



# VisualAge<sup>®</sup> Generator

**A Powerful New Vision of Programming™**

## Special millennium edition

### Contents

The best is yet to come .....	2
Performance, performance, performance! .....	3
Effective MQSeries application deployment using VisualAge Generator .....	5
Transforming TUIs into Web Transactions .....	7
Generating VAGen Java programs: High altitude instructions .....	11
To Java or not to Java? .....	13
What is VAGen Object Scripting all about and why do I care? .....	14
Object Scripting made easy using the VAGen Script Wizard .....	18
DB2 for OS/390 hints and tips .....	21
Some frequently asked questions for VisualAge Generator on OS/400 .....	28
FUNCTIONS - Here at last! .....	30
Functions, local storage, parameters: everything you wanted to know and then some .....	32
VisualAge Smalltalk vs. VisualAge for Java - a simple comparison .....	34
I'll never forget . . . Whatshisname .....	36
SHARE in Long Beach, February 25 - March 2, 2001 .....	38
Planning for migration to VisualAge Generator 4.X .....	39
Client/Server configurations for VisualAge Generator version 4.0 .....	43
Accessing VSAM Files on OS/390 from VisualAge Generator .....	47
Testing modifications to generated UI Record JSPs .....	55

# The best is yet to come

by Steve Choquette, VisualAge Generator Product Manager

It's hard to believe that 2000 is over and the new millennium has officially begun. The memories of preparing for Y2K and hoping that nothing cataclysmic would occur seem so fresh. Through careful planning that started many years ago, Y2K turned out to be a reason to celebrate, instead of a disaster. As the old adage goes, "prior planning prevents poor performance." Congratulations on being a survivor!

Allow me to introduce myself. My name is Steve Choquette and I am the new product manager for VisualAge Generator. Prior to joining this organization in January of 2000, I was a developer, support manager, and release manager for an IBM product called Communications Server for OS/390. CS/390 provides the SNA and TCP/IP support for the S/390, Amdahl, and Hitachi mainframes. As a support manager for a product that controls mainframe access for most of the Fortune 500 companies, I learned that product quality is important to your success and IBM's.

This year has brought about a heightened emphasis on e-business – connecting your existing corporate infrastructure with the web. And what better product to simplify that task than VisualAge Generator? The UI Record programming model, introduced in Version 4.0, allows you to easily create web pages that provide browser access to the important applications and data running on your enterprise servers – without knowing all the underlying details of HTML, JSPs, multitier servers, gateways, communication protocols, state management, and database query languages. How much simpler could it get?

The VisualAge Generator team has had a busy year in 2000. Hopefully you got a chance to meet some of our developers at conferences – COMMON, SHARE, AS/400 Technical Conference, CICS & MQ Technical Conference, Solutions Developer 2000, Colorado Software

Summit, VG Users Group, or the VG International Symposium.

In addition to two Fixpacks for Version 4.0, the VisualAge Generator team has worked on Version 4.5, which shipped in September. That release, described in detail in this newsletter, provides numerous enhancements that will help your business be successful in the new web economy. The MQSeries Integration function, for example, allows you to take advantage of this powerful message queuing tool, using the same I/O verbs you are already familiar with. The Java Server Generation item adds the capability to generate Java to a Windows NT server. Future releases will generate Java for other server platforms. (Remember that Java GUI client support was added in Version 4.0.)

In 2000, we partnered with the IBM S/390 University Relations team, Perficient (an IBM Business Partner), and Northern Illinois University to teach a two-week course on e-business to faculty members from universities around the world. The second week of the class focused on VisualAge Generator. Our goal is to have several universities whose Computer Science graduates are trained in VisualAge Generator.

In the next few months, look to hear more about VisualAge Generator and the Application Framework for e-Business. The Framework integrates a large number of application development tools, continuing IBM's move toward open standards and platforms, and making it easier for business partners or ISVs to enhance IBM's application development toolkit via plug-ins. As I said in the title, the best is yet to come. I am excited about being a part of the VisualAge Generator team. I look forward to meeting many of you, perhaps at the Nordic GUIDE session January 24-26 in Sweden. See our web site at <http://www.software.ibm.com/ad/visgen> for more information. ■



Powerful enterprise  
e-business solutions

# Performance, performance, performance!

by Wing Hong Ho, VisualAge Generator Tester and Jay Cagle, VisualAge Generator Developer

Recently, a number of performance enhancements were made in the Java version of VisualAge Generator Developer and VisualAge Generator Workgroup Services. This article will detail those improvements as well as provide general recommendations for improving performance.

By making enhancements in the Java library, we improved the performance of accessing VAGen parts throughout the integrated development environment (IDE). In addition, we also optimized the code for some of our part editors so that navigating and editing code is now much more efficient, particularly where extensive tree-diagrams are involved.

These performance improvements will be incorporated in the up-coming FixPak2 for VisualAge Generator on Java version 4.0 and a future FixPak for version 4.5. Performance enhancements can be found in the following areas:

- **Opening the Program Editor** The time it takes to open a large program in the Program Editor is dramatically improved. Large programs that call many functions are now opened up to ten times faster than before with their structure diagrams fully expanded.
- **Expanding the Table and Additional Records list** The operation is now done in a fraction of a second.
- **Opening a record or table** Performance improved by at least an order of magnitude, particularly for records and tables that have many shared data items. Most records and tables are opened almost instantly.
- **Testing programs in Interactive Test Facility (ITF)** The time it takes to start the test is greatly improved. Programs load and run faster in ITF, particularly when testing the program for the first time when the parts are not already in memory.
- **Working with open editions** Performance of accessing VAGen parts from an open edition is greatly improved. There is no longer a

performance difference between open editions and versions of a package.

You will also see an improvement in performance when you import large ESF files and when you migrate large development projects. These improvements will help your team work more efficiently and increase their productivity.

On the runtime side, we also have good news to bring you. The server code for C++ platforms has been optimized, and our testing indicates a speed increase that ranges from 20-30% for a typical generated program. In fact, in a test using C++ server with AIX CICS, we saw a tenfold improvement in the number of transactions completed per second.

## Some performance hints and tips:

There are some easy steps you can take to ensure that performance is optimal. Here are some of them:

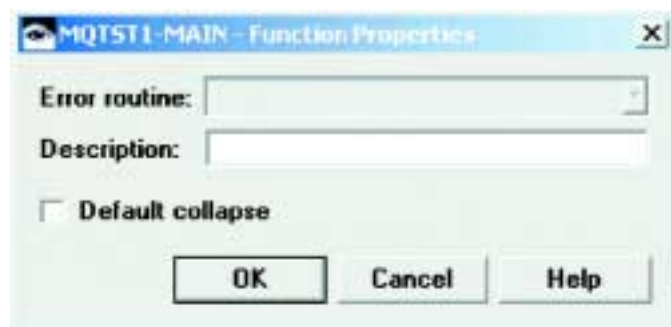
- **Partition your code into smaller logical units** Put each functional area of your code in a separate package/application. Not only does this make your program easier to understand and maintain, it is required to maintain acceptable performance. You should generally have fewer than 500 parts of the same part type in the same package/application. Exceeding this limit can degrade performance.

- **Take advantage of the Program Editor option** The “Preferred Editor View” option controls how much information is initially displayed in the structure diagram of the Program Editor. You can set this option from the Program section of the VisualAge Generator Preferences window. In the Part Browser, choose Windows -> Options in Java, or Options -> Preferences in Smalltalk.

There you will find three choices:

- **Main function:** Choose this option to display only the main function of the program initially.
- **Entire structure diagram:** Choose this option to display the entire structure diagram, fully expanded.
- **Default collapse:** Choosing this option gives you the most control over how much detail is displayed when the program is first loaded in the Program Editor. You can prevent certain function nodes from being expanded by specifying the “Default collapse” property. To specify this property, from the Function Editor, select Define -> Properties and click the “Default collapse” checkbox.

Intuitively, you can guess that selecting the “Main function” option takes the least time to execute, while selecting “Entire structure diagram” takes the most time. Choosing “Default collapse” will be somewhere in between, depending on how you arranged the “Default collapse”



## Performance, performance, performance, continued

settings for your functions. By default, the “Default collapse” check box is blank for all functions, so all function nodes are expanded in the structure diagram, making the situation the same as if you selected “Entire structure diagram”.

The latest improvements in the Program Editor code reduced the performance difference between the initial display options. However, a judicious use of the “Default collapse” option can still make the structure diagram much easier to read.

**In a test using C++ server with AIX CICS, we saw a tenfold improvement in the number of transactions completed per second.**

Consider checking the “Default collapse” button for functions that have very isolated and well-defined uses. This way, you can make the function appear like a subroutine, with its internal structure hidden away to reduce clutter.

