on what is different in the use of MSLs and ENVY.

## Member Types

In releases prior to VisualAge Generator V3.0, there was one member type for each type of code that could be written: application, GUI (for VisualAge Generator only), record, table, data item, map group, map, process, statement group, and PSB (Program Specification Block).

For VisualAge Generator V3.0, each 4GL *member type* is a *VAGen part class* that is indicated by the prefix *VAGen* (for example, *VAGenRecords*). For GUIs, there is no corresponding VAGen part class, because each GUI becomes a separate view.

## Storing Members and Other Information

In releases prior to VisualAge Generator V3.0, an MSL was an OS/2 directory and each member was a file within the directory. In CSP, an MSL was a VSAM file and each member was stored as records within the file.

Control information was needed for test and generation. Depending on the version of CSP and VisualAge Generator, this information included: resource association information, generation options, linkage table, bind commands, and linkage editor control statements. For CSP, the resource association information and some generation options were stored in the MSL; the other control information was stored outside the MSL. For previous releases of VisualAge Generator, all the control information was stored outside the MSL. For VisualAge Generator V3.0, all information is stored in the ENVY library. Each 4GL member becomes a *VAGen part* and is associated with a Smalltalk method. Each GUI becomes a *view* or *visual part* and is a Smalltalk *class*.

Resource association files, generation options, linkage tables, bind command files, and linkage editor control statement files all become new *VAGen part classes* in ENVY and the files become *VAGen parts*. Thus, all the data required for test and generation is contained in a single library management system.

## MSL Concatenation

In releases prior to VisualAge Generator V3.0, MSLs could be concatenated. When MSLs were concatenated, for test and generation, only the first found member with a given name was used. For viewing, members from MSLs other than where the first found member was located could be referenced. If changes were made to a member in the MSL concatenation sequence, the changed member was stored in the read/write MSL (the first MSL in the concatenation sequence).

For VisualAge Generator V3.0, there is no concept similar to MSL concatenation. All editions and versions are available in the ENVY library. However, only one edition or version of a part can be loaded at a time. *Browsers* are available to compare editions and versions within the ENVY library before loading them into your image to determine which one is the required level of code. Configuration maps can also be used to group code for a particular level. For example, you might have a configuration map for production that indicates the version of each application that is in production.

## Functional Organization

In CSP, and previous releases of VisualAge Generator, members were grouped together into MSLs. Generally, an MSL contained all the members for a particular subsystem. Because the MSL was the only method for grouping members by function, the functions tended to be quite large. In CSP, the number of MSLs in a concatenation sequence was limited to 6. This also contributed to having a large number of members in each MSL.

With VisualAge Generator V3.0, the configuration map and the ENVY application provide a two-level capability for grouping parts. The configuration map is the higher level of organization and more closely resembles an MSL in terms of the number of parts. ENVY applications enable you to organize your parts into smaller groups than was reasonable to do with MSLs. This provides more capabilities in terms of controlling access to the parts, finding a part, and limiting the number of parts displayed in the **VAGen Parts Browser**.

## Member Associations

In releases prior to VisualAge Generator V3.0, members could have associates—other members that were referenced within the member. For example, the associates for an application included referenced map groups, maps, records, tables, PSBs, processes, statement groups, and their associates. Any global data item referenced directly or indirectly was included. For example, a data item used as a called parameter was included as an associate of the application. Only maps that were actually used by the application were included in the associates. The associates for a GUI included referenced records, tables, processes, statement groups, or embedded GUIs, and their associates. (An external GUI and its associates were not included.) For a specific member (for example, application XYZ), the only associations that could be detected were those that existed in the current MSL concatenation sequence, using the first found members.

For VisualAge Generator V3.0, the same associations between 4GL VAGen parts still exist.
For a *view*, the following are associates:

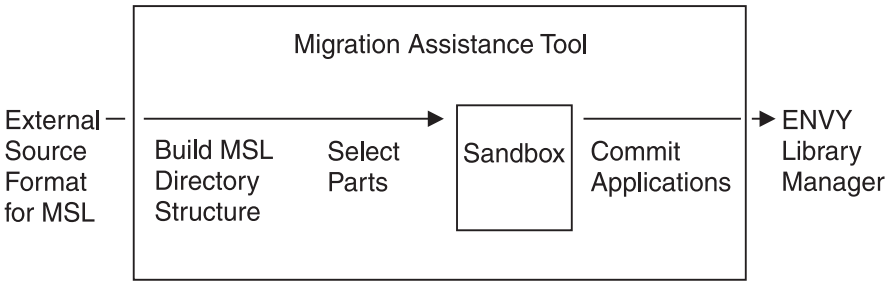
- 4GL parts—records, tables, processes, and statement groups and their associates
- For any embedded view, its associates. However, the embedded view itself is not included as an associate.

Thus, the associates of a view in VisualAge Generator V3.0 are the same as they were for a GUI in VisualAge Generator V2.2, except that the embedded view itself is no longer an associate.

The only associations that can be detected are the ones that exist within the current image.

