The IBM VisualAge Generator Newsletter

# VisualAge® Generator

A Powerful New Vision of Programming™

Volume 2, Number 2
April 1997

## Contents



Formerly the VisualGen Newsletter

# With a Little Help From Your Friends

*by Rich Hagopian, Manager, VisualAge Generator GUI Development*

The goal of this newsletter is to share ideas and techniques that we believe will help you get the most out of VisualAge Generator. The need to achieve even higher levels of productivity continues to drive each of us. Year-2000 support, e-commerce, network computing, and an increasing demand for rapid development of applications are just a few of the many challenges faced by the professional developer.

I am happy to say that this issue is full of information to help you address some of these challenges. Topics range from Java enablement to Year-2000 support. You will also find exciting announcements about products and services, such as the VisualAge Generator Templates feature and the VisualAge Generator Consulting Services, which is designed to assist the VisualAge Generator developer in achieving new levels of productivity. VisualAge Generator Templates is a new orderable feature of VisualAge Generator Version 2.2. It is designed to automate the repetitive coding required to perform non-business logic functions that are commonly required by all systems. Take a look at the “Rapid Application Development Using VisualAge Generator Templates” article on page 5 to see how you can use this new feature to more rapidly deliver applications. I think you will agree that this represents an exciting step forward for VisualAge Generator as a rapid development tool.

The “VisualAge Generator Consulting Services” article describes the establishment of the VisualAge Generator Consulting Services organization. The group, housed with the VisualAge Generator development team, is available to assist current and prospective VisualAge Generator and Cross System Product customers in areas ranging from the setup of the development environment to the deployment of enterprise client/server systems.

A little help from your friends, a services organization dedicated to VisualAge Generator services, and VisualAge Generator Templates, all aimed at providing you with the solutions you need to meet today’s and tomorrow’s application development challenges.



## Corrections

1. The previous issue of the newsletter stated that the support expiration date for V2.2 was 12/31/97. The support date has been extended; the new support expiration date is 12/31/98.
2. We stated in the previous issue of the newsletter that Sri Lanka was the smallest country in the world. This was incorrect, and we apologize to anyone we might have offended.

# Year 2000! Help Is On the Way

*by Allan DeLoach, Application Development Sales*

As we are all aware, the year 2000 will soon be here, bringing with it a unique challenge for many computer programs— recognizing and representing dates in two different centuries.

There are three aspects to the Year 2000 support with the VisualAge Generator and CSP products:

- Dates associated with members created and edited within VisualAge Generator or CSP
- Built-in date-related functions invoked by user coding that manipulate date values
- Date data manipulated by applications developed with VisualAge Generator or CSP but not dependent on specific VisualAge Generator or CSP functions

Of the three, the latter one, application-specific date data, is the area that will require the most time and effort to ensure that it works correctly with dates beyond 1999.

This article provides a brief overview of the challenges associated with supporting 4-digit years in VisualAge Generator and CSP applications. You can find more details in the white paper, *VisualAge Generator, CSP, and the Year 2000,* plus a scanning tool (ESFSCAN). Copies of the white paper and the date scanning tool are available from the IBM Year 2000 web site:

`http://software.ibm.com/year2000`

Your IBM sales representative can also obtain copies of the white paper and the tool from the MKTTOOLS disk (ADVGYR2K PACKAGE). VisualAge Generator and CSP 4.1 are considered by IBM to be Year-2000 ready, provided you have the most current maintenance level. CSP 3.3 is Year-2000 tolerant in that it executes correctly on systems after 1999, provided you have the most current maintenance level. Refer to documentation APAR II09591 for the latest maintenance requirements for CSP 3.3 and Year-2000 support.

## Member Dates

In VisualAge Generator, CSP 4.1, and CSP 3.3 member dates shown in a member list are displayed using a 2-digit year format. The earliest date that a member was created is 1980, so from context you can tell which year a member was last saved.

In VisualAge Generator, when members are displayed in date sequence, members created after 1999 are displayed in the correct sequence even though only the last two digits of the year are displayed.

## Date Functions

VisualAge Generator and CSP 4.1 provide EZE words that return the current date with both 2-digit and 4-digit year formats. CSP 3.3 date function EZE words only support 2-digit year date formats.

VisualAge Generator and CSP 4.1 provide map edit date masks for both 2-digit and 4-digit year formats. CSP 3.3 map date edit masks only support 2-digit year formats.

There are no plans to add 4-digit year current date EZE words or map date edit masks to CSP 3.3. Installations with CSP 3.3 should give serious consideration to migrating to VisualAge Generator to gain current date functions and map date edit mask support for 4-digit year formats.

## Identifying 2-digit Year Exposures

To identify the potential exposures caused by using 2-digit year dates, you first need to locate all references to all date-related data items. Locating date-related data items is itself a major part of any Year-2000 effort.

The two ways to search VisualAge Generator and CSP members for date data item references are:

- Using the ***where used*** utility
- Exporting members and scanning the export file

The ***where used*** utility enables you to search the members in a member list for references to a particular member (that is, a data item). When invoked, the utility displays a list of all members containing references to the item.

You cannot use the ***where used*** utility to search for EZE words (for example, EZEDTE), and you can only search for one member name at a time. These and other restrictions limit the ***where used*** utility usefulness for broad searches for date references in members. Because of this scanning task, an export file might be a more useful approach. A sample ESF scanning tool (ESFSCAN) is available from the IBM Year-2000 web page:

`http://software.ibm.com/year2000`

Again, your IBM sales representative can also obtain copies from the MKTTOOLS disk (ADVGYR2K PACKAGE).

# Reformatting Year-Date Notation

There are three fundamental approaches to handle Year-2000 dates within your installation and applications:

- Conversion to full 4-digit year format

  This solution requires changes to both data and the programs by converting all references and uses of 2-digit year format (YY) to 4-digit year format (YYYY). This is considered to be the only complete, permanent, and obvious solution.

- Windowing techniques

  This is a 2-digit solution that externalizes either 2-digit or 4-digit year formats. This approach only require changes to programs, not to data. However, it cannot be used if dates outside of a 100 year range must be maintained.

- 2-Digit encoding/compression techniques

- This is a 2-digit solution that externalized either 2-digit or 4-digit year formats. It requires changes to both your data and your programs, but it does not increase the size of external files. It also requires that you convert, simultaneously, all applications that reference or use the encoded date fields.

A full discusion and examples of these approaches can be found in the *VisualAge Generator, CSP, and Year 2000* white paper.

The techniques you adopt to handle Year-2000 dates within your installation is determined by many factors. If files containing 2-digit year date fields are shared between VisualAge Generator or CSP applications and programs written in other languages, you must consider the capabilities of all the languages used to process the files when evaluating solutions.

The solutions and techniques that you can use to correct Year-2000 exposures are not technically difficult. However, you will likely find that most, if not all, programs and files in your installation will have to be examined and perhaps changed. Thus, the major challenge in a Year-2000 project is the project management.

# Testing

By nature, Year-2000 exposures are time-sensitive and time-driven. Basic Year-2000 testing requires that you set the system date and time so that Year-2000 problems are found. Changing the system date to permit testing CSP applications can be difficult and usually requires running a separate copy of the host operating system as a VM guest or LPAR. However, VisualAge Generator can provide a safe Year-2000 test environment because you can change a workstation's system date to future dates and not affect other users, systems, and files. For this reason, you should consider migrating CSP applications to VisualAge Generator as part of your Year-2000 conversion effort.

# Summary

VisualAge Generator is considered by IBM to be Year-2000 ready; VisualAge Generator provides both 2-digit and 4-digit year format functions. Applications developed in VisualAge Generator can be tested in isolation on a workstation for Year-2000 exposures.