- **Hide the monitors when using ITF**  
The “Executing Stack Monitors” and “Statement Monitor” in ITF are updated frequently when running a program in ITF. This can put additional demands on your processor and reduce performance. By hiding these monitors when you don’t need them, you can improve the response time of ITF.

The easiest way to accomplish this is by choosing View -> Hide all monitors in ITF. If your program is already running, you can still select it by first pausing the execution. Alternatively, you can minimize ITF during execution to improve performance.

*What about times when you want to visualize the program flow? By hiding the monitors, haven’t you defeated the purpose of having them in the first place?*

You can place a breakpoint where you want to start tracing through your code. When execution pauses at the breakpoints, you can re-display the monitors and proceed in order to get a feel of the program flow.

- **Regularly defragment your hard drive.** This is more important to do on the ENVY server. The ENVY server is typically I/O bound, so anything that improves I/O performance will enhance your team’s productivity. By defragmenting your hard drive regularly, you can make sure that parts are retrieved efficiently.

- **Consult the Design Guide for more performance tips.** The Design Guide has information that can help you optimize the performance of your generated programs. Some of the topics include:

- Using NUMC and PACK instead of NUM and PACF
- Making a Table ‘Resident’
- Generating options that can affect performance

For details about these and other topics, see Appendix A of the Design Guide. ■

### Interested in learning more about VisualAge Generator? Check out these classes.

#### VisualAge Generator v4.5 on Java - Basics

(Class # SW410)

Jan. 29-Feb 2 Pittsburgh PA

Mar 19-23 Pittsburgh PA

#### VisualAge Generator v4.5 on Java - Transitions from Cross System Product

(Class # SW411)

Mar 6-9 Pittsburgh PA

#### VisualAge Generator on Java - Web Transaction Workshop

(Class # SW417)

Feb. 20-22 Middletown RI

Mar. 20-22 San Diego CA

**To learn more about these classes or enroll, see our web site:**

<http://www.ibm.com/software/ad/aimclasses>

# Effective MQSeries application deployment using VisualAge Generator

by Paul Hoffman, VisualAge Generator Architect and Sanjay Chandru, VisualAge Generator Developer

The IBM MQSeries messaging system provides a simple asynchronous alternative to server program calls as a way of connecting applications across dissimilar environments.

With MQSeries messaging, programs communicate by writing messages (strings of data) to queues and reading messages from queues via MQ API calls to a message queue manager. The queue can reside on the same system as the program (local queue) or on another system (remote queue). The queue manager handles any communications necessary to access remote queues.

- Communication through queues makes asynchronous operations possible. For example, when you use queues:
- Communicating programs can run in parallel.
- Communicating programs can run at different times.
- Intermittent communications link failures do not stop communication since the queue manager can store and forward messages until the link comes back up.

## MQSeries and VisualAge Generator Version 4.0

VisualAge Generator is an effective tool for hiding the complexities of implementing MQSeries functionality on multi-platform applications. VisualAge Generator supports most of the platforms on which MQSeries applications can be deployed currently. You can write and test MQSeries programs for these platforms in the VAGen development environment before you run them on the target platform. VisualAge Generator Version 4.0 contains new sample programs and reusable parts for building programs for MQSeries Version 5.0 APIs.

## New reusable parts

Reusable MQ 4GL parts include:

- Functions for calling each MQSeries API
- Records for each MQSeries data structure
- Functions for initializing each data structure
- Tables for MQSeries constants, return codes, and return code descriptions
- Function for checking MQSeries return codes and building a standard error description

The names of the reusable parts shipped with version 4.0 are different from those shipped with previous versions of VisualAge Generator. The functions for API calls are now defined with parameters, allowing the same functions to be used for accessing different queues within the same program.

## New sample programs

Sample programs implementing the basic MQSeries calls are also available with VisualAge Generator 4.0. The sample programs are:

- MQGUI - graphic user interface
- MQWEB - web browser user interface
- MQ3270 - 3270 user interface
- MQBATCH - batch print program

All the sample programs access a message queue via the reusable MQ parts. The MQWEB program is an example of a browser program for which no state is saved on the server.

The sample programs and reusable parts can be loaded from MQS.DAT in the Smalltalk sample directory or MQJ.DAT in the Java samples directory.

## Prerequisites for running programs

- Install MQSeries server and client on the platform on which you wish to test the sample applications (MQSeries Version 5.x for AIX, AS/400, HP-UX, OS/2, Sun Solaris, Windows NT, Version 1.x for OS/390, Version 2.x for VSE).

**...the developer does not have to worry about modifying or writing new code for every platform.**

- Verify queue manager using MQSeries sample programs provided with MQSeries software
- Define local queue. The queue name must be TESTQUEUE if running the MQBATCH program
- VAGenerator 4.0 needs to be installed and configured.
- Load the sample program .dat file
- Start the queue manager. The target queue must have its Get and Put message options enabled.
- Run the sample programs in test facility or generate and run in any supported runtime environment.

## Example

This example shows function calls written in VAGen 4.0 4GL for an MQSeries specific application. The code is derived from the MQWEB\_PUT function from the MQWEB sample program.

In the code sample, connection to a queue manager is established, a queue is accessed, messages are written to the queue, and the connections are closed. The MQSeries API calls are made by reusable VAGen parts that have the same names as the MQ APIs:

- MQCONN - connect to queue manager
- MQOPEN - open queue
- MQPUT - put message to queue
- MQCLOSE - close queue
- MQDISC - disconnect from queue manager

## Effective MQSeries application deployment using VisualAge Generator, continued

The records used in the code represent MQ data structures:

- MQOD - object (queue) descriptor
- MQMD - message descriptor
- MQPMO - put options

or program specific information:

- MQUIR - web user interface record
- MQSAMPLE\_STATE - program state

(alternate specification of reusable MQSTATE record)

Whenever a call is made, the completion code and reason code are checked for the success or failure of each call. Subsequent steps are based on the result of previous calls. The advantage of developing MQSeries specific applications using VAGEN is that, code written in 4GL is then

generated to any target platform. This ensures that the developer does not have to worry about modifying or writing new code for every platform on which this application is deployed.

### Information and reference

For general information on MQSeries and VisualAge Generator:

<http://www-4.ibm.com/software/ts/mqseries/>

<http://www-4.ibm.com/software/ad/visgen/>

<http://www.redbooks.ibm.com/>;  
MQSeries Primer, REDP0021

For specific information on implementing MQSeries programs in VisualAge Generator, refer to Chapter 21, Implementing Client/Server Processing Using the Message Queue Interface in the VisualAge Generator Version 4.0 Client/Server Communications Guide, SH23-0261-00.

Interested in MQSeries support in version 4.5? See our white paper at <http://www.ibm.com/software/ad/visgen/library> ■

```
/* check for input message
IF MQUIR.PUT_MESSAGE EQ ` `;
  MOVE "Enter message text in put message text field"
  TO MQUIR.ERROR_MESSAGE;
  EZERTN();
END;

/* Connect to message queue manager
MQCONN(MQSAMPLE_STATE, MQUIR.MQMANAGER);
MQWEB_MQCHECK();

/* Put messages to queue;
IF MQSAMPLE_STATE.COMPCODE LE MQCC_WARNING;

/* Open message queue
MQOD_INIT(MQOD);
MOVE MQUIR.MQQUEUE TO MQOD.OBJECTNAME;
MOVE MQUIR.MQMANAGER TO MQOD.OBJECTQMGRNAME;
MQSAMPLE_STATE.OPTIONS = MQOO_OUTPUT;
MQOPEN(MQSAMPLE_STATE, MQOD);
MQWEB_MQCHECK();

/* Put messages to queue
IF MQSAMPLE_STATE.COMPCODE LE MQCC_WARNING;
  MQPMO_INIT(MQPMO);
  MQMD_INIT(MQMD);
  MQSAMPLE_STATE.BUFFERLENGTH = EZEBYTES(MQUIR.PUT_MESSAGE);

  MQPUT(MQSAMPLE_STATE, MQMD, MQPMO, MQUIR.PUT_MESSAGE);
  MQWEB_MQCHECK();
  IF MQSAMPLE_STATE.COMPCODE EQ 0;
    MOVE "Message written to queue" TO MQUIR.ERROR_MESSAGE;
  END;

/* Close queue
MQSAMPLE_STATE.OPTIONS = MQCO_NONE;
MQCLOSE(MQSAMPLE_STATE);
MQWEB_MQCHECK();
END;
END;

/* Disconnect from message queue manager
MQDISC(MQSAMPLE_STATE);
MQWEB_MQCHECK();
END;
```

**Looking for the latest industry R&D news? Checkout the IBM Journal of Research and Development and the iBM Systems Journal at:**

<http://www.research.ibm.com/journal/>

# Transforming TUIs into web transactions

by Theresa Smit, Manager, VisualAge Generator Consulting Services

Are you taking advantage of the opportunities that e-business provides? Let VisualAge Generator Version 4 help you move your Cross System Product or VisualAge Generator applications into the world of e-business. IBM is leveraging its experience in delivering scalable and secure systems to equip this new world with optimized infrastructure and tools. The result is a set of products and architectures that supports the creation and deployment of end-to-end, multi-tiered e-business systems. If an e-business system is viewed as a logical 3-tier solution, IBM WebSphere Application Server provides the run-time environment for tier-2, WebSphere Studio provides the tools for Web-site management and page

have an analysis phase to identify the functionality to include in the Web application. The main purpose of this phase is to determine the functions in the existing code that are to be retained and to define what is required for screen navigation. This step can save converting program functions that are not needed in the new implementation and serve to clarify end user interface components.