## The Migration Assistance Tool

The Migration Assistance Tool is designed to assist in migrating MSLs to the ENVY library.




**Note:** Although the Migration Assistance Tool can help determine when members are not found, it cannot locate missing members. Similarly, the Migration Assistance Tool can help determine when duplicates of a given member exist, but it cannot make the determination as to which is the correct or current level of the member.

The Migration Assistance Tool works as shown in the above figure. The figure shows a sample MSL concatenation for multiple subsystems.

You can run the Migration Assistance Tool from either OS/2 or Windows NT.

The Migration Assistance Tool enables you to build an MSL directory structure from the external source format that corresponds to an MSL. This step is required if you are migrating from CSP or moving to the Windows NT development environment.

After the MSL directory structure(s) are created, you can select parts (members) from an MSL or MSL concatenation, group them into an application or series of applications, and move them to a “sandbox”. When you move a part to the sandbox, its associates also move. For a GUI, its referenced embedded GUIs and external GUIs and their associates are moved. The applications in the sandbox are in the current image, but are not in the ENVY library yet. This enables you to manipulate the applications and to rearrange the VAGen parts within the applications until you are satisfied with the grouping. Only one version of a part can be in the

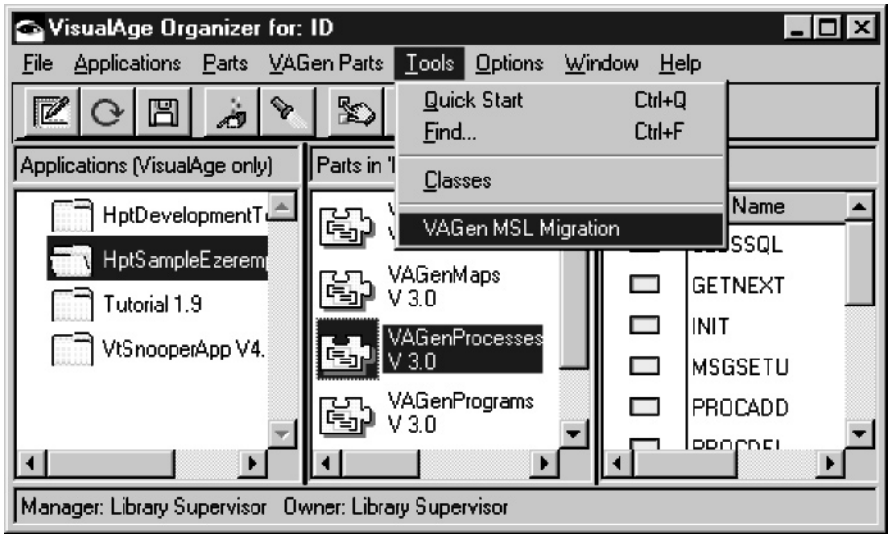
sandbox at one time. Each part can only be in one ENVY application within the sandbox. After you are satisfied with your organizational structure for the applications, you can commit the applications to the ENVY library. Committing the applications creates an edition of the applications in the ENVY library, creates any needed VAGen part classes for the 4GL member types, creates views for GUIs, and creates the VAGen parts for the 4GL members.

After the applications are in ENVY, you can use any of the ENVY library management functions, such as:

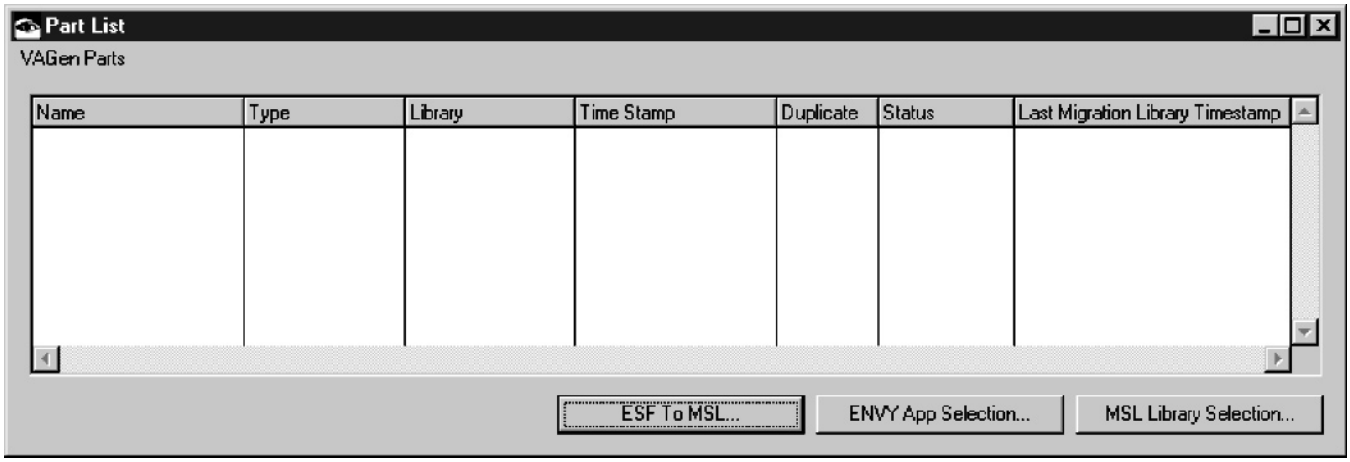
- Versioning and releasing the VAGen part classes and parts
- Versioning and releasing the applications
- Creating configuration maps
- Dividing applications into subapplications
- Assigning a class owner or application manager

You can start the Migration Assistance Tool from the **VisualAge Organizer** window by selecting **Tools** and then **VAGen MSL Migration**. The **Part List** window appears. These windows are shown below.