CSP 4.1 is considered by IBM to be Year-2000 ready. CSP 4.1 provides both 2-digit and 4-digit year format functions. However, testing CSP 4.1 applications for Year-2000 exposures might be difficult because it is dependent on the host operating system environment and files.

CSP 3.3 is considered by IBM to be only Year-2000 tolerant. While CSP 3.3 will execute after the year 1999, it does not include any support for 4-digit year format functions. In addition, testing CSP 3.3 applications for Year-2000 exposures might be difficult because it is dependent on the host operating system environment and files. If you are currently running CSP 3.3, you should seriously consider migrating to VisualAge Generator as part of your Year-2000 effort.

# Rapid Application Development Using VisualAge Generator Templates

*by Stefano Sergi, Application Development Sales and Technical Support*

Despite an abundance of disciplines, technologies, and tools devised to ease the production of business application software, development managers continue to be faced with increasing backlogs. Software development remains largely a manual, craftwork-like process rather than an automated, rigorous and predictable engineering discipline. *Application frameworks* and *template technologies* relieve programmers of coding repetitive sequences of instructions necessary to perform application functions that are not specific to the business problem, but are commonly required by all systems.

VisualAge Generator is evolving in this direction to provide an automated software production workbench. This evolution begins with VisualAge Generator Templates V2.2.

## VisualAge Generator Templates

VisualAge Generator Templates is a new feature of VisualAge Generator that provides programming with true Rapid Application Development methodology and data-driven automated code generation, which when combined with the flexibility, scalability, and portability of the base product, constitutes IBM's premier high-productivity solution.

Using a windowed graphical interface, the application developer can build a fully functional database manipulation application in a matter of minutes. The typical development effort consists of creating instances of the VisualAge Generator Templates Information Model by specifying what data (tables/views and columns) the application will manipulate, and how the data presents itself to the user. VisualAge Generator Templates automatically generates all the VisualAge Generator source code components necessary to do the following:

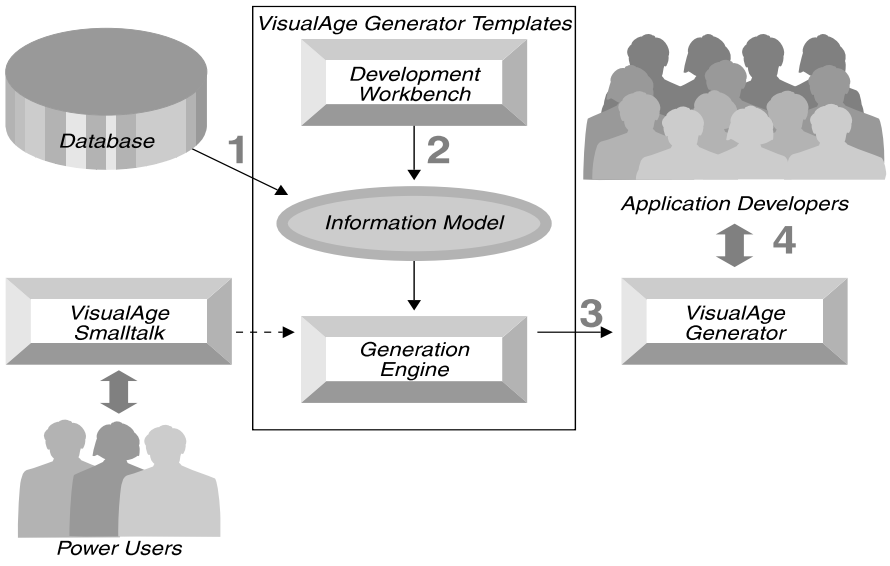
- Access and manipulate database (Create, Read, Update, Delete)
- Present the data to the end-user
- Manage navigation among multiple windows
- Manage multi-user data access concurrency
- Manage paging when data result set is larger than page size
- Manage error conditions
- Provide window and field level online help

The generated source is well structured, consistent, and error free. It also includes hooks to easily add specific business logic using VisualAge Generator. Additions to the generated components are preserved, thus providing the ability to maintain the functional specifications at the Information Model level.

The VisualAge Generator Templates feature is architected to be fully open to enable any degree of customization of the generated code. An open API combined with the interactive specification facilities provides for easy extensions of the base Information Model. This enables developers to incorporate and automatically enforce site standards in the generated systems, and to further extend the already rich base functionality.

Using VisualAge Generator Templates, software developers can deliver robust, high quality, and fully functional systems in a fraction of the time required by manual VisualAge Generator coding. The automatically generated application infrastructure easily makes up 70 to 80 percent of the code required for the final application. This enables the developers to focus on the remaining 20-30 percent business logic specific code.

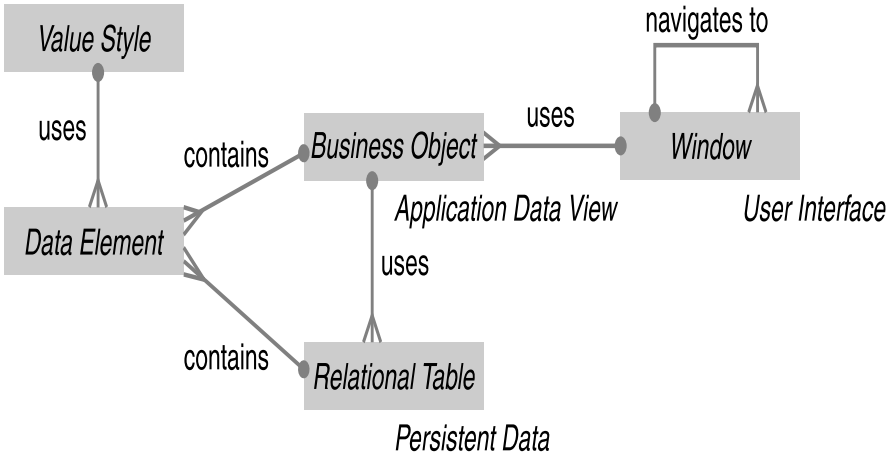
The following figure shows VisualAge Generator Templates' components and identifies the four steps of application development.



VisualAge Generator Templates has the following three components:

- An Information Model

  The Information Model shown in the following figure is the heart of VisualAge Generator Templates. The Information Model provides the methodology to specify a database application at a logical level through a set of entities. Persistent data is represented by *Relational Tables* and relative *Data Element* entities. The application view of the data is represented by *Business Object* entities. Each business object manages a set of data elements from one or more relational tables. The application interface through which business objects are presented to users is represented by the *Window* entities. Each window can contain one or more business objects and each business object can appear to the end-user as a list or individually in detail.




- A Development Workbench

  This is the windowed graphical environment the programmer uses to create instances of the Information Model entities relative to a specific business application.

  The Development Workbench is also equipped with a facility to automatically generate Relational Tables and Data Elements instances from existing databases.

  An important aspect of the VisualAge Generator Templates architecture is the separation of functional specifications from technological specifications. For example, during the creation of a new instance of the *Business Object* entity, the ***functional specification*** is the definition of the tables and data elements that the Business Object work with. The ***technological specifications*** are used to select presentation choices such as the number of fields shown in a notebook page, or whether popup menus should be used to enable user actions instead of push buttons or action bar choices.

  This separation enables you to define technological specifications at the entity level, so that the definition of all instances of that entity only requires the specification of functional level information, further enhancing programmers productivity.