For example, a TUI screen that displays a selection list of 20 codes or records and has forward, backward, right, and left scrolling can be presented as a web page that just does forward scrolling and handles the selection of a record. The code that handles the back, left, and right functions would be

## Not prepared to make the coding changes to get to the Web, but need to be there today?

composition, and the VisualAge products allow professional programmers to rapidly create transactional data servers and business logic. In this context, VisualAge Generator plays a central role in the development of new third-tier transactional servers, which must guarantee high performance and transaction volumes and reach a variety of legacy platforms and data, automatically generating the code necessary to use their services in a servlet/JSP component dynamic HTML context. This article describes the issues you should consider and processes you can use to convert your existing TUI applications to Web Transactions and maximize the reuse of existing business logic, enabling you to retain your current investment.

Not prepared to make the coding changes to get to the Web, yet need to be there today? Then, you might consider using a screen-scraping implementation such as CICS Host On-Demand, CICS Web Interface with 3270 Bridge or NetCICS. This tool makes your CICS TUI transactions immediately available on the Web! Although this implementation does not exploit Web UI functionality, it can quickly get your TUI applications to the Web. You might consider this your first step into Web development. See <http://www-4.ibm.com/software/ts/cics/library/whitepapers/cicsweb/webtable.html>

Your next step into Web development should maximize Web UI functionality with features like radio buttons, drop-down lists, forms, and program links. To achieve this, you should consider using the VAGen UI Record to transform your TUI application into an e-business Web Transaction.

### Phase 1 - analysis

You should approach this transformation effort as you would approach any other new Web development. First, you should

eliminated. You can also eliminate code for fields such as a small (less than 100 items) single field selection list, which can be better presented as a drop-down list. Other considerations include the number of items shown in the list and how a user makes a selection from the list (radio button vs. program link).

### Phase 2 - basic transformation

After you have identified the functions and code that need to be changed, there are a few steps to transform the basic functions into a Web user interface.

1. Import your code into VisualAge Generator V4 using the migration tool or the import function (refer to *VisualAge Generator Migration Guide, SH23-0267* for more information). There are additional considerations if you are moving from Cross System Product V3.3 or earlier to VisualAge Generator for the first time because migrating from an interpretive execution to generated COBOL might require additional changes to your source code (refer to *Migrating Cross System Product Applications to Visual Age Generator, SH23-0244-01* for more information on this type of migration).
2. Test the TUI base code to ensure that all components are loaded and to establish that the development environment is functioning properly. Establish your naming conventions and library organization (Projects, Packages, etc.) for your transformed parts and for the new part types, UI Record and Web Transaction. (Refer to *VisualAge Generator Guide to Migrating MSL's to ENVY, SH23-0252-01* and the *VisualAge Generator V4 Migration Guide, SH23-0267* for more information.)
3. For each VAGen Map in the VAGen Parts Browser, select the map, then select **Create UI Record from Map from the**

## Transforming TUIs into Web Transactions, continued

**context menu.** This action creates a new VAGen UI Record with fields for each named map field (variable field). Note that any label text or constant fields are not included in the UI Record. The default label text is the field name. Be sure to follow your naming standards for the new UI Record name. Do not exceed the length of the name of the old field. For example, if the original map was named XY01M02, then a good name for the new UI record would be XY01U02. This naming convention will enable you to more easily change the external source format.

4. Export the program to external source format using **VAGen Export with Associates**. Using a text editor, globally change the map name to the UI Record name where it is referenced in the code. Change all occurrences of EZEMSG to EZMSG, and, optionally, change other component names such as program name and function names. Consistent naming standards will simplify this step and reduce duplicate parts if you have both the old TUI program and the new Web Transaction in your workspace at the same time. After you have made the changes, import your new program from the changed external source format file.
5. Make program changes to implement the UI Record using the Converse model (see *VisualAge Generator V4 System Development Guide, SG24-5467-00* for more information on defining a UI Record)
  - Change the program type to **Web Transaction**.
  - Ensure that any CONVERSE functions are using the new UI Record. If you have several maps that are DISPLAYed prior to the final CONVERSE then, convert them to UI Records, and add all of the data items to one record. Pop-up maps will have to be considered separately. You might use a drop-down list or another UI part to replace their function.
  - Add labels to each Input/Output field. UI Record fields can be customized to add table, function and other edits, and help text. Table edits will appear as drop-down lists. For function edits that are not doing I/O or server calls, you can choose to have them run on the web server or on the host server. Help text for each field is included in the generated Java bean and can be referenced in the JSP. This is not referenced in the default JSP generated by VisualAge Generator.

- Add submit buttons to your UI Record for the Enter key and each function key processed. You can set the initial value for each button to the EZEAIID equivalent value (ENTER, PF1, PF3...); however if your program executes a **SET uirecord EMPTY** this will clear the initial values. If submit buttons have no value they are not displayed. So set the submit values prior to the CONVERSE as shown in the following example:

```
MOVE 'PF3' TO EXIT-BUTTON;  
MOVE 'ENTER' TO SUBMIT-BUTTON;  
  
***----- CONVERSE UGDETB-MAP1 -----***  
  
IF EZEAIID NOT PF3;  
  
..
```

- Statements that SET or TEST the map item's attributes are not recognized and will cause errors. For example, SET map item PROTECTED will cause an error. These statements need to be commented out or removed. The way in which fields are displayed or hidden is determined by the UI designer. To make this determination, additional data will need to be passed in the UI Record. For example, if your map has certain fields that are updatable depending on user security, then that user security information needs to be included in the UI record. The UI designer can then change the JSP to access the security data passed and set the display or hide fields appropriately. Testing a field for MODIFIED is allowed, but setting a field to MODIFIED is not. This might also have an impact on your security or edit processing.
  - Add a title to your UI Record by editing the properties for the UI Record.
  - Change any date fields that are defined as numeric and have a date mask to a character field (length 10). Then move EZEDTELC to this field to format the date.
  - For map array fields that have been converted to items with occurrences, you need to add a counter item to the UI Record to hold the number of elements to display in the list when the UI Record is converted. Then, in the custom settings for the item with occurrences, specify the counter item as the occurrences item
7. Test your program. When a CONVERSE is reached, the Browser is displayed with your UI Record contents. (see Figure 1) You may need additional code and UI Record changes to reach a baseline functionality.

**Need to know more about VAGen Web Transactions? Check out VisualAge Generator classes and workshops at:**

<http://www.ibm.com/software/ad/visgen/education>





Figure 1. Testing Web Transaction

Other things that may need special consideration in this new environment are:

- **Security** Use of EZEUSR or application-specific security for read-only access versus update authority.
- **Attributes** Setting of extended or highlight attributes - how to identify in the UI Record. Could use a separate attribute field for each field or a general one if possible.
- **Messages** If you are using a Message Table, you will need to change references to EZEMNO and EZEMSG to EZEUIERR. In this new environment, when you have multiple fields in error, all field level error messages are displayed below the field rather than one at a time in EZEMSG. General or informational messages such as “Key selection and press enter” moved to EZEMSG will need to be moved to a UI record field like EZMSG instead. Be sure the text of the message still applies to the Web interface.
- **Error checking** If you currently have edit functions specified in your map, these will be included in the UI Record you create from the map. Additionally, you can specify if the edits are to be performed on the Web Server or on the VAGen Server. However, if you SET map fields as MODIFIED to force the edit routine, you will need to change the program to always execute the edit routine after the CONVERSE.
- **Navigation** Consider the use of menus, “fastpath” fields, or other navigation. This function may be better implemented by the Web designer.