Refer to the *VisualAge Generator Migrating MSLs to ENVY* (SH23-0252) document for more information.




*The VisualAge Organizer window*




*The Part List window*

# Everything You Had with MSLs and More!

*by Henry Jicha, VisualAge Generator Consulting Services*

In VisualAge Generator V3.0, the transition from MSLs to a library control and versioning system will seem formidable at first, but a closer look reveals that the changes to VisualAge Generator really offer increased control and flexibility. In addition, it provides functionality that is comparable to or exceeds that provided with MSLs. The key is to understand how the new terminology and concepts relate to traditional MSL-based development concepts and tasks. Once you understand the concepts and tasks, the planning process for migration from previous MSL-based versions of Cross Systems Product (CSP), previous releases of VisualAge Generator, and VisualAge Generator V3.0 capabilities becomes clearer and easier.

The purpose of this article (along with the article on page 4) is to review VisualAge Generator V3.0 and explain how the new concepts relate to existing MSL and product functionality. This should assist you in the planning and use of the new environment.

## *What's Changed?*

One of the major changes in VisualAge Generator V3.0 is in the terminology—for example, the renaming of *applications* (.app parts) to *programs*. This makes sense, because this is the truest mapping of what these components really are in the development environment. In general, an application is a group of one or more programs that together perform a function. This corresponds to VisualAge Generator's new definition of application: a group of parts that are functionally related, and that can be versioned and controlled during development as a unit. The exception to this is applications made up of common parts. These will be discussed later. Note that this new definition of application will be used throughout the rest of this article. Components referred to as applications in prior releases of VisualAge Generator will be called programs.

Another change is MSL *members* are now referred to as *parts*. This emphasizes the object-based modular programming approach and the assembly-from-parts paradigm inherent in previous releases of VisualAge Generator and its predecessor, CSP.

Prior to VisualAge Generator V3.0, related application code, generally comprising one or more user application systems, was commonly kept in a single MSL. Common parts and commonly shared MSLs will be discussed later. The MSL library management system provided a logical grouping of functionally related parts and a limited means of control, so that when the application system was put into production, or a new release was put into production, the MSL could be either copied or exported to preserve the exact source code associated with the executable object or ALF contents.

## *Code Storage in Applications*

In VisualAge Generator V3.0, the approach is to put this code into one, or usually into multiple applications, since MSLs were not generally as granular as applications are in the new ENVY library management system used by VisualAge Generator V3.0. This library management system provides the same logical grouping of related code. As a rule of thumb, an application should have no more than 500 parts, and should be made up of code that will only be addressed by one developer at a time. Although multiple developers working on a single application is fully supported, it can require additional change coordination.

## *Accessing Code*

In the ENVY library management system used by VisualAge Generator V3.0, the code is organized into applications. To access this code, the developer loads the desired application into the image owned by the developer from the repository. This makes the source accessible to the developer for change or enhancement.

## *The Image*

In VisualAge Generator V3.0, the *image* is the working set of ALL the code that a developer has current access to. The image includes the application code being developed by VisualAge Generator V3.0 as well as the code that makes up

VisualAge Generator itself. This image is loaded every time VisualAge Generator is started. The initial image shipped with the VisualAge Generator is made up of all the Smalltalk classes that comprise VisualAge Generator and VisualAge Smalltalk. Applications can then be loaded and unloaded to enable development on the applications code. Additionally, parts that were developed by third parties, such as those available through the Object Connection program, can be added to the image, enhancing the capabilities of the VisualAge Generator. Prior to VisualAge Generator V3.0, VisualAge Generator functionality was fixed, since the image was not explicitly referenced or accessible. All application code was stored and accessed outside the image in MSLs.

One of the most important productivity benefits of CSP and previous releases of VisualAge Generator was the ability to reuse parts and definitions. The parts and definitions were commonly kept in shared common MSLs. This meant that there could be parts that were used in more than one application. In VisualAge Generator V3.0, applications can be used not only for functional groupings of code, but also as storehouses of common code elements such as record or data item definitions, or common processes or statement groups. This replaces the shared common MSLs in the repository.

## *Versioning*

A major enhancement of VisualAge Generator V3.0 is the ability to create versions of stored applications. A *version* is a copy of the application source that has been frozen by the application manager, and cannot be changed. This enables the application to be moved to test and production, knowing that additional changes have not been made to the source. It also alleviates the process of exporting, promoting, or copying the code. Best of all, creating a new version does not overlay previous versions or working copies of the code in the library. You can always refer back to the previous versions or working copies, which will be saved indefinitely. Versioning also means that a complete change history is maintained for the code. To go back to a previous version, load the previous version from the code repository into the developer's image.