- A Generation Engine

  The Generation Engine is the facility that transforms the Information Model specifications into VisualAge Generator applications including client and server components. Default layout of the GUIs are produced by the Generation Engine to rapidly obtain correct presentations. Modifications and addition of business logic can be performed using VisualAge Generator, and the default layout is preserved if subsequent modifications of the logical specification are made in VisualAge Generator Templates.

  The Generation Engine can be invoked from the Development Workbench. At the end of this generation step, the VisualAge Generator MSL is directly populated with the generated source components from the VisualAge Generator Templates Development Workbench.

  The Generation Engine is a VisualAge for Smalltalk open framework. Using VisualAge for Smalltalk, the Generation Engine can be extended to customize the existing generator functions or even to create new additional generators. This powerful architectural characteristic is usually exploited by specialized users (power users as shown in the next figure) to set up the Model infrastructure to establish corporate standards and occasionally to fulfill new code generation requirements.

  *Customizing a VisualAge Generator Templates generator* is creating a Smalltalk class that inherits from the class of the standard generator that needs customization and redefinition of the standard methods whose behavior needs to be modified.

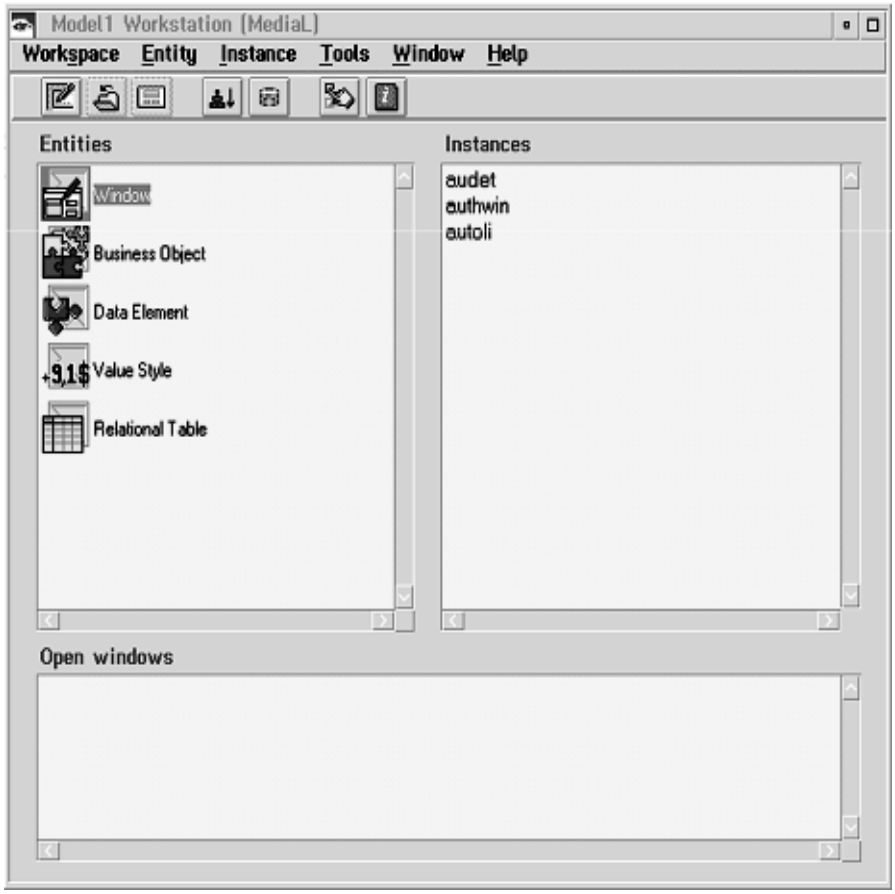
*Creating a new VisualAge Generator Templates generator* is writing a Smalltalk class that inherits directly from the superclass of all VisualAge Generator Templates generators and then defining the class behavior using a specialized VisualAge Generator Templates Smalltalk API.

# A Walk Through VisualAge Generator Templates

An example will help to clarify the VisualAge Generator Templates concepts and facilities. Let's say for instance, we want to define a simple application that displays a list of book authors, add new authors, retrieve and display detail information for specific authors, update the information, or delete authors. To have VisualAge Generator Templates generate all the code to perform these functions, we need to do the following:

- Define all the Relational Tables and Data Elements

  We can accomplish this function automatically if the tables are already defined in the relational database. We can invoke the VisualAge Generator Templates utility from the **VisualAge Generator Templates Main Window** (as shown in the following figure), retrieve all of the tables information from the database catalog and create equivalent instances of the VisualAge Generator Templates entities *Relational Table* and *Data Element*.



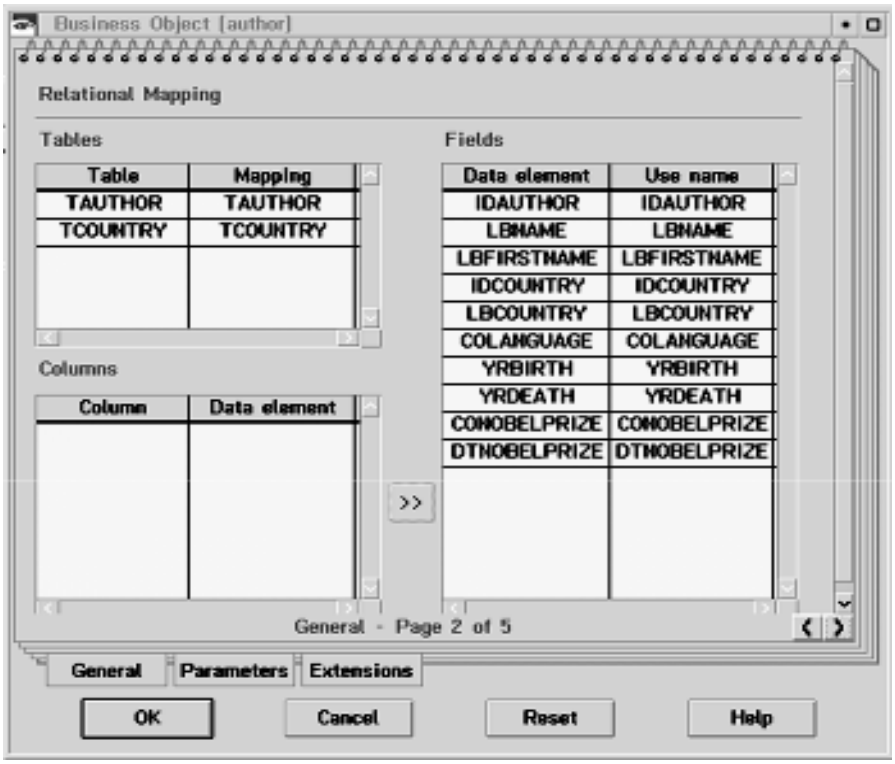

- Define an *Author* Business Object.

  This function consists of selecting what fields the Business Object contains and the tables the fields belong to. A simple notebook interface guides us through the selection of the relational tables and data elements contained in this Business Object. For example, included in the *Author* Business Object might be some data elements from the *Authors* relational table and some data elements from the *Country* relational table.

  We can also define an extraction criteria, which is used by VisualAge Generator Templates to generate the code that retrieves the list of authors to display.