### Phase 3 - make it more Web-like

At this point your application will still look very Text-like even though it is displayed using a Browser. To better utilize the benefits of the Web, the UI Record and Web Transaction can be enhanced to make the navigation and internal functionality more Web-like. Some additions might include:

- **Using Program Links as a record selection** - In the UI Record, change a table column to be a UI Type of Program Link, then customize it to specify the program and the first UI Record that will be passed. Be sure that only key information is passed in this manner because there is a limit to the amount of data that can be passed (around 400 bytes). The Linked program must be prepared to handle the First UI record information and use the key information to read the detail record. (see Figures 2a,b & c)



Figure 2.a. UI Record - Adding program link



Figure 2.b Program link properties

## Transforming TUIs into Web Transactions, continued



Figure 2.c Test of UI Record - List with program link

- Use forms to combine multiple Text screens into one UI Record. Because you are no longer limited to a 24x80 screen, you might improve user productivity by retrieving more information on each request. Work with your Web designer to identify what would be best for your users.
- Consider using the XFER stateless model for selection lists and inquiry-only programs and stay with the CONVERSE model for maintenance (Insert, Update, Delete). To implement the XFER model, divide your single application into two programs. Everything prior to the CONVERSE is in the first program and everything after the CONVERSE is in the second program. Each program would use the same UI Record. The UI Record would have a parent field with UI Type of Form that links to the second program. Instead of the first program doing a CONVERSE, it would XFER ' ;UIRECORD. After the user selects a submit button, the second program is invoked and will evaluate the request and continue. (see before-Figure 3 and after-Figure 4)
- Some code is so intertwined with the map attributes and other settings or so heavily modified that it may be faster to use VAGen Templates to recreate the functionality of the program. This is especially true for simple (not much business logic) programs. For example, search for a record based on selection fields, then display a selection list that has a program link column that displays or maintains the detail record. See <http://www-4.ibm.com/software/ad/visgen/about/v4temp.pdf>.

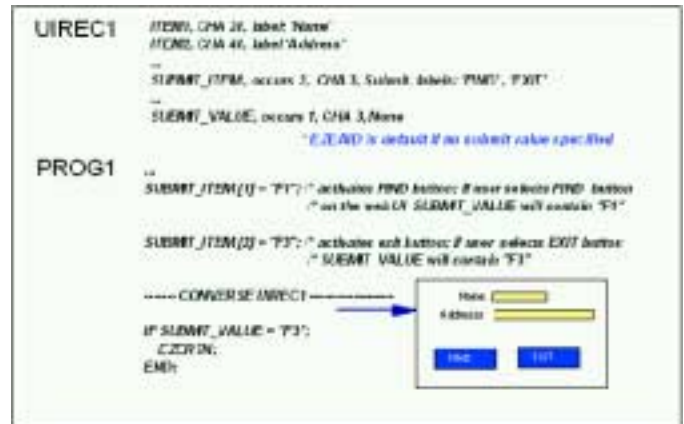


Figure 3. Before (using CONVERSE Single program)



Figure 4. After (using XFER model with two programs)

### Phase 4 - modify default JSP

Up to this point you have only seen the default Java Server Page (JSP) displayed in the Browser. To make the user presentation more weblike, this JSP will need to be enhanced with graphics and incorporated in existing Web pages using Web authoring tools such as Websphere Studio. The UI designer can use the default JSP generated as a base or guide to create the customized JSP.

### Summary

This transformation can retain a large portion of your application business logic. However this effort is not a trivial task. VAGen's new Web technology can be implemented by the same developers who maintain your existing Cross System Product or VAGen applications without having to learn much about Java. However, you will need to develop some expertise in HTML and JSPs to successfully move your applications to the Web. ■

# Generating VAGen Java programs: High altitude instructions

by Matt Heitz, VisualAge Generator Developer

For years, VisualAge Generator has enabled you to quickly develop programs and generate them into either COBOL or C++. In version 4.5 we have added the ability to generate programs in Java as well. This article explains the basics of generating, deploying, and running Java programs.

## Developing for Java

There are no special steps to take while developing a Java program in VAGen and you don't need to be a Java programmer. Whether you're redeploying an existing program or creating a new one, the process is the same as for any VAGen program.

**Most of the generation options that apply to Java programs are similar to the ones for C++ programs, but there are a few exceptions. . .**

However, you should keep in mind that there are currently several limitations on what Java programs can do. They must either be batch programs or web transactions, meaning that maps (text user interfaces) are not supported. Databases are accessed using JDBC, which supports SQL but not DL/I. Java programs don't take advantage of CICS, and at present Windows NT and Windows 2000 are the only platforms they run on. Future versions of VisualAge Generator will address many of these issues.

## Generation options

When you're ready to generate your program into Java, you need to set the generation options. Most of the generation options that apply to Java programs are similar to the ones for C++ programs, but there are a few exceptions. The target system (/SYSTEM option) should be JAVAWINNT. (If you specify the system value as WINNT, you will generate a C++ program.) The only

valid value for the /DBMS option is JDBC, and as in version 4.0 the /PACKAGENAME option specifies the package of the generated Java code.

There is also a new generation option, /GENPROPERTIES, which causes a properties file to be generated along with the program. If you specify /NOGENPROPERTIES, a properties file will not be generated. /NOGENPROPERTIES is the default.

VisualAge Generator Java programs use these properties files instead of environment variables. You can have one properties file that contains global settings and application-specific properties files that override or extend those defaults. The generated properties files contain information taken from some of the generation options, along with the linkage table and resource associations part that were specified with the /LINKAGE and /RESOURCE options.

## Outputs of generation

Java programs are generated into one or more .java files (e.g. PGM1.java) as well as a .tab file made from each table and possibly a .properties file. In addition, some command files (with .bat, .cmd, and .ftp extensions) are created. They are used to deploy the program.

## Deployment

Deployment of generated Java programs is similar to the way generated C++ programs are deployed. There are three different scenarios (see the *VisualAge Generator Generation Guide* for more detailed information).

- You used the /PREP generation option and the target machine is the one you're generating from. The Java code will be placed in the proper directory and compiled for you.
- You used the /PREP generation option and the target machine is different from the one generating the code. In

addition to the code, VisualAge Generator creates a command file that causes the code to be compiled. The code and command file will be copied to the target machine, but you must manually run the command file to compile the code.

- You used the /NOPREP generation option. The code and command files will be placed in the directory identified by the /GENOUT generation option. Running one of those command files will activate steps identical to what the /PREP option does, so you'll be in the one of the first two scenarios.

VisualAge Generator Server must be installed on the machine where your program is compiled.

## Customizing properties

If you used the /GENPROPERTIES generation option, you may want to modify the properties file that was generated with your program. It will have the same name as the program, with a .properties extension. For example, PGM.properties will be generated from a program named PGM.

Properties files are ordinary text files, so you can use your favorite editor to change them. Properties consist of name-value pairs of the form **propertyName = propertyValue**. Each property must be on a separate line in the file.

## Using jar files

Java programs created by tools such as VisualAge for Java are often deployed in Jar files. A Jar file is a collection of compiled Java code that has been compressed to save disk space and minimize download time. VisualAge Generator won't put your Java program in a Jar file, but after it's compiled you can do it yourself if you wish.

## Generating VAGen Java Programs: High Altitude Instructions, continued

The easiest way to put your program in a Jar file is to use the jar program that comes with the JDK. (JDK stands for Java Development Kit, which is the set of programs that includes javac, the Java compiler.) The basic syntax of the jar command is **jar optionLetter f jarName fileList**. *optionLetter* is c if you're creating a new Jar, x if you're extracting a file from the Jar, or u if you're updating the Jar by adding or replacing a file. *JarName* is the name of the Jar, and *fileList* is the names of the files you want to add, extract, or update. Use spaces to separate file names in *fileList*.

Here is an example. If your program was generated into a package called my.java.pkg, and you want to create a Jar named program.jar, change to the directory where the program was deployed and use the command **jar cf program.jar my\java\pkg\*.class \*.tab**. (Notice that the periods in the package name are replaced by backslashes. The jar command needs to know which directory the files are in, not which package they're in.) The .class files are your compiled program, and the .tab files are used by the tables in your program.

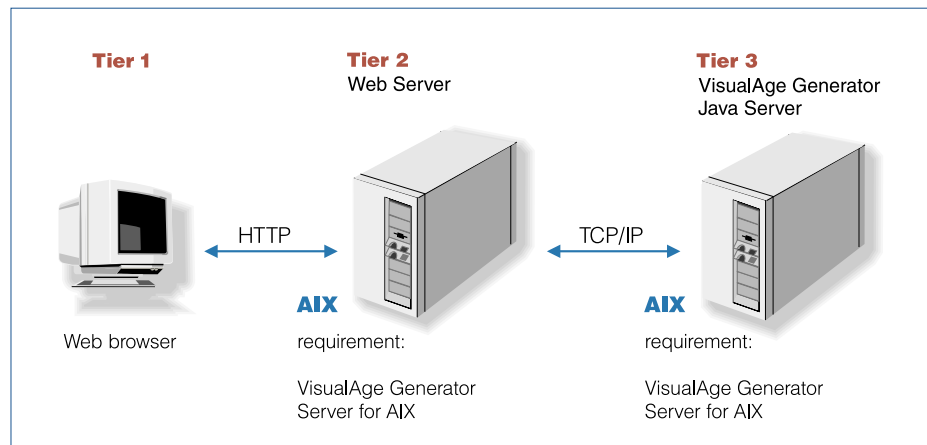
You can also put your properties file in the Jar by adding **\*.properties** to the *fileList*. Putting your properties file in a Jar makes it easier to deploy the program, because everything is in one file, but you'll have to do a little extra work if you decide to change the properties later. Assuming the program is named PGM, use **jar xf program.jar PGM.properties** to extract PGM.properties from the Jar. After editing the file, use **jar uf program.jar PGM.properties** to put it back.