Versioning is also done at a more granular level within applications to assist with development management and control. This capability enables developers to freeze the code at various points during development and control the visibility of the changes to other developers. Additionally, VisualAge Generator V3.0 provides facilities that compare the different versions of the code (by highlighting changed areas). This is particularly useful in parallel development, or if an emergency fix needs to be integrated with an ongoing development effort.

## *Configuration Maps*

Previously, applications could be loaded one at a time. This provided access to code for development. However, for a large subsystem with multiple sets of common parts this could be a long and tedious process. VisualAge Generator V3.0 library manager addresses this process with *configuration maps*. Configuration maps consist of related groupings of applications that can be loaded together at appropriate controlled release levels. Once a configuration map is defined, the developer needs to load only the configuration map, not the individual applications. Configuration maps also provide the vehicle for loading the image with the appropriate applications during LAN generation.

A better way to look at configuration maps is as replacements for MSL concatenations previously used by developers. MSL concatenations were usually created by the project leads and reflected the source code management strategy for the project, as well as the granularity of code management. Likewise, configuration maps are generally created, managed, and maintained by project leads, who are designated as the configuration map managers. These configuration maps reflect the scope of the project components, and the code management strategy. Configuration maps are also versioned by configuration map managers. This enables the configuration map manager to ensure consistency between the releases of the applications in a configuration map. To further ensure consistency, it is recommended that an application appear in only one configuration map.

You can set up a configuration map where it requires other configuration maps. This is particularly useful with reusable code. The applications with shared definitions can all be put into one configuration map, and then this configuration map can be included in the configuration maps of multiple subsystems that use these parts in development. This ensures that an application only appear in one configuration map as well. Applications loaded because they are part of a required configuration map are still viewed as only existing in the primary (the required) configuration map, and are therefore in only one configuration map.

VisualAge Generator V3.0 library management system enables the storage of several versions of the same code in the code repository by version number. However, if you are loading applications from a configuration map, there must be a means of identifying the correct versions of the applications to load automatically. This is done via the release process. Each application has a manager that determines when a version is created. The application manager or the configuration map manager, then has the ability to release this code to the configuration map. When code is released, the released version is the one used when the configuration map is loaded. This provides a new level of control unavailable in normal MSL concatenations. With MSLs, a whole new concatenation had to be created every time a new version of code was included, while preserving the old one.

## Editions

If the code in a version is frozen, then how does a developer perform development on existing code? After loading the correct version of the desired application, the developer creates an edition of the component that needs work. An *edition* is a working copy of the code that can be modified. An edition is created automatically for a developer whenever a save is done on new or changed code. A save does not overlay the existing code. Instead, each time changes are saved, a new edition is created in the code repository. Each edition has a date and timestamp for identification. This means that developers can retrace their steps, or backtrack if they introduce an unwanted behavior. Prior to VisualAge Generator V3.0, once a developer saved changes in a read/write MSL, any previous copy of the member in the MSL was lost.

## Code Control

Within VisualAge Generator V3.0 applications, there is another layer of code control. This layer groups all VisualAge Generator parts of the same type (for example record parts) in an application for control. This grouping is called a *class*. Each GUI is also a class, and is controlled at the class level. Each class has an owner. When a developer is satisfied with the changes made to one of the groups of VisualAge Generator parts, the developer can version the class. This freezes the changes, and enables the developer to easily reload the class and all its parts at this level. The owner then determines what developer versions to release to the application. By deciding what versions to release, and when to release them to the application, the class owner controls the quality of the code that goes into the application. This can also prevent conflicting changes to an application in the case of multiple parallel-development efforts on a single class. It offers considerably more control than collision detection, as offered with MSLs in previous releases of VisualAge Generator.

## Code Control Strategies

Because versions of applications are fixed, and the application manager controls creation of versions and editions of applications, multiple strategies can be used for development of code in an application without the need for constant application manager intervention. One option is that the application manager open an edition of the application, so that code can be changed in this edition. This strategy enables developers to share code changes in the components they own across the development team without further intervention of the application manager. This is done by having the class owner release changed code into the open application edition after the code developer creates an appropriate version of the application edition. If an open edition is not created, then VisualAge Generator V3.0 enables the creation of a *scratch edition* of an application by each developer. This scratch edition is visible only to the developer, and cannot be versioned or released. However, it enables the developer to do code development and create new versions of code in the application. Periodically, the application manager opens an edition. At this point, the code versions created in the scratch editions can be loaded into the application edition, and released by the class owner(s) to prepare to version the entire application.

This article is not a full review of all of the functions available in VisualAge Generator V3.0. It focuses on features that provide functions analogous to those used in MSL-based development. There are many more subtleties, many of which result from the new open access to the true object nature of VisualAge Generator. Careful review of these subtleties can help in your migration to the VisualAge Generator V3.0.