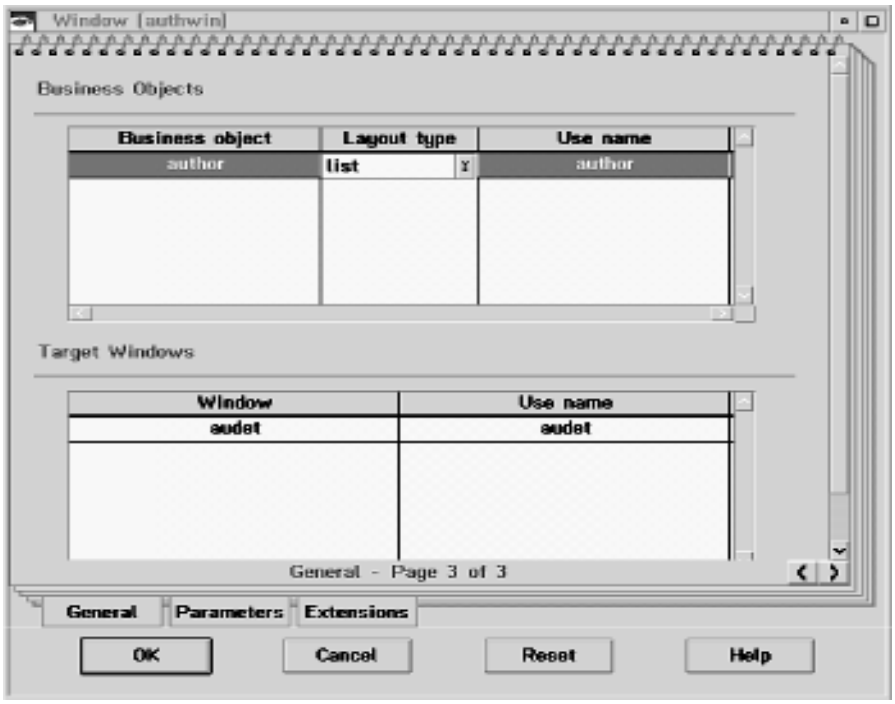
  A separate notebook tab (parameters) enables us to override the default technological specifications associated with all Business Objects, which we might have previously specified at the entity level.

For example, we can display the Business Object detail information organized into a notebook, instead of the default which would display them in a window. We can change the default labels that allow the user to page through the list, and we can limit the list view to the first two data elements contained in the Business Object instead of showing a list with as many columns as there are data elements. The following figure shows a typical dialog used to define a VisualAge Generator Templates Business Object definition.




- Define the Windows

  We can have the choice of showing the list and detail views of the *Author* Business Object through one or more windows. If we choose to use a single window, we simply define a new *Window* instance. Using the familiar notebook interface, we can specify that the single window contains the Business Object *Author* in list form, and we can specify the Business Object *Author* again in detail form. If we choose to use two windows, we can define two new instances of the *Window* entity. The first one containing the *Author* Business Object in list form, the second one containing the *Author* Business Object in detail form. We can also specify navigation information from one window to another. The following figure shows a typical dialog used to define a VisualAge Generator Templates *Window*.




- Generate and populate an MSL

  From the VisualAge Generator Templates main window, we can invoke the Generation Engine for the first window that the application displays to the end-user, and a propagation option instructs VisualAge Generator Templates to also generate all dependent *Windows* and *Business Object* instances.

  Once the user is notified that generation completed successfully, an action bar choice enables the user to trigger the import of the generated source in a designated MSL.

  Without hand-coding a single line of code, 2 Applications, 10 Records, 9 GUIs, 28 Processes, and 23 Statement Groups were just tested.

# Conclusion

The addition of VisualAge Generator Templates to the VisualAge Generator product significantly reduces the amount of hand-crafted code required to deliver database business applications.

VisualAge Generator Templates will help application developers deliver systems that are:

- Adherent to corporate standards
- Robust and fully functional
- In record time
- Higher quality

Thus, facilitating the transition of software production from craft to the industrial engineering discipline.

# Java and VisualAge Generator

*by Michael Rhoads, Manager, World Wide Application Development Solutions*

If you are wondering when VisualAge Generator will add Java support, then wonder no longer. We have some exciting things coming your way.

## Have You Ever Wished Upon a Star?

- Are you intrigued by a thin *Java* client, but need a rapid application development (RAD) solution to develop your enterprise server applications?
- Have you ever wanted to leverage the power you get from a proven object-oriented (OO) fourth-generation language (4GL) in your emerging Java world?
- Would you like to rapidly develop server applications for MVS, VM, VSE, AIX, Windows NT or OS/2 and then generate a Java Bean for it and all its client to server communication code?
- Would it help you leverage Java if you could "quick-form" a Java Bean representation of your VisualAge Generator server applications from within VisualAge for Java?
- Have you ever thought of using intelligent templates to manufacture applications, combining Java clients and VisualAge Generator servers?
- Would you like the flexibility to combine any vendor's Java development tools with a Java function for IBM's VisualAge Generator?

## Hold onto Your Seat—Java Support is Just Around the Corner!

VisualAge Generator will soon have new functions that will provide *Java support* capability. You will be able to combine VisualAge Generator's proven OO-4GL client/server rapid application development (RAD) capability with the new Java generation functions.

Let's take a look at Network Computing, the value of a Java support solution, and the applications you will build with this new function. Then we will take a look under the hood at how this really works, when this new function is available, and what to look forward to.

## Keep Your Hat On—It's Network Computing Gone Wild!

Network computing will become the dominant topology of our age. It has already captured our imagination and is quickly transforming the world we live in. Today's systems are being linked together or transformed in ways never imagined before! Tomorrow's systems will include new and existing systems connecting customers, suppliers, and partners in new and exciting ways.

As we transition from browsing to business, the things we will do on the internet will go far beyond spinning heads, wild web sites, and big chat rooms. As real business moves to the web en masse, everyday practical tasks will be accomplished on the web. We will buy and sell products of all kinds. Banking, entertainment, education, and all sorts of other activities will become commonplace on the web. We will log-on and expect to be able to do things instantaneously!

Imagine one billion people all having access to the internet. We can quickly see that the supporting applications will need to be:

- Scalable
- Reliable
- Secure

These traits are the pillars of much of the world's server applications today. They have been addressed through optimized code, transaction monitors, and optimized databases. Our network-enabled world will need to combine these server traits with the emerging client topology. The defining client access medium will be the browser and Java technology, which plays a key role.

## Tie Down Your Tent—Java Support Gives You New Capabilities!!

VisualAge Generator's new Java function is opening all sorts of new opportunities. You can build your server applications once, along with all the communication middleware using VisualAge Generator. Then through VisualAge Generator, you can automatically generate the Java Bean representation or wrapper for this server application. Then, you can connect or consume this Java Bean from any of a number of Java client development environments.

## New Application Types

From an application perspective, this means you can instantly bring the scalability, reliability, and security of VisualAge Generator's proven server development capability to the portable web-enabled world of Java applets and applications.

For instance, you can build new web-enabled banking applications that leverage a browser-based Java client connected to new or updated banking transactions on the server. These new transactions run in an existing, high-performance, reliable, and secure infrastructure.