### Running Java programs

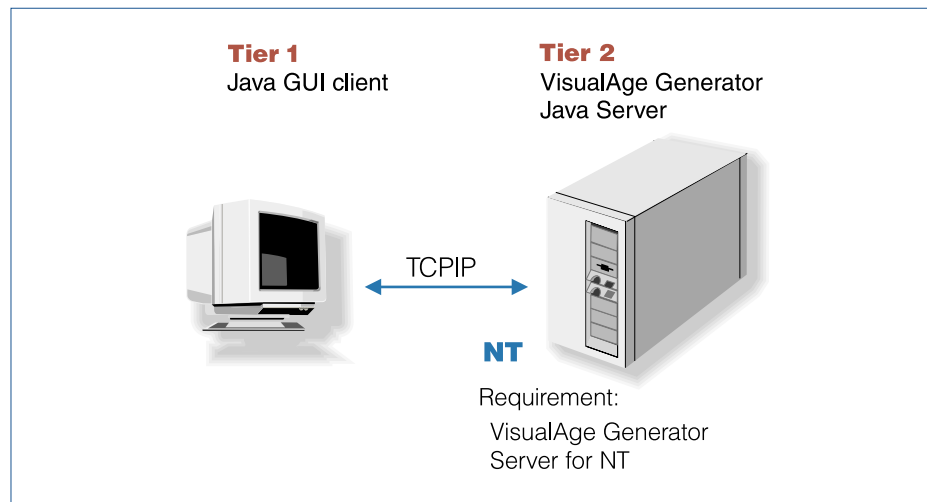
VisualAge Generator Server must be installed on the machine where you want to run your Java program. The CLASSPATH environment variable should include the directory or Jar file where the program is.

Main batch programs can be invoked using a command like **java xyz.123.PGM**. In this case, the program's name is PGM and the /PACKAGENAME specified at generation was xyz.123.

C++ and COBOL web transactions are run according to settings defined in a properties file on the web server. The same is true for Java web transactions, although the settings are slightly different. Java web transaction programs can be contacted over TCP/IP, like a C++ web transaction, or they can run directly inside the web server's Java Virtual Machine. This new "direct" linkage, available only for Java web transactions, may improve performance by eliminating the overhead caused by communicating with a remote machine. ■



VisualAge Generator Web Transaction



Java GUI Client/Server System

# To Java or Not to Java?

by Y.C. Lui, VisualAge Generator Developer

VAGen V4.5 provides Java, in addition to Cobol and C++, as a runtime environment within VisualAge Generator Server. With Java runtime support, users will be able to generate Main Batch, Called Batch, Web Transaction programs and User Interface parts (GUI clients) into Java source code that can then be deployed and executed in the target Java environment.

Now that you can choose either Java or nonJava output (Cobol or C++) as the target language for server program generation, this article illustrates some of the benefits and some of the limitations of choosing Java as the target language.

## Benefits for Java server Generation

A generated VAGen Java program consists of pure Java code, which is portable and can be executed on any platform with Java Virtual Machine (JVM) support. The JVM is free and can be downloaded from IBM, SUN, or other platform vendors. Being a Java program, a VAGen part can now exploit and better interface with other Java programs or features in a JVM environment.

Because Java is an interpretive and dynamic language, it is less susceptible to changes in the supporting systems. A VAGen Java server program, in that respect, is less likely to be impacted if an update is made in VisualAge Generator Server or the JVM, so users don't have to regenerate or redistribute their code.

VAGen Java programs use Java Database Connectivity (JDBC) for SQL database access. JDBC defines a standard set of Java APIs for tool and database developers and makes it possible for database applications that use the API to transparently access any databases, provided that the JDBC driver for the database is available for the environment. Since the base JDBC API is packaged with the JVM, users do

not have to be concerned with additional JDBC installation. All they have to do is specify the location of the JDBC drivers provided by the database vendor to be used in the database applications.

With the support of Java generation, the Java server program can now be generated as an Enterprise Java Bean (EJB), which can then be invoked by a VAGen session bean and run in the same Enterprise Java Server (EJS) process as the session bean. The Java server program participates in true transaction processing, coordinated by the EJS transaction services.

## Some limitations to consider

V4.5 of VisualAge Generator is an initial step into Java runtime support. However, there are certain important VAGen functions that are not currently supported but could be crucial to your enterprise processing. If that is the case, you might have to consider using C++ or Cobol as the target language as an alternative for Java generation. These missing functions are described below.

There is no Map support (i.e. 3270 text based user interface) for Java server programs. Users have to use a Java GUI program to call into a Java Server program or define a Web Transaction program using a User Interface Record.

Currently, a VAGen Java server program can access data sources via JDBC interface. Access to DL/1 databases, VSAM, or Btrieve file systems is not available.

CICS, as a target runtime system, is not currently supported by VAGen Java server programs. The generated VAGen Java server program is simply a Java application. It is not a Java CICS application and cannot access CICS resources or participate in CICS transaction processing.

## Future directions

Future enhancements to Java program generation will probably be focused on adding new function and support for additional target platforms. With the inception of Java server generation, VAGen has placed itself in an excellent position to help our customers address new challenges evolving from Java or Internet environments. ■

*Interested in what else you can do with VisualAge products? Check out VisualAge Developer Domain at:*

<http://www7.software.ibm.com/vad.nsf>

# What is VAGen Object Scripting all about and why do I care?

by Beth Lindsey, VisualAge Generator Developer and Roger Newton, VisualAge Generator Developer

Ever wanted to control GUI parts from within your VisualAge Generator (VAGen) business logic? Tired of GUIs with so many visual connections you can't figure out what's what on the composition editor? Confused about how to use the VAGen PerformRequest function and irritated that its execution is delayed? Want to communicate with other objects like EJBs, etc? Well, if you answered 'Yes' to any of these questions, Object Scripting support is just what you're looking for!

In VAGen V4.0, we've added a new language construct that allows you to call out of a VAGen function to synchronously execute an OO script. You can use these OO scripts to control GUI parts and handle things you are doing today using visual connections or PerformRequests. But I don't know how to write Smalltalk or Java scripts, you might say. Well then, check out the new VAGen Script Wizard, which helps guide you through the steps needed to build an OO script. You can use the VAGen Script Wizard to build simple scripts and also learn enough Smalltalk or Java syntax so you can build more complex scripts on your own. To learn more about the VAGen Script Wizard, see the Object Scripting Made Easy: Using the VAGen Script Wizard article in this newsletter.

This article describes the function and use of Object Scripting support. It also provides a real example that was used in our system testing. The example shows how to rewrite an existing GUI with many visual connections using the new capability of invoking OO scripts from within VAGen functions. Many of the OO scripts in the new GUI were written using the VAGen Script Wizard. The resulting GUI is much easier to understand and maintain. Things you are sure to be happy with!

## Object Scripting overview

A new built-in function, EZESCRPT, allows you to invoke an OO script in the middle of 4GL statement processing. In other words, EZESCRPT allows you to make a "call" to a Smalltalk or Java script from the 4GL logic. This capability enables you to control non-VAGen objects, such as GUI subparts, EJBs, etc., in the normal flow of your business logic.

EZESCRPT is only valid in functions that are invoked from a Smalltalk or Java client. It is not supported in server programs. The scripts that can be invoked must be defined as instance methods of the class you are currently executing. For example, suppose you have a Smalltalk or Java client part named MYGUI, which has a visual connection to execute a VAGen function named FUNCTION1. FUNCTION1 contains the following statement:

```
`EZESCRPT("script1");`
```

Since FUNCTION1 is invoked by a client part, the EZESCRPT

statement is valid. The script, "script1", must be an instance method in the MYGUI class.

The script cannot expect arguments and return values are ignored by VAGen. Within the scripts, you can communicate with any object that is on the visual layout or free-form surface of Smalltalk or Java client parts, other Smalltalk or Java classes, instance and class variables, Enterprise Java Beans (EJBs), etc. EZESCRPT Statement The EZESCRPT function accepts one argument, which is a character, mixed, or DBCS literal or data item containing the name of the script to execute. The method name must match the case, spelling,



## Alternative uses of EZESCRPT statement

and spacing of the script or it will not be found. Double quotes should be used for VAGen literals to preserve case. Mixed and DBCS script names are supported in Java but not in Smalltalk.