## *Planning the Migration*

How then does one prepare for the migration to VisualAge Generator V3.0? The first step is the gathering of information such as:

- The inventory of MSLs
- The user application systems
- The user application systems components
- The parts

This must be done so that a plan for mapping to VisualAge Generator V3.0 applications and configuration maps can be developed. Note any situations where duplicate names exist either for the same member in different MSLs in a concatenation (which can be addressed during migration), or in different member types which have a common name (for example, a record and data item with the same name). In VisualAge Generator V3.0, all names in the image, regardless of part type, must be unique. If the duplicate names for different member types are used at the same time in development, then code modifications are required to change the names and resolve the changes in embedded references. Note that while this means that there can be no duplicate names in an application, it does not mean that different applications cannot have copies of the same member (as with different MSLs), or duplicate member names for different member types. This is acceptable, though not good practice, so long as there is no need to load the two applications simultaneously.

When inventorying your parts, it is important to remember the methodology you use to move code from development to production, and the scope of the components that are promoted. These will be important factors in determining how to define configuration maps. A good rule of thumb is that a configuration map should reflect a piece of code that would be put into production. Applications can then be created within this configuration map, but keep in mind the size guidelines for an application. If production changes are extremely granular, it might be appropriate to have at least an application for each unit that can be put into production for version control. Configuration maps can also specify a developer responsibilities, enabling the loading of only that developer's code. These developer-based configuration maps can then become required configuration maps in a configuration map that is based on a functional breakdown of the application code library.

If production changes are extremely infrequent, and are extremely comprehensive, VisualAge Generator provides several options. In addition to configuration maps and applications, you can divide the applications into *subapplications*. Subapplications are applications that are grouped logically under a single master application, but are not delivered individually. For example, ledger update is always updated as a whole, but consists of marketing ledger update, accounts receivable ledger update, and accounts payable ledger update, each of which is small enough to meet the size guidelines for an application. These might be good candidates for subapplications. One disadvantage of subapplications is that they are not directly supported by the MSL Migration Tool provided with VisualAge Generator V3.0.

Just as every application and configuration map must have a manager in VisualAge Generator V3.0, every class within an application (every GUI), and each type of VisualAge Generator part in an application must also have an owner. These owners can vary from application to application. While developers can version definitions at the class level, the owner is responsible for releasing the classes that they own into applications.

In CSP and earlier releases of VisualAge Generator, many shops might have had informal ownership based on shop standards, but it was not strictly enforced. To prepare for the new ownership environment, it is important to:

- Create an organization chart
- Outline the responsibilities of each role
- Review roles and responsibilities in promotion of code from development to production
- Set audit points in this process
- Define any separation of duties requirements

This information should be sufficient for an initial plan for designating owners of various pieces of code within the organization, with appropriate spans of control and responsibility. If you have different developers for maintenance and new development on the same system(s), it might be difficult to determine ownership. Careful consideration will have to be given to those who really should own the code for versioning and release purposes. One option is to have two separate code repositories, one for maintenance and another for new development with a coordination strategy.

Once these plans have been developed and validated by walkthroughs of different development scenarios, you can begin to implement your plan. Application lists and dependencies can assist in determining how to stage moving systems, subsystems, or single applications from your present MSL-based environment into VisualAge Generator V3.0. Again, it is recommended that you move related groups together in manageable pieces. Ensure that you have sufficient staff to migrate and maintain normal operations as well as adequate change control to manage the code in your old system and your new VisualAge Generator V3.0 system while in transition.

For information to assist you in planning the physical migration of your code to match your plan, refer to the *VisualAge Generator Guide to Migrating MSLs to ENVY* (SH23-0252) document. This document provides guidance and scenarios for implementing your VisualAge Generator V3.0 environment, based on how your existing MSLs are set up. It is recommended that you obtain additional education on the technical infrastructure of VisualAge Generator V3.0 and its code repository. Other sources available are the redbook *VisualAge Generator System Development Guide* (SG24-4230) and the VisualAge Generator Consulting Services group.

The VisualAge Generator Consulting Services expertise can facilitate the planning and execution of your migration, and provide the leverage of skills acquired across a wide variety of customer situations, many of them like yours. Consultants also offer an impartial outside view that can help you through difficult issues of determining ownership, or defining application boundaries for mapping into the new code. Whether or not you use consultants, ensure that you have and use all the resources necessary to do the migration. It is recommended that you initially migrate a small segment of your applications, then validate your plan. The experience you gain from this process can then be used to modify your strategy for the rest of the migration.



## *Conclusion*

In conclusion, the migration of your code into the VisualAge Generator V3.0 code repository from MSLs is a manageable process. Although it will take some time to understand VisualAge Generator V3.0 development environment, terminology, and new functionality, the development process used with MSLs can be updated and moved to VisualAge Generator V3.0 with enhancements to provide more functionality and control than was previously available.