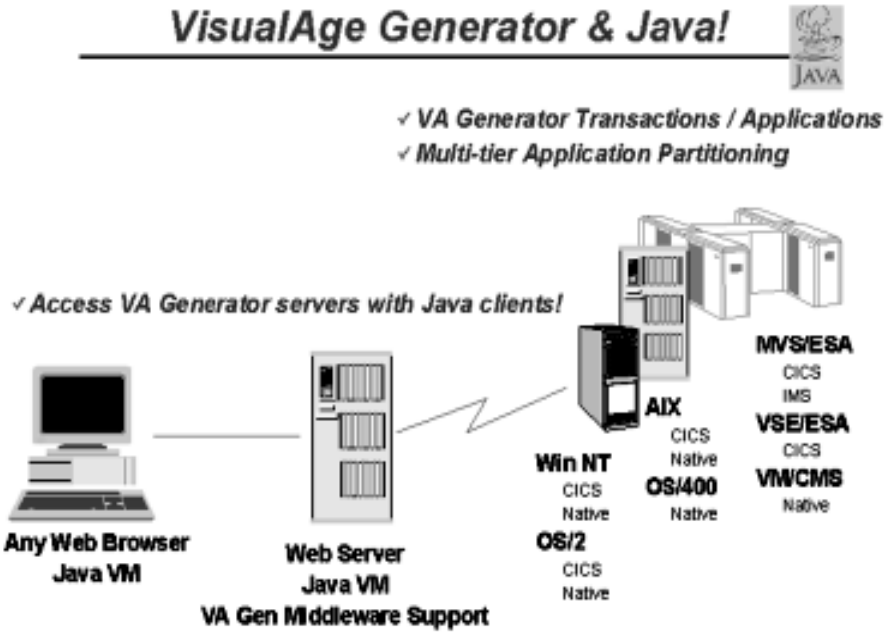
It's a marriage between two key technologies (Java and VisualAge Generator) to make enterprise oriented internet-enabled business applications a reality today!

## Productive Development Environment

You and your programming teams also benefit from the marriage of Java and VisualAge Generator technologies. With VisualAge Generator you will be using award-winning visual programming technology and an OO-4GL to rapidly develop AND functionally test the server application entirely from the desktop. You can use state-of-the-art business object frameworks or templates to improve your productivity and quality even more (see VisualAge Generator Templates article in this Newsletter issue).

VisualAge Generator gives you ultimate flexibility in your deployment choices. For example, you can first build your server application and deploy on AIX. Later, you can scale-up to MVS or down to Windows NT without changing your logic. Or, you can first deploy using an IMS/TM transaction monitor for MVS and later change to CICS on the same or different system, again without changing your logic. The OO-4GL provides you with a level of abstraction, which shields you from the complexities of the execution environment.

As shown in the figure below, you can develop and connect to a whole range of server platforms.




Once you have completed your iterative build and test of the server applications, VisusalAge Generator can automatically generate the associated Java Bean. This bean wrappers the server logic and communication code. You can then use a product like VisualAge for Java to build your client logic. You can even "quick-form" the Java Bean from within VisualAge for Java to visually code the client connections. Now that's productivity.

## This Car's Ready to Roar—Let's See What's Under the Hood

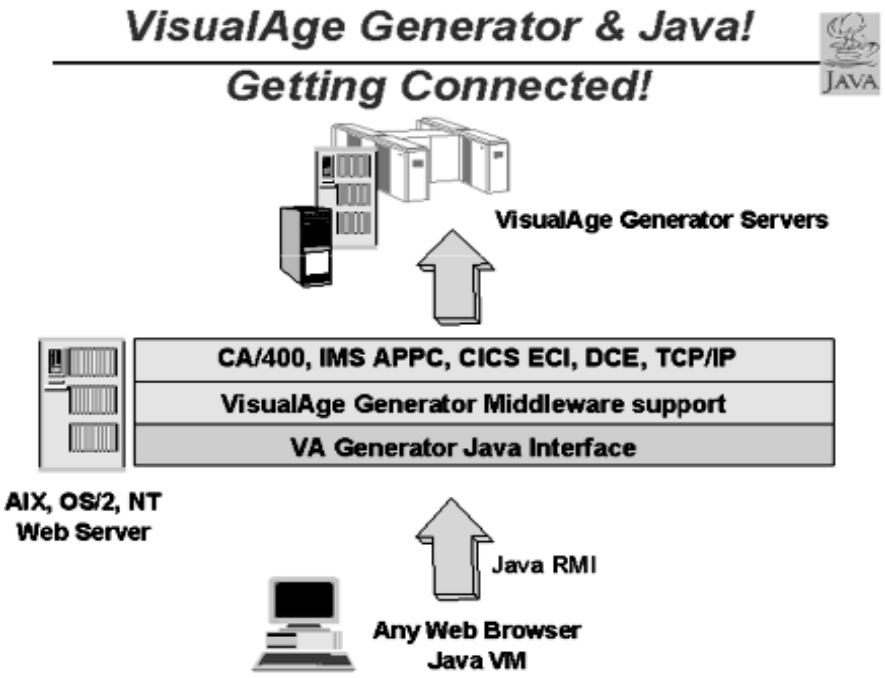
Let's take a closer look at how we are making this work...

We use the VisualAge Generator server interface definition as input, then we automatically generate Java Bean classes that "wrapper" calls to server transactions. Standard Java 1.1 Beans are created and, therefore, they can be consumed by any Java 1.1 compliant catcher application. Just use these beans and connect them directly to your Java client applets or applications using tools like VisualAge for Java, or some other vendor's Java development environment.

The wrapper classes handle all aspects of communicating with the enterprise server, including marshalling data from objects to server record and database structures, converting data between Java and host formats, and controlling commit/rollback for extended units of work for multiple server calls within a transaction.

Communication with server systems use a middle-tier Java server running on NT, AIX, or OS/2. The Java applet communicates with the middle tier server using Java remote method interface classes; the middle tier server communicates with the VisualAge Generator server transaction via the VisualAge Generator POWERserver middleware. You don't need to do any coding for the middle tier. This is generated for you.

As shown in the figure below, the Java interface layer handles the marshalling and converting of data as well as commit/rollbacks of server calls within transactions. The layers above represent the communications you get with VisualAge Generator today.




## Put Your Seatbelt On—Java Support Capability is Coming Fast!!

The VisualAge Generator Java function will be released real soon. Within our development labs we are measuring ourselves in "Web-Years". Three calendar months is considered one "Web-Year". This Web-Year timing is a means of reminding us how quickly new web-enabled technology is coming to the market.

Our current product is VisualAge Generator 2.2. We expect to make our Java support function available in VisualAge Generator in June of this year!!!

In a recent issue of this newsletter, we described how you could build new CICS or AS/400 server applications with VisualAge Generator and very quickly deploy the user interface on the Web. This approach used our Gateway technologies (CICS Internet Gateway or AS/400 Gateway) to capture the data streams and convert them to HTML.

Now we are proud to bring you this update on our Java client access support. It's coming to you in the "Web Years"!

## Hold On to the Reins—There's More Java and Web Support to Come

While the upcoming level of Java support will provide significant benefits to you, your staff, and your end-users, it is not the end of our planned Java and Web support. The Network Computing initiative is a revolution. VisualAge Generator is in a unique position to be able to add new Web function in ways that bring real business value to you and the industry.

Other VisualAge Generator Web support functions under development or research include Java applet authoring and web page authoring functions. We are also working on possible Java client extension to VisualAge Generator Templates. And down the road, we are considering extensions to our Java generation capabilities.

We are working in “Web Years”. Very soon you will see our new VisualAge Generator Java support. This capability extends your web support beyond the data stream HTML conversion of our Gateway technologies. A few short “Web Years” later you will see additional VisualAge Generator Java and Web support functions rolling out.

## This Car’s Racing Down the Track—And You’re the Winner!

It’s truly an exciting time to be in our industry. Network Computing is sweeping through our landscape. It’s bringing a lot of changes, but it’s opening great new possibilities.

By combining robust state-of-the-art OO-4GL approaches with leading-edge web technology, you will be able to make the promise of real business on the Web for the masses a reality.

VisualAge Generator and its Java Web support will help you harness the productive power you need to WIN!