## Synchronous execution

The invocation of the OO script named in the EZESCRPT statement is executed synchronously like a CALL statement or VAGen function invocation. The script and any triggered side effects, including possible execution of other VAGen functions or programs, execute before the next 4GL statement.

For example, we have a function, FUNCTION1, that contains an EZESCRPT function that invokes a script. The script sends the 'enable' message to a push button on a view. The push button has a visual connection from the 'enabled' event to the execution of another VAGen function, FUNCTION2. When the EZESCRPT function in FUNCTION1 is executed, the triggered 'enabled' event will cause the FUNCTION2 function to be executed. Once FUNCTION2 has completed and any other statements in the script are executed, control is returned to the 4GL statement following the EZESCRPT statement in FUNCTION1.

Note that events that are deferred by VisualAge, like the 'clicked' event for push buttons, will have any triggered VAGen side effects deferred as well. For example, if the script sends a 'click' message to a push button and the 'clicked' event is connected to a VAGen function, the function will not execute synchronously. You should not rely on the timing of such deferred events.

## Data modifications

For upward compatibility and efficiency, we still collect client data modifications and do not fire them immediately. They are fired before each EZESCRPT statement and at the end of the 4GL function. If no EZESCRPT statements are executed in the function, then the data modifications are handled as they were in previous releases of VAGen. This ensures that existing client code will function predictably.

If EZESCRPT functions are used, the user expects any data modifications from previous MOVE statements to already have been communicated to the client. So, all collected data modifications are fired prior to each EZESCRPT function and any events triggered by the data modifications will execute prior to the actual EZESCRPT message send.

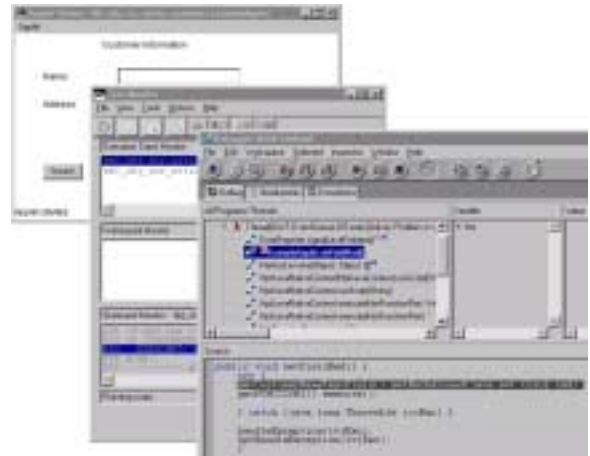
## Communication Between the 4GL and OO scripts

Although arguments cannot be passed to the scripts invoked via EZESCRPT, information can be shared between VAGen and the OO scripts through VAGen data items. Data items that are in records or tables on the free-form surface are visible inside Smalltalk or Java scripts because the records and tables are subparts just like push buttons or text fields. Therefore, if you want to set up information for the script to use, you can move data into VAGen data items prior to the EZESCRPT statement. Then, in the OO script, the data can be accessed to control script logic. Likewise, if the script wants to communicate information back to the 4GL, the script can move data into a VAGen data item that the 4GL can check upon return from the script.

This communication is especially useful in two areas: writing reusable scripts and error processing. Instead of writing a separate script for each text field the user wants to turn red, they could write one reusable script that gets the subpart name from a VAGen data item. The VAGen code would move the appropriate subpart name into the data item prior to invoking the script. For error processing, you could identify a specific VAGen data item that is checked after invocation of a script to see if an error occurred. The script could move a non-zero value into the data item if an error occurred.

## Error handling

If an error occurs in the script, or the script cannot be found, VisualAge Generator does NOT attempt to mask the problem. A Smalltalk or Java debugger will appear so you can see the problem in the context of the OO language you are using and can deal with it like you would other OO problems. You can set breakpoints in the OO script which will also result in a debugger window coming up. If you see an obvious problem in the OO script, you can fix it in the debugger and resume execution of the script. For example, in the following figure, the



**Debugger stopped on simple error in OO script**

debugger came up because the script has the name of the text field spelled incorrectly. You could correct the spelling mistake, save the script method, and resume execution of the script.

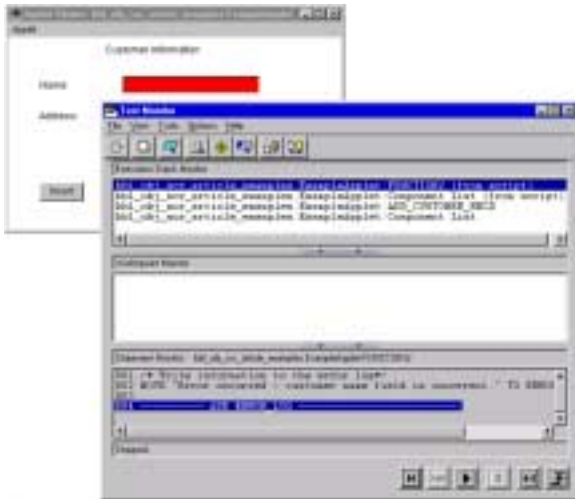
## Interactive Test Facility behavior

There are several important differences you will notice in the behavior of the VAGen Test Monitor when you use it to test client code that uses EZESCRPT:

- When EZESCRPT statements trigger other VAGen logic events, either directly from the script or from other GUI side effects, the same Test Monitor window is used and the subsequent VAGen functions or programs are shown in the Execution Stack Monitor. When you use the Execution Stack Monitor to view the history of VAGen logic execution, you can see the original function that contained the EZESCRPT statement that is in the middle of processing. This behavior is very similar to the way function or CALL statements are handled. The identifier '(from script)' is displayed at the end of the Execution Stack monitor entry to show which logic events were triggered indirectly by an EZESCRPT statement. See the figure below for an example of multiple events being executed from the GUI.
- Shutting down the Test Monitor while you are testing a VAGen logic part triggered via an EZESCRPT statement does not affect the GUI client being tested. Execution of the current logic part is cancelled, but control is returned to the GUI client to continue script processing. If the GUI client invokes another VAGen logic part, then a new Test Monitor will open and run the requested logic. You must shutdown the GUI client to completely quit the test.
- If, while executing an EZESCRPT statement or executing logic parts triggered by an EZESCRPT statement, you reposition the test or save a VAGen part that causes the test facility to reposition the test, the OO script is cancelled and no other script statements are executed. This means that no other VAGen logic parts are invoked. If the test was

## What is VAGen Object Scripting all about and why do I care? continued

repositioned prior to the EZESCRPT statement, you can continue testing to execute the statement again and restart the script.



**Figure 3:** ITF executing LOGPRG, which was executed in the 'setFieldRed' script invoked from ADD\_CUSTOMER\_RECDD via EZESCRPT

### Example: Object scripting comes to the rescue of a complicated GUI

The following example illustrates how simple it is to use the new Object Scripting capabilities in VisualAge Generator Version 4 to control objects on your GUI from your 4GL logic. In this example, an existing GUI application, G4TCS01, has been modified to call various scripts using the EZESCRPT function to control GUI objects/parts. The scripts that are referred to in this example were all developed using the VAGen Script Wizard.

In this example, a number of different objects are used to demonstrate the capabilities of object scripting. The first object is a window part. The following script is used to open the EMPLOYEE INFORMATION window rather than a visual connection.

```
/* Open G4TCSDetailViewSwing and set focus on */
/* empno field. */
public void DisplayG4TCSDetailView() {
    try {
        getG4TCSDetailViewSwing().show();
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
        hptHandleException(ivjExc);
    }
}
```

#### **Script to open the EMPLOYEE INFORMATION window**

Secondly, scripts are created to control the visibility of the UPDATE push button. The UPDATE push button should only appear when the UPDATE or DELETE operation is selected and when a valid employee record is entered.

```
/* Hide Update button from user until valid row */
/* is returned. */
public void hideUpdateButton() {
    try {
        getJUUpdate2().setVisible(false);
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
        hptHandleException(ivjExc);
    }
}

/* Make Update button visible. */
public void showUpdateButton() {
    try {
        getJUUpdate2().setVisible(true);
    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
        hptHandleException(ivjExc);
    }
}
```

#### **The scripts used to hide and show the UPDATE push button**

The following scripts are used to control the message area field. The setErrorMsg script illustrates how you can move data between your VAGen working storage record data items and the text and label fields on your GUI. It also shows how you can change the color of your message area to indicate an error condition.



```

/* Highlight message area with bright red. */
public void setErrorMsg() {
    try {
        getJError_msg().setForeground(java.awt.Color.red);

getJError_msg().setText(getRWS_G4TCS01_02().getStringData("ERROR_MESSAGE"));

    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
        hptHandleException(ivjExc);
    }
}

/* Clear/Reset the message area before next operation. */
public void clearErrorMsg() {
    try {
        getJError_msg().setForeground(java.awt.Color.black);
        getJError_msg().setText(" ");

    } catch (java.lang.Throwable ivjExc) {
        handleException(ivjExc);
        hptHandleException(ivjExc);
    }
}
}