*A special thanks to Jeri Petersen for her assistance with this article.*

# Interoperability—Passing Data to 3GL Programs

*by Chuck Proffer, Chris Biega, and Rob Swofford, VisualAge Generator Development*

Occasionally, you need to call a non-VisualAge Generator program and pass information via parameters. VisualAge Generator supports many different data types and some of them do not map directly to data types found in the workstation environments (OS/2, Windows NT, AIX, HP). This article discusses each of the VisualAge Generator data types in the context of the workstation environments and shows how to pass them to 3GL programs written in C/C++, COBOL, and PL/I.

This is the first in a series of articles that will cover the interoperability between VisualAge Generator and programs written in 3GL languages. Follow-on articles will cover 3GL programs calling VisualAge Generator, VisualAge Generator calling 3GL programs, advanced debugging, and conversion between the various workstation file formats supported by VisualAge Generator.

## *Data Types*

The following data types are supported by VisualAge Generator:

BIN

The BIN data type contains numeric data stored in a binary format. For the best performance, compatibility, and interoperability, numeric data should be stored in binary format.

CHA

The CHA data type contains ASCII alphabetic, numeric, or national characters. The maximum length for a CHA data item is 32767 bytes.

DBCS

The DBCS data type contains double-byte characters. DBCS (Double Byte Character Set) data requires 2 bytes for each character. DBCS is required for languages such as Chinese, Korean, and Japanese. These languages have so many characters in their language, they cannot be represented in 1 byte. They require special terminals and printers that are DBCS capable.

HEX

The HEX data type contains hexadecimal characters. Each byte of data is represented by two hexadecimal digits.

MIX

The MIX data type contains both single-byte and double-byte characters.

NUM

The NUM data type contains numeric data stored in zoned decimal format. The sign of the number is stored in the upper half of the last byte. A positive sign is represented by the hexadecimal digit 3 and a negative sign is represented by the hexadecimal digit 7. The NUM data type is supported for compatibility with previous versions of VisualAge Generator. For the best performance, compatibility, and interoperability, numeric data should be stored in binary format.

NUMC

The NUMC data type is identical to the NUM data type in ASCII. The NUMC data type is supported for compatibility with previous versions of VisualAge Generator. For the best performance, compatibility, and interoperability, numeric data should be stored in binary format.

PACF

The PACF data type contains numeric data stored in packed decimal format. Packed decimal data contains 2 digits in each byte. The sign of the number is stored in the lower half of the last byte. A positive sign is represented by the hexadecimal F and a negative sign is represented by the hexadecimal D. A hexadecimal B is also accepted as a negative sign by some products. The PACF data type is supported for compatibility with previous versions of VisualAge Generator. For the best performance, compatibility, and interoperability, numeric data should be stored in binary format.

PACK

The PACK data type also contains numeric data stored in packed decimal format with the sign of the number stored in the lower half of the last byte. A positive sign is represented by the hexadecimal C and a negative sign is represented by the hexadecimal D. A hexadecimal B is also accepted as a negative sign by some products. For the best performance, compatibility, and interoperability, numeric data should be stored in binary format.

# *Data Type Comparison*

The following shows the data type comparison for 3GL programs written in C/C++, COBOL, and PL/I:

| VisualAge Generator | C/C++ | COBOL | PL/I |
|---|---|---|---|
| BIN(2 byte) | SHORT | PIC S9(4) COMP-5 | fixed bin(15) |
| BIN(4 byte) | LONG | PIC S9(9) COMP-5 | fixed bin(31) |
| CHA | CHAR | PIC X(?) | CHAR |
| DBCS | CHAR | PIC G(?) usage display-1 | graphic(?) |
| HEX | CHAR | PIC X(?) | CHAR |
| MIX | CHAR | PIC X(?) | CHAR |
| NUM | CHAR | PIC S9 usage display | CHAR |
| NUMC | CHAR | PIC S9 usage display | CHAR |
| PACF | CHAR | PIC S9 comp-3 | fixed decimal |
| PACK | CHAR | PIC S9 comp-3 | fixed decimal |

## Examples

The following examples show how to define each of the data types in C/C++, COBOL, and PL/I. Parameters can be passed either as separate data items or combined and passed as a record. The examples assume that the VisualAge Generator program is passing a working storage record defined as follows:

| Name | Type | Length | Decimals | Bytes |
|---|---|---|---|---|
| BIN2ITEM | Bin | 4 | 0 | 2 |
| BIN4ITEM | Bin | 9 | 0 | 4 |
| CHAITEM | Char | 8 | 0 | 8 |
| HEXITEM | Hex | 8 | 0 | 4 |
| NUMITEM | Num | 8 | 0 | 4 |
| NUMCITEM | Numc | 8 | 0 | 4 |
| PACKITEM | Pack | 5 | 0 | 3 |
| PACFITEM | Pacf | 5 | 0 | 3 |