## VisualAge Generator Events Update

- The third Great Lakes Area VisualAge Generator Users Group meeting was held in Columbus, Ohio the week of March 24.
- The third annual Germany VisualAge Generator Users Group meeting was held in Essen, Germany in March.
- The first Hungarian VisualAge Generator hands-on workshop was held in Budapest in March.
- In California, VisualAge Generator was recently demonstrated at the Gartner Show and Expo, Software Developers Conference, and CICS Technical Conference.
- Several VisualAge Generator business partner update sessions were held in February, including sessions in Italy, Australia, and Korea.
- VisualAge Generator is demonstrated at the IBM VisualAge Forum Seminars. The first show was held in Phoenix, Arizona earlier this year. This Forum is being held in cities throughout the world.
- VisualAge Generator Update was held at the Integral Users Group conference in Tulsa in February.

# Optimizing Data Transmission

*by Jim Eberwein and Alex Akilov, VisualAge Generator Development; Judy Hansen, ConAgra Trading and Processing; and Ted Borawski, Software Marketing*

With the onset of client/server technology, application developers are faced with a wide range of development tools that promise to make their job easier. Although these tools may make it easier to develop applications, the developers must realize that good programming practices are still required to produce applications that will not place a burden on the enterprise network. This article documents two programming practices that you can use to develop VisualAge Generator applications.

## Client/Server Design Philosophy

In the enterprise client/server environment, both logic and data is distributed in various locations across the network. Due to this system architecture, runtime performance is affected by factors such as bandwidth and network transmission times. Client/Server application programmers must recognize this, and therefore design their systems in a manner that minimizes the amount of data that is sent across the network when client and server programs communicate with one another.

By minimizing the amount of data transmitted across the network, the application system improves the overall response time of the system. VisualAge Generator allows application programmers to develop various techniques that adhere to this programming philosophy. Two such techniques available to the programmers are *packeting* and *null suppression*.

## Packeting

Packeting is a technique that is used in conjunction with the *Container Details (CDV)* object defined in a VisualAge Generator GUI client. The *CDV* is an object that displays data in a tabular format. The general way to build is to require all of the CDV contents to be retrieved prior to the CDV displaying the data.

Packeting removes this restriction, and instead only requires a portion of the data to be displayed. As you scroll through the CDV contents, additional requests for data are issued from the CDV to logic member parts defined in the GUI client. This allows large amounts of data to be viewed only when needed instead of getting it all up front.

## CDV Example

An example of how to use the CDV object follows:

1. Open a new GUI definition. The window of this GUI contains the CDV.
2. From the *Lists* category, select a *Container Details* object and place it in the *window* part.

   The CDV now appears in the window.
3. From the *Logic Member Parts* category, select a *Statement Group Member Part* object and drop it on the GUI's free-form surface. Name the object CDVINIT.

   The statement group initializes the CDV for packeting.
4. Connect the *AboutToOpenWidget* event of the window to the *execute* event of CDVINIT.

   This initializes CDV.
5. Connect the *hasExecuted* event CDVINIT to the *packetEnabled* attribute of the CDV.

   An incomplete connection appears.
6. Double-click on the incomplete connection and set the connections value parameter.

   The *Constant Value Parameter Settings* window shows a Value check box. Verify this is checked to complete the setting.
7. From the *Logic Member Parts* category, select a *Statement Group Member Part* object and drop it on the GUI's free-form surface. Name the object CDVPAGE.

   The statement group executes when the CDV requests data.
8. Tear off the *packetattribute* of the CDV.

   This gives you access to the start and ending rows that are currently being displayed in the CDV, plus access to the entire contents of the CDV's data rows.
9. Connect the *packetRequested* event of the CDV to the *execute* action of CDVPAGE.

   The statement group executes when the CDV requests new data to be displayed.
10. From the *Data Member Parts* category, select a *Record Member Part* object and drop two objects on the GUI's free-form surface. Name the objects CDVRCD1 and CDVRCD2

11. Double-click on the CDVRCD1 working storage record and define the following structure:

| Name | Level | Occurs | Type | Bytes |
|---|---|---|---|---|
| ROWSEND | 10 | 1 | Bin | 2 |
| NEWSTART | 10 | 1 | Bin | 2 |
| NEWEND | 10 | 1 | Bin | 2 |
| ROWS | 10 | 500 | Char | 20 |
| ROWNUM | 15 | 1 | Bin | 2 |
| COL2 | 15 | 1 | Char | 10 |
| COL3 | 15 | 1 | Char | 8 |

12. Double-click on the CDVRCD2 working storage record and define the following structure:

| Name | Level | Occurs | Type | Bytes |
|---|---|---|---|---|
| INDEX | 10 | 1 | Bin | 2 |
| END | 10 | 1 | Bin | 2 |
| COUNT | 10 | 1 | Bin | 2 |
| I | 10 | 1 | Bin | 2 |
| TOTALROWS | 10 | 1 | Bin | 2 |
| PAGESIZE | 10 | 1 | Bin | 2 |
| ROWS | 10 | 50 | Char | 20 |
| ROWNUM | 15 | 1 | Bin | 2 |
| COL2 | 15 | 1 | Char | 10 |
| COL3 | 15 | 1 | Char | 8 |

13. Connect the *ROWS* attribute of the CDVRCD2 record to the *items* attribute of the CDV.
14. Select the CDV and, from its context menu, choose the option *Initialize Columns Based On Connections to #items.*

    This initializes the CDV to display columns ROWNUM, COL2, and COL3
15. Delete the connection *ROWS* of CDVRCD2 to *items* of the CDV.

    This connection is no longer necessary.
16. Connect the *startRow* attribute of the packet object to the *NEWSTART data* attribute of the CDVRCD1 record.

    Each request for a new set of rows automatically assigns the starting row of the current page.
17. Connect the *endRow* attribute of the packet object to the *NEWEND data* attribute of the CDVRECD1 record.

    Each request for a new set of rows automatically assigns the ending row of the current page.
18. Tear off the *ROWS* attribute of the CDVRCD1 record.
19. Connect the *has executed* event of CDVPAGE to the *getFieldsStartingAt To* action of the *ROWS tear-off* attribute.

    NEWSTART and NEWEND is used as the indexes into the array.
20. Connect the *NEWSTART data* attribute of CDVRCD1 to the *firstIndex* parameter of the event to action connection.
21. Connect the *.NEWEND data* attribute of CDVRCD1 to the *lastIndex* parameter of the event to action connection.
22. Connect the *dataRows* attribute of the *packet* object to the *result* parameter of the event to action connection.

    This completes this connection.
23. Connect the *PAGESIZE data* attribute of CDVRCD2 to the *packetSize* attribute of the CDV.

    A packet request occurs when you scroll past the page size.
24. Connect the *TOTALROWS data* attribute of CDVRCD2 to the *totalRows* attribute of the CDV.

    This displays the maximum number of rows that the CDV can manage.
25. Edit the CDVINIT statement group and add the following logic.

```
TOTALROWS = 500;
PAGESIZE = 50;
```

26. Edit the CDVPAGE statement group and add the following logic.

```
/* Check to see if the rows     */
/* have already been retrieved. */
/* If not, then make call to    */
/* server.                     */
IF NEWSTART > ROWSEND
  OR NEWEND > ROWSEND;
  INDEX = ROWSEND + 1;
  CALL CDVAPP CDVRCD2 (REPLY;
  MOVEA CDVRCD2.ROWS TO DVRCD1.ROWS(INDEX);
  ROWSEND = INDEX + COUNT - 1;
END;
```

27. Save the GUI definition and name it CDVGUI.

In this next example, the server application CDVAPP hard codes the values that are assigned to the array. In actual business applications, the server application needs logic to perform database input/output. The server should be coded to determine if previous calls have been made and, if so, then retrieve the next row following the last row retrieved on the previous call.