```

#### ***Scripts to change and reset the color of the message area***

```

/* Prevent user from updating the fields. */
public void disablePanel() {
    try {
        getLastdataJTextField1().setEnabled(false);
        getFIRSTdataJTextField1().setEnabled(false);
        getMIDDLEdataJTextField1().setEnabled(false);
        getDEPTdataJTextField1().setEnabled(false);
        getSEXdataJTextField1().setEnabled(false);
        getSTREETdataJTextField1().setEnabled(false);
        getCITYdataJTextField1().setEnabled(false);
        getSTATEDataJTextField1().setEnabled(false);
        getZIPdataJTextField1().setEnabled(false);

        } catch (java.lang.Throwable ivjExc) {

        handleException(ivjExc);
        hptHandleException(ivjExc);
    }
}
}

```

#### ***Scripts to disable all Fields in JPanel1***

This example code shows how the sample scripts were coded inline with the rest of the 4GL logic using the EZESCRPT function. In the case of the inquiry operation, fields are disabled to prevent the user from updating any of the fields. Without the ability to use EZESCRPT functions inline, these updates would have had to be handled visually or with a performRequest.

# Object Scripting made easy using the VAGen Script Wizard

by Alex Akilov, VisualAge Generator Developer

The VisualAge family of products is based on the notion of visual programming using the Visual Composition Editor. This means that graphical user interfaces (GUIs) are constructed visually using a palette of parts that are “wired” together using connections that describe the application’s behavior.

Unfortunately, when you want to describe anything beyond basic application behaviors, you either develop it with a layered approach, which requires some understanding of object-oriented concepts (e.g. encapsulation, public interfaces) or you end up creating what is affectionately known as “spaghetti” code.

Using hundreds of connections that overlap each other makes it almost impossible to follow the flow of the graphical user interface program (not to mention the startup time of opening the composition editor on the window and/or the runtime costs of having to create and manage all of those connection objects).

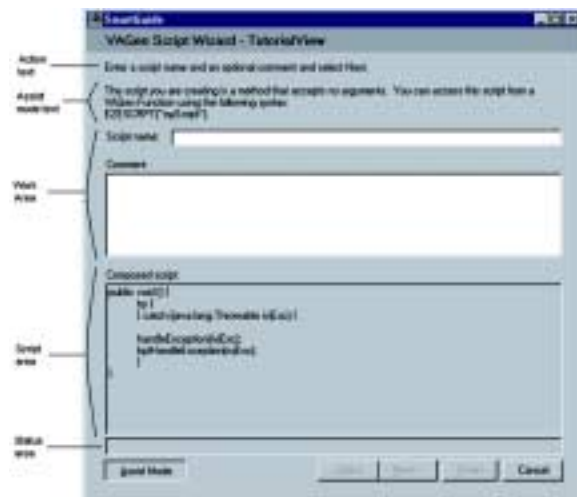
To augment visual programming, you usually have to add 4GL code to make conditional connections or to have a single event trigger multiple operations. The 4GL has no built-in constructs for working with GUI objects although you can use a couple of data-structure-based mechanisms known as data triggers and the performRequest construct to help communicate from the 4GL logic to the GUI world. These constructs are not easy to use and they don’t execute in synchronous fashion inline in your 4GL, which makes it difficult to schedule all of your GUI processing in a manageable fashion. The communication between the 4GL logic and the GUI objects is greatly enhanced in VisualAge Generator Version 4 through the new EZESCRPT function.

In Version 3.x of VisualAge Generator you could also use **Event-to-Script** connections to implement some of your GUI behavior using Smalltalk scripts but that assumes you know the Smalltalk language. VisualAge Smalltalk has a tool called **Subpart Feature Syntax**, which does part of the job by allowing you to paste a reference to a part or property or method visible within the current context, into a free-form text area.

However, you are still expected to complete the statement manually by using the output of this tool in the free-form text editor, which requires you to be comfortable with Smalltalk syntax and the Smalltalk method structure. Version 4.0 of VisualAge Generator adds another twist; you can now define your GUIs in either Smalltalk or Java, which have very different syntax rules. VisualAge for Java has an **Add Method** wizard that helps you add an empty method definition to your class, but you’re still expected to type in its contents. Java also has a contextual syntax assistant that allows you to get a list of choices to complete what you’ve started typing, but that still assumes that you know the basic syntax rules of the language.

The VAGen Script Wizard is designed to help you specify the procedural portion of your object-oriented program without having to know the syntax of the object-oriented scripting language. Also, you do not need to remember the objects involved in your program and the properties and protocols implemented by these objects. Rather than dropping into a free-form editor to code your logic, you can use the Script Wizard (or SmartGuide) to specify your method in a step-by-step, prompted fashion by simply picking objects, properties, and methods from a filtered list based on the current specification and context of your object.

After you make a selection, the appropriate code is instantaneously displayed in the script composition area, complete with any necessary type conversions and matching parentheses. You do not need to overwrite any portion of the statement that gets pasted in. But if additional arguments are required, then you are prompted as many times as are necessary with a new list of valid object references that can be used to complete this portion of the statement.



You can enter as many statements as you need and the wizard prompts you for local variables and automatically generates some default exception handling logic that you can then customize using a standard free-form editor. The wizard also allows you to add free-form statements and/or literal expressions to your method if you are familiar with the Java or Smalltalk syntax or want to use a literal value instead of a reference to one of the objects in your lists.

If you are not a Java or Smalltalk programmer, the wizard enables you to add object-oriented code by simply dropping some parts on the visual composition editor and connecting a single event to this new script. This script can completely drive the application procedurally and still provide the same ease of use that visual connections provide in the visual composition editor.

The script wizard is included in both the Java and Smalltalk flavors of VisualAge Generator. The Smalltalk version is customized for building scripts using Smalltalk syntax and the Java version uses Java syntax. The “Action text” and “Assist mode text” sections change as you proceed from step to step; the “Status Area” contains contextual messages based on user actions in the current step. The “Work area” changes to show a different set of objects depending on the step you’re on.

### Invoking the Script Wizard

On Smalltalk, if you are editing a visual class (one that has visual composition information), you can now invoke the VAGen Script Wizard from the Script Editor page of the Composition Editor window by clicking the tool icon under the Subpart Feature Syntax icon. It prompts you for the name of the new Smalltalk method you’re creating and it checks the name, as you type it, for valid Smalltalk method names. A message is displayed in a status area of the wizard as soon as an invalid name specification is detected. The **Next** button is disabled until you have entered a name that will be allowed by the compiler.

On Java, you can invoke the wizard from any browser that contains a **Classes** or **Methods** menu or a menu that is related to a selected Class in a list. For example, if you are working with the main Workbench window of the VA Java environment, you can expand a project and a package within it and select a class in the package. Once a class or one of its methods is selected, you can then go to the **Selected** menu and choose **VAGen Script Wizard** to invoke the wizard to add a method to that class.

After you have typed the name, you can optionally enter a method comment. The text that you type is automatically displayed in the composed script area of the wizard. You can then press the Next button to go to the next step in the wizard.

The next step is to build the local variables list. You can type in the name of a local variable that you think you’ll need and press the Add button. You can add and remove as many variable names as you need. You can also return to this step later as you discover that you need more variables and add them then without losing any of the code that you’ve already composed. On Java you also need to enter the type for the variable since Java is a typed language and expects everything to be properly declared.

This figure shows the local variables page of the Script Wizard on Smalltalk. Note how the “Action text” and “Assist mode text” sections have changed to contain a new explanation of what’s expected at this step. The Work area contains different objects to prompt for the needed information, whereas the Composed Script area show the specification you’ve entered so far, which includes the name of the method and the comment as well as the myLocal1 variable specification.



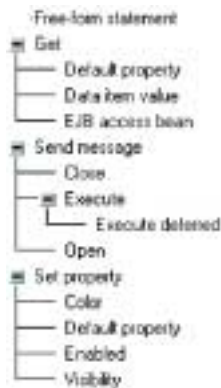
The same page on the Java wizard is shown below. Note that the “Work area” is completely different since additional information is needed for each local variable you declare. The “Action text” prompts you to specify the type.



The next step is called the statement pattern list. This list allows you to choose what type of statement you’d like to build. You can choose a free-form or a predefined statement type. The following figure shows the current list of predefined statement types that are included:

**You do not need to remember the objects involved in your program and the properties and protocols implemented by these objects.**

## Object Scripting made easy using the VAGen Script Wizard, continued



Once you select a statement type from the above list, you are then shown a parts list (or beans list in Java terminology) that contains the list of all the current parts that are visible within the current class, that is, the parts that you dropped on the free-form surface of your visual composition window. This list of parts is filtered to show only the parts that are valid targets for the statement pattern you've selected. Furthermore, if the statement pattern is one that allows you to work with features of this part (e.g. the color or some other property) the parts list can be further refined to show the valid properties under that part.