## Examples using C/C++

In C or C++, the record is defined using the struct type declaration. Note the use of the *_Packed* keyword. If the structure is not packed, the compiler will align each element of the structure according to its data type and add pad bytes as necessary. Since the VisualAge Generator records are byte aligned, a misalignment will occur and garbage data will be received by the C/C++ program. Also note that C++ does not support the *_Packed* keyword. Therefore, the #pragma pack must be used:

```
_Packed typedef struct
{
   short  bin2Item;
   long   bin4Item;
   char   chaItem[8];
   char   hexItem[4];
   char   numItem[8];
   char   numcItem[8];
   char   packItem[3];
   char   pacfItem[3];
} passed_rec;
```

## Examples using COBOL

To align the record data on a byte boundary in COBOL, add the SYNCHRONIZED clause to the main level of the record being passed. If the SYNCHRONIZED clause is not specified, the compiler aligns each element of the structure according to its data type and add pad bytes as necessary. Since VisualAge Generator records are byte aligned, a misalignment occurs if the SYNCHRONIZED clause is not specified.

```
01  passed-rec SYNCHRONIZED.
        10 bin2item    PIC S9(4) usage comp-5.
        10 bin4item    PIC S9(9) usage comp-5.
        10  chaitem    PIC X(8).
        10  hexitem    PIC X(4).
        10  numitem    PIC S9(8) usage display.
        10  numcitem   PIC S9(8) usage display.
        10  packitem   PIC S9(5) usage comp-3.
        10  pacfitem   PIC S9(5) usage comp-3.
```

### Examples using PL/I

To align the record data on a byte boundary in PLI, add the unaligned attribute to the main level of the record being passed. If the unaligned attribute is not specified, the compiler will align each element of the structure according to its data type and add pad bytes as necessary. Since VisualAge Generator records are byte aligned, a misalignment occurs if the unaligned attribute is not specified.

```
dcl 1 passed_rec unaligned
BYADDR,
    2 bin2item  fixed bin(15),
    2 bin4item  fixed bin(31),
    2 chaitem   char(8),
    2 hexitem   char(4),
    2 numitem   char(8),
    2 numcitem  char(8),
    2 packitem  fixed decimal(5),
    2 pacfitem  fixed decimal(5);
```

# Want More Control? You've Got It!

*by Sharon Thompson, VisualAge Generator Development*

VisualAge Generator supports a variety of client and server system combinations via a number of communication protocols and middleware. One of the most widely used middleware vehicle is the CICS Client External Call Interface (ECI) used to call CICS servers. In past releases of VisualAge Generator, there was little control of errors when using the CICS client middleware. A return code of 7650 in EZERT8 was often returned from a call, but the code could represent so many situations. For example, a return code of -3 from CICS client indicates that the CICS server system is unavailable. In this case, a user might decide to make a call to another CICS system as opposed to ending the application.

VisualAge Generator V3.0 provides additional messages for the most common errors encountered when using the CICS client middleware, thus giving the user more control of the application. The new messages are as follows:

| | |
|---|---|
| CSO7651E | An error was encountered calling program %1 using the CICS ECI. Return code: -3 (ECI_ERR_NO_CICS). CICS system identifier: %2. |
| CSO7652E | An error was encountered calling program %1 using the CICS ECI. Return code: -4 (ECI_ERR_CICS_DIED). CICS system identifier: %2. |
| CSO7653E | An error was encountered calling program %1 using the CICS ECI. Return code: -6 (ECI_ERR_RESPONSE_TIMEOUT). CICS system identifier: %2. |
| CSO7654E | An error was encountered calling program %1 using the CICS ECI. Return code: -7 (ECI_ERR_TRANSACTION_ABEND). CICS system identifier: %2. Abend code: %3. |
| CSO7655E | An error was encountered calling program %1 using the CICS ECI. Return code: -22 (ECI_ERR_UNKNOWN_SERVER). CICS system identifier: %2. |
| CSO7656E | An error was encountered calling program %1 using the CICS ECI. Return code: -27 (ECI_ERR_SECURITY_ERROR). CICS system identifier: %2. |
| CSO7657E | An error was encountered calling program %1 using the CICS ECI. Return code: -28 (ECI_ERR_MAX_SYSTEMS). CICS system identifier: %2. |

Message CSO7650E is still used for all other error conditions. These messages can also be found in the VisualAge Generator *Messages and Codes Reference* document (GH23-6597).

# TriviaGen

VisualAge Generator is enjoying great success around the world. Let's see if you can answer the questions below. Answers to these questions are on page 19.

1) VisualAge Generator is installed in 42 of the states in the U.S. Can you name the eight states in which there are no VisualAge Generator customers?

2) Can you name the southernmost country in the world in which there is a VisualAge Generator customer?

3) In what countries, completely or partially located south of the equator, can you find a VisualAge Generator customer?

4) One of VisualAge Generator's newest customers is a bank with over 25,000 branch offices. In what country is this customer located?

5) VisualAge Generator is installed in 51 countries around the globe. On which continent is there no VisualAge Generator customer?

6) Government accounts make up a very large part of the VisualAge Generator customer base. Of the 51 countries in which VisualAge Generator is installed, how many have an organization using this product?

7) VisualAge Generator is a "hot" product in the Asian countries (east of Pakistan and west of Hawaii). Based on the number of installed customers at the time this newsletter was written, what are the top three Asian countries with the largest number of installed customers?