1. Open a new APPL definition.
2. Add record CDVRCD2 as a called parameter.
3. Define the application as a *Called Batch*
4. Use the *Save as* option and name the application CDVAPP.
5. Add a main process called CDVPRC1. The process type is EXECUTE.
6. Define the following logic in CDVPRC1:

```
I = INDEX;
COUNT = 1;
END = INDEX + PAGESIZE - 1;
IF END > TOTALROWS;
  END = TOTALROWS;
END;
WHILE I <= END;
  MOVE I TO ROWNUM(COUNT);
  MOVE 'ABC' TO COL2(COUNT);
  COUNT = COUNT + 1;
  I = I + 1;
END;
COUNT = COUNT - 1;
```

## Summary of the CDV Example

You are now at the point where the client and server programs are ready to be tested. A brief summary of the application design follows.

Prior to the client GUI opening, the CDVINIT initializes the CDV, setting the total number of rows that CDV can manage and the maximum number of rows that are returned on a packet request. After the window opens, the CDV issues a request for data. This triggers the execution of the CDVPAGE statement group. The CDVPAGE statement group must determine if a call should be made to the server application to retrieve additional rows, or if the rows the CDV is trying to display are already stored in the CDV's packet. After CDVPAGE executes, the correct index into the CDV is shown. The index is determined by looking at the NEWSTART and NEWEND variables.

If the total rows size is set too small, you can modify the GUI definition to store the packet in an ordered collection. Additional logic then needs to be coded that can determine what packet should be currently shown in the CDV.

# Null Compression

This technique is a little known feature within CICS. This feature compresses the trailing null values (X'00') from the end of the CICS Communications Area (COMMAREA), which greatly reduces the amount of data transmitted on the network, thereby improving response time. CICS compresses the null values from the COMMAREA by starting at the end and working backwards until the first non-null value is found.

CICS/OS2 checks one character at a time while the host checks blocks of 256 characters at a time. If you do not pad with nulls, the entire COMMAREA (up to 32,763 bytes) is transmitted. This is unnecessary and affects the overall response time of the transaction and potentially the entire network, depending on bandwidth, line speed, and current network traffic.

VisualAge Generator can utilize the null compression feature of CICS but requires up-front design work of the COMMAREA. Proper record definition, initialization, and placement of records within the CALL statement provide the framework necessary to take advantage of this null compression feature.

Several alternatives to this technique have been investigated by VisualAge Generator customer ConAgra Trading and Processing.

# Record Definition

To initialize the records with binary zeros, the record definition needs to be substructured so that either the highest-level data item or lowest-level data item has a HEX data type. This data item then needs to be initialized via either a MOVE or the SET RECORD EMPTY statement. Note that depending upon how the record is structured, the correct initialization statement must be used.

The SET RECORD EMPTY statement initializes the data item based on the data type of the lowest level. Thus, use the SET RECORD EMPTY statement when the HEX data items are used in the lowest level, and use the MOVE statement when the HEX data item encompasses the entire record structure. The following two examples illustrate the definitions of these structures:

**Record XXX**

| Name | Level | Occurs | Type | Bytes |
|---|---|---|---|---|
| Field1 | 05 | 1 | Char | 20 |
| * | 10 | 1 | Hex | 20 |
| Field2 | 05 | 1 | Char | 25 |
| * | 10 | | Hex | 25 |
| DataRows | 05 | 10 | Char | 50 |
| * | 10 | 1 | Hex | 50 |

**Record YYY**

| Name | Level | Occurs | Type | Bytes |
|---|---|---|---|---|
| HEX-AREA | 05 | 1 | Hex | 95 |
| Field1 | 10 | 1 | Char | 20 |
| Field2 | 10 | 1 | Char | 25 |
| DataRows | 10 | 10 | Char | 50 |

It is recommended that you place the *occurs* items at the end of the record structure.

## Other factors

Programmers must also understand that additional logic must be coded on the server application when the XXX record structure is used. The EBCDIC/ASCII conversion provided with VisualAge Generator will not occur since the lowest data type of each level is HEX. To ensure that the data is converted correctly, the server application needs logic that moves the contents of XXX to another record with similar data structure, except it omits the HEX data items. This new record is then passed as a parameter to the VisualAge Generator routine ELACONV. This call should be made when the server application starts and when the server application stops.

In addition to the definition of the record structure and the initialization of the record, placement of the record in the server's called parameter list is crucial. If multiple parameters are passed, then the last parameter specified should be the one most likely to contain the binary zeros.

Simple test cases developed by ConAgra Trading and Processing showed an 80% reduction in the COMMAREA length when the above design practices were implemented.

# Summary

Successful implementation of a client/server application system is not guaranteed by using the most popular development tools. Instead, application developers must continue to practice good programming techniques. This article has introduced two such techniques, which help reduce the amount of data transmitted across the network. Both techniques can be implemented using VisualAge Generator.

# Displaying 4-digit Year Values in Windows

*by Alex Akilov, VisualAge Generator Development*

This article describes one way to customize VisualAge Generator to display a 4-digit year value when running in the Windows environment.

There are several layers and transformations that a date value goes through from the database to the GUI screen and back.

A field is declared in the database as DATE type. Because VisualAge Generator has no DATE type, you must use either a CHAR data item or a NUM data item to represent the value. These data items must have a standard size (6 bytes, 8 bytes, or 10 bytes). VisualAge Generator attempts to perform a transformation on the value so that the value is interpreted properly. This transformation is controlled by the setting in the environment variable, EZERSQLDATE.

The value must then be taken from the data item and put into a field on the GUI screen. The field has a data type declaration that can be used to control how the value is presented. The value that the field will actually hold onto is a DATE object that always has a 4-digit value. You can tell the text field (in the customized settings) whether to show all year digits or not. You can also control the order to display the day/year/month components of the DATE object by using the settings of the text field.

The format of the DATE in the data item has no implication on how the DATE should be displayed. A transformation is performed based on the specifications of how the text field wants to display the DATE and how the value in the data item should be interpreted. The latter is controlled by either the environment variable EZERGRGL_XXX (where *XXX* is the NLS suffix, for example, ENU) or EZERGRGS_XXX (if the size of the data item was too small to hold the 4-digits).

In conclusion, how you display 4 digits has nothing to do with what is actually returned in the data item and eventually stored in the database. But what is important is that you set the environment variables as recommended in this article and make sure that your VisualAge Generator data item is declared to be large enough to contain a 4-digit year value.

## A VisualAge Generator Fact

There are over 700 VisualAge Generator customers throughout 50 countries and 38 states in the U.S.

## The VisualAge Generator Web Page

Visit our Web site to get the lastest product information and customer briefs:

**http://www.software.ibm.com/software/ad/visgen**

# Try Our One-Stop Shopping Spot

*by Heather Albright and John Berry, VisualAge Generator Consulting Services*

The IBM Software Solutions Division development laboratory has established the VisualAge Generator Consulting Services organization. This organization provides a variety of services to VisualAge Generator and Cross System Product (CSP) current and prospective customers. The group is located in the Research Triangle Park facility in North Carolina and is prepared to offer services world-wide.

The consulting team consists of a core group of knowledgeable IBM professionals with links to a network of other IBM service providers and external business partners. These professionals have an extensive technical knowledge of VisualAge Generator and are knowledgeable in how to apply the solution in multiple customer environments.

To effectively prepare your environment for the VisualAge Generator solution, the consulting team is prepared to perform the following services:

- Set up and establish the local area network (LAN) infrastructure
- Install the VisualAge Generator Developer and prerequisite products
- Install the VisualAge Generator runtime environment
- Demonstrate the use of VisualAge Generator from application development to execution (Proof of Concept (POC))
- Migrate existing Cross System Product (CSP) applications to the VisualAge Generator environment
- Provide specialized education
- Provide mentoring service for developers as they increase their expertise
- Design and develop specific client, server, or stand-alone applications using either a graphical user interface (GUI) or text user interface (TUI)
- Integrate VisualAge Generator with the TeamConnection library management system and Data Atlas for database definition

You can reduce your cost and overall development cycle time by utilizing these services to accelerate your transition to the VisualAge Generator products. Taking advantage of these services gives you access to an extensive technical skill base. You can use this skill base to either help you build and grow your own internal skills or to avoid the need to hire and train new personnel.

For more information regarding VisualAge Generator Consulting Services, you can contact your local marketing representative or contact John S. Berry at (919) 254-6745 or jsberry@carvm3.vnet.ibm.com.

The establishment of the VisualAge Generator Consulting Services group augments other services organizations provided by the Software Solutions Division: for example, the VisualAge Solutions Group, which provides services for VisualAge Smalltalk; and the TeamConnection Consulting Services, which provides services for TeamConnection and DataAtlas products. For information on VisualAge Smalltalk services, contact Greg Bonadies at (919) 254-1116 or vsg@us.ibm.com.

For information regarding TeamConnection and DataAtlas services, contact John Senegal at (919) 245-0045 or jsenegal@vnet.ibm.com.

# Configuring DCE Servers

*by John Snyder and Kristine Heaton, VisualAge Generator Development*

Configuring distributed computing environment (DCE) servers can be complicated, but this article provides some suggestions to simplify the process and maximize system performance. This article covers the following:

- Pros and cons of using VisualAge Generator's middleware and DCE
- Insight into how DCE works
- A process walk-through for deciding how many servers you need and what to put on the servers

## A Comparison of the Middleware and DCE

Both VisualAge Generator middleware (DNA) and VisualAge Generator DCE use a constantly running catcher program to monitor requests. The most important difference is that the VisualAge Generator middleware produces a new system process to handle each request, and VisualAge Generator middleware can handle many requests simultaneously. The VisualAge Generator DCE catcher does not require as much overhead as the middleware catcher to start the called application, but DCE can only handle one request at a time. The DCE catcher maintains a queue of up to 8 requests while processing, but all requests received when the queue is full are rejected.

## DCE Highlights

Well, you are probably wondering, "why use the DCE protocol if the DCE servers single-thread all requests and will only queue 8 requests while one request is running?" By starting multiple DCE servers, DCE randomly assigns one server from all servers available that accept requests for a called application. DCE does not query for available servers each time a call is made from the client. DCE clients cache location information to eliminate the overhead of sending a request to the DCE cell directory service server each time. This means that once a client locates a server that handles a particular called application, the client continues to call that server to handle the request. As long as all the servers are started and their location is known by the clients prior to making a request, the distribution of calls to each server should be distributed evenly.

## Deciding How Many Servers

How many servers you need and the best way to separate the applications serviced by each of the servers depends on a number of conditions. The critical factors affecting configuration are:

- The time required by each called application to process a request
- The number of clients making calls to the DCE servers
- The frequency of calls to each application
- The response time criteria for the clients

## A Sample Process

1. Start one VisualAge Generator DCE server with all called applications listed. Ensure that any applications requiring secure DCE calls are listed in the SECURE section. Set CSOTROPT=2.
2. Start a client and make calls to each of the called applications using actual amounts of data that you normally send to a client. You might want to make multiple calls to each application to ensure that you get a representative sampling of the time it takes to process each of the called applications.
3. Examine the server's CSOTRACE.OUT file and determine how long it takes to process each called application. You can view the CSOTRACE.OUT file on the client machine to see the effect of adding in the network transport times.
4. Divide the called applications into two groups: applications called frequently, and applications called infrequently. Establish a required response time for each of the applications. Keep in mind that unless clients have their own dedicated DCE servers, there might be a short wait for each request.
5. Separate the two groups of applications again with respect to processing time. Place each of the applications in one of four categories:
   - Frequently called applications—short duration
   - Frequently called applications—long duration
   - Seldom-called applications—short duration
   - Seldom-called applications—long duration

6. Address each of the categories as follows:

   - Frequently called applications—short duration. Bundle these applications together onto a couple of servers. The number of servers depends on how many clients there are and the number of concurrent requests outstanding at any one time. The more of these servers available, the better the response time.

     We recommend starting out with 1 server per 5 clients.

   - Frequently called, long-duration applications need to have more available servers to keep the response time down. These applications should not be mixed with the short duration applications because that could cause a short duration application to wait for a long duration application to complete.

     We recommend 1 server per 3 clients.

   - Seldom-called,short-duration applications should be bundled together onto one server. Because these applications are not called often, multiple concurrent requests for them should be rare.

     We recommend 1 server per 50 clients.

   - Seldom-called, long-duration applications should be bundled together onto a few servers. Multiple concurrent calls are more likely to occur in this category than in the seldom- called, short-duration application category because these applications have longer-running processing times.

     We recommend 1 server per 25-30 clients.

Remember, the more servers that you have waiting to receive requests, the better performance should be.

The following are some rules for splitting the applications across multiple DCE servers:

- Each DCE server MUST have its own DCE principal.
- All servers that advertise in the same server ID and location combination *must* be running the *same* set of applications. If they are not, you will send requests to DCE servers for applications that they do not support.
- Duplicate DCE servers can run on either the same machine or different machines.

In the previous example, there are four sets of DCE configuration files: one for each of the four categories. All the configuration files used by servers for a particular category would be identical except for the DCE principal name.

When you plan your servers, give some thought to security. Security for the called applications is done at the DCE server level. If you have one or more applications that require special security, you should put them on a separate server. Then use the considerations listed above to determine how to configure the secure server.

## Acronyms

| | |
|---|---|
| 4GL | fourth-generation language |
| AIX | Advanced Interactive Executive |
| API | Application Programming Interface |
| AS/400 | Application Systems/400 |
| CAE/2 | Client Application Enabler/2 |
| CASE | Computer-aided Software Engineering |
| CICS | Customer Information Control System |
| CICS OS2 | Customer Information Control Operating System/2 |
| CPU | Central Processing Unit |
| CSP | Cross System Product |
| DB2 | Database 2 |
| DDL | data definition language |
| DBMS | database management systems |
| DCE | distributed computing environment |
| EMEA | Europe/Middle East/Africa |
| GUI | graphical user interface |
| IBM | International Business Machine |
| IMS | Information Management System |
| ITSO | International Technical Support Organization |
| 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 |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| VM | Virtual Machine |
| VSE | Virtual Storage Extended |
| WWW | World Wide Web |

# 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 2.2.

☐ I'd like to participate in a Beta program for the next release of VisualAge Generator.

☐ 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 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.*

G242-0315-05

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

G242-0315-05

Cut or Fold Along Line

# Questions & Answers

**Question:** Can I use the COMMDATA parameter format (parameters passed in the CICS COMMAREA) when calling a generated application from a non-VisualAge Generator application in the CICS environment?

**Answer:** Yes, you can. To have the called application receive parameter data in the COMMAREA, specify the following linkage table entry when generating the application:

```
:calllink applname=application-name linktype=cicslink parmform=commdata
```

Use a CICS Link in the calling program to call the generated application. The length of the COMMAREA that is passed must match exactly the total length of the parameters declared in the called parameter list for the generated application.

# 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**

G242-0315-05