8) There are hundreds of companies, educational institutions, and government organizations using VisualAge Generator in Europe. Which European country has the largest number of installed customers?



# E-mail—One More Time!

In this extremely fast-paced world, having a way to contact you immediately can be beneficial. There are things that come up from time to time about VisualAge Generator that we would like to pass on to you. Product updates, changes, teleconference schedules, news, and customer experiences are but a few categories of information that can help you do your job better and save your company money.

This newsletter is one way we communicate better, but it is published only three times a year. If you would like us to send you information as it happens, send your name, your company name, and your e-mail address to Rusty Edmister at **edmister@us.ibm.com**.

# Comment Form

## *Please check any appropriate boxes:*

- ☐ I'd like to receive future issues of this newsletter. (You need to check this item only if you have not already responded.)
- ☐ I'd like more information about Version 3.0.
- ☐ I'm interested in writing an article to include in *The VisualAge Generator Newsletter*. Subject:_____
- ☐ I'm interested in participating in an AD users' group meeting.
- ☐ I'm interested in participating in a VisualAge Generator users' group meeting.

**I have a question I'd like to submit for the Question & Answer section of this newsletter:**

**Are we putting the type of information you want to see in the newsletter? If not, what would you like to see in the newsletter?**

**Any comments you'd like to share with us about VisualAge Generator or about this newsletter? (Include your comments or concerns about VisualAge Generator's future directions here.)**

| | |
|---|---|
| Name | Title |
| Company Name | |
| Street Address/P.O. Box | |
| City | State/Province |
| ZIP/Postal Code | Country |
| Phone No. | FAX No. |

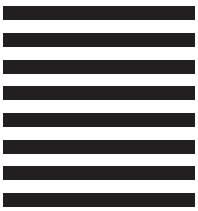
*Fold, tape, and mail this page - no postage is required. Or FAX it to (919) 254-0206.*

G242-0315-07

Cut or Fold Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines
The VisualAge Generator Newsletter
Newsletter Editor
T22/062/J125
P.O. Box 12195
RTP, NC 27709-2195
USA

Fold and Tape **Please do not staple** Fold and Tape

G242-0315-07

Cut or Fold Along Line

# A Question from Us to You

Do you have questions on how to attach individual parts on your windows/forms/dialogs, etc. so that they look consistent and size appropriately? If so, use the Comment Form in this newsletter to send us your questions. Then, in future issues of the newsletter we will provide you with articles on how to perform the various tasks.

## VisualAge Generator Web Pages

The VisualAge Generator web address is:

`www.software.ibm.com/ad/visgen`

For IBM's predecessor 4GL, Cross System Product, the web address is:

`www.software.ibm.com/ad/visgen/csp`

**TriviaGen Answers** (Questions are on page 16)

1) Delaware, Idaho, Maine, New Hampshire, Nevada, South Dakota, Vermont, and Wyoming
2) Chile
3) Argentina, Australia, Brazil, Chile, New Zealand, and South Africa
4) People's Republic of China
5) Antarctica
6) 32
7) Taiwan, Japan, and Korea (in that order)
8) Germany

## Acronyms

| | |
|---|---|
| 3GL | third-generation language |
| 4GL | fourth-generation language |
| AIX | Advanced Interactive Executive |
| API | Application Programming Interface |
| AS/400 | Application System/400 |
| CAE/2 | Client Application Enabler/2 |
| CASE | Computer-aided Software Engineering |
| CICS | Customer Information Control System |
| CICS OS2 | Customer Information Control System Operating System/2 |
| CPU | central processing unit |
| CSP | Cross System Product |
| DB2 | Database 2 |
| DBCS | double-byte character set |
| DBMS | database management system |
| DCE | distributed computing environment |
| GUI | graphical user interface |
| IBM | International Business Machines |
| IMS | Information Management System |
| LAN | Local Area Network |
| MSL | member specifications library |
| MVS | Multiple Virtual Storage |
| NT | Notes |
| OS/2 | Operating System/2 |
| OS/400 | Operating System/400 |
| RAD | rapid application development |
| SQL | Structured Query Language |
| TCP/IP | Transmission Control Protocol/ Internet Protocol |
| VM | Virtual Machine |
| VSE | Virtual Storage Extended |
| WWW | World Wide Web |

# The VisualAge Generator Newsletter

This newsletter is published by the IBM Software Solutions Division, Research Triangle Park Development Laboratory. Letters to the editor are welcome. Please address correspondence to:

**The VisualAge Generator Newsletter**
**Managing Editor**
**IBM Corporation**
**Dept. T22/062**
**P.O. Box 12195**
**3039 Cornwallis Road**
**RTP, NC 27709-2195**
**USA**
**FAX: (919) 254-0206**



**The VisualAge Generator Newsletter**
**IBM Corporation**
**Dept. T22/062**
**P.O. Box 12195**
**3039 Cornwallis Road**
**RTP, NC 27709-2195**
**USA**

G242-0315-07