One of the advantages of the Smalltalk language is that it has a very simple Backus Naur Form (BNF). All statements essentially are composed of objects followed by messages with potential arguments. Java is very similar, but it adds traditional constructs such as looping and conditional keywords (among others). The Script wizard uses this fact in the way it collects the parts of the statement. Every statement pattern is essentially composed of a target expression specification followed by a source expression specification. A target expression is either an object or one of its properties, or an expression containing the object and a message (without the arguments). The source expression is one or more argument objects, properties or message sends (this time with arguments). Therefore, a statement pattern is typically concluded by prompting for a single target expression followed by "0-to-many" source expressions. An expression prompter is a list of parts with (or if not needed contextually, without) the properties and methods that the object implements.

For example, if you select the **Get** pattern, pick one of your local variables as the target object (the **Get** pattern implicitly defines the assignment operator as part of the target expression), and then press **Next**, you'd potentially see the following list of source parts that are valid to be assigned to your local variable. Note that the **Parts** category contains subcategories such as **Label Parts** and **Data Parts** that can be further expanded to show the list of parts and their **Properties** and **Methods** that can be used as the source expression of the statement.



Once you pick the source object, you may be prompted again to select additional source objects to be used as additional arguments that might be needed by any part of your statement. If no additional arguments are required, you are returned to the statement pattern list where you can then start a new statement, or finish and compile your composed script into the class you're working with.

One of the limitations of the current implementation is that the list of patterns does not yet include looping and conditional statement patterns. In a future release, we hope to implement this function, but in the meantime, the free-form statement pattern can be used to create conditional and looping statements as well as other statements that are not yet included in the default pattern list.

**The VAGen Script Wizard is one of the tools provided to help you embrace object-oriented programming at your own pace.**

### Summary

Constructing GUIs using VAGen has always been a little bit awkward since the 4GL in VAGen wasn't originally designed to operate with GUI constructs. The languages necessary to express processing in a GUI program usually require sophisticated object-oriented constructs, which are not currently present in the VAGen 4GL. Visual programming allows you to describe most of the processing in the GUI, but when you want to exercise a little more control over your GUI, beyond what the visual programming paradigm gives you, the new EZESCRPT construct and the VAGen Script Wizard come to your rescue.

The VAGen Script Wizard is one of the tools that VAGen provides to help you embrace object-oriented programming at your own pace. If you are comfortable with Java or Smalltalk, the wizard still provides quite a bit of value in that it allows you to code your logic in prompted fashion instead of having to remember all the various objects involved in your program as well as their properties and methods. And, if you're using Java, all the complex type conversions that you might have to do are handled for you. However, if you are not familiar with Smalltalk or Java, this is a great way to learn the basics you need to know to be able to take advantage of the powerful capabilities of these languages. ■

# DB2 for OS/390 hints and tips

by *Mitch Johnson, WebSphere/VisualAge Consulting Services*  
and *Jim Eberwein, VisualAge Generator Support*

This article provides some hints and tips for working with and administering DB2 on MVS and OS/390. These hints and tips can help your team become more productive, improve security, and simplify administration tasks.

## DB2 authorization ID translation

To access remote DB2 Tables from VisualAge Generator during development, many customers have adopted the practice of sharing a common SQL userid and password amongst their developers. This enables the database administrator to define a manageable set of synonyms for all of the relational tables that might be accessed by VisualAge Generator. However, sharing a password could easily be viewed as a serious violation of security policy. There is an alternative available to DB2 for MVS V4 and DB2 for OS/390 V5 administrators that alleviates the need to share a userid and password. This alternative uses a facility provided by DB2 and does not require any additional software.

For many customers, the SQL userid is the same as the DB2 authorization ID (AUTHID). An alternative to sharing a SQL userid is to use a DB2 facility that can translate the SQL userid (AUTHID) provided via a SNA DRDA connection to a different authorization ID. Validation of the SQL userid and password with the external security manager is still performed, but the translated authorization ID is used to control access to DB2 resources. This facility is implemented by inserting information into tables SYSIBM.LUNAMES and SYSIBM.USERNAMES (DB2 for OS/390 V5) and SYSIBM.SYSLUNAMES and SYSIBM.SYSUSERNAMES (DB2 for MVS V4).

When DB2 receives a connection request, the LUNAME of the remote client is checked for a match in the SYSIBM.LUNAMES table. If the LUNAME is found in the table and the USERNAMES column has a value of 'I' or 'B', then the SQL userid (AUTHID)

and/or the LUNAME in the request are checked for the best match in the SYSIBM.USERNAMES table. If a match is found (see Table 1) and the NEWAUTHID column has a non-blank value, then this non-blank value is used as the DB2 authorization ID, otherwise the original SQL userid (AUTHID) is retained. If no match is found, the connection request is rejected with a SQL error code -904.

## DB2 Authorization ID translation: Example

**Step 1.** Insert the following rows into the SYSIBM.LUNAMES table:

```
INSERT INTO SYSIBM.LUNAMES(LUNAME,USERNAMES) VALUES(' ',O');
INSERT INTO SYSIBM.LUNAMES(LUNAME,USERNAMES) VALUES('NR50C16I','I');
INSERT INTO SYSIBM.LUNAMES(LUNAME,USERNAMES) VALUES('NR50506I','B');
```

The USERNAMES values indicates the request direction a row handles, 'O' handles outbound requests, 'I' handles inbound requests, and 'B' handles both directions. If there are no provisions for a particular LUNAME (either a blank LUNAME or an explicit entry), that LUNAME is ignored and no authorization ID translations are performed.

LUNAME	USERNAMES
blank	O
NR50C16I	I
NR50506I	B

**Step 2.** Insert the following rows into the SYSIBM.USERNAMES table to associate a translated authorization ID with a connection authorization ID or LUNAME.

AUTHID	LUNAME	NEW AUTHID	Results
Userid	LU Name	blank	Userid is used as the authorization ID for this LU Name
Userid	LU Name	non-blank	NEW AUTHID is used as the authorization ID for this LU Name
Userid	blank	blank	Userid is used as the authorization ID for all LU Names
Userid	blank	non-blank	NEW AUTHID is used as the authorization for all LU Names
Blank	LU Name	blank	SQL AUTHID is retained as the authorization ID for this LU Name
Blank	LU Name	non-blank	NEW AUTHID is used as the authorization ID for this LU Name

Table 1. Precedence order when searching SYSIBM.USERNAMES

## DB2 for OS/390 hints and tips, continued

```
INSERT INTO SYSIBM.USERNAMES
  (TYPE,AUTHID,LINKNAME,NEWAUTHID) VALUES('I',' ','NR50C16I','TESTSQL ');
INSERT INTO SYSIBM.USERNAMES
  (TYPE,AUTHID,LINKNAME,NEWAUTHID) VALUES('I','MITCH ','NR50C16I',' ');
INSERT INTO SYSIBM.USERNAMES
  (TYPE,AUTHID,LINKNAME,NEWAUTHID) VALUES('I',' ','NR50506I ',' ');
INSERT INTO SYSIBM.USERNAMES
  (TYPE,AUTHID,LINKNAME,NEWAUTHID) VALUES('I','MITCH','NR50506I ',' ');
```

AUTHID	LINKNAME/LUNAME	NEWAUTHID
blank	NR50C16I	TESTSQL
MITCH	NR50C16I	blank
blank	NR50506I	blank
MATT	NR50506I	TESTSQL

In the following table, the results of these sample rows is shown. SQL userids MATT, JIM, and ORION have been translated to authorization ID, TESTSQL.

SQL Userid (AUTHID) from NR50C16I	Final DB2 Authorization ID
MITCH	MITCH
MATT	TESTSQL
JIM	TESTSQL
ORION	TESTSQL

In the following table, SQL userid MATT has been translated to authorization ID TESTSQL when using LUNAME NR50506I.

SQL Userid (AUTHID) from NR50506I	Final DB2 Authorization ID
MITCH	MITCH
MATT	TESTSQL
JIM	JIM
ORION	ORION

In the following table, LUNAME NRR5N600, is not in SYSIBM.LUNAMES with either an 'I' or a 'B' in the USERNAMES column; therefore, no userid translations are performed.

SQL Userid (AUTHID) from NRR5N600	Final DB2 Authorization ID
MITCH	MITCH
MATT	MATT
JIM	JIM
ORION	ORION

Simply by controlling the contents of these two tables, a shared SQL userid is no longer needed. Each user can be authenticated by the security manager using an unique userid and password and each can connect directly or via a gateway using a common authorization ID.

### RACF list of groups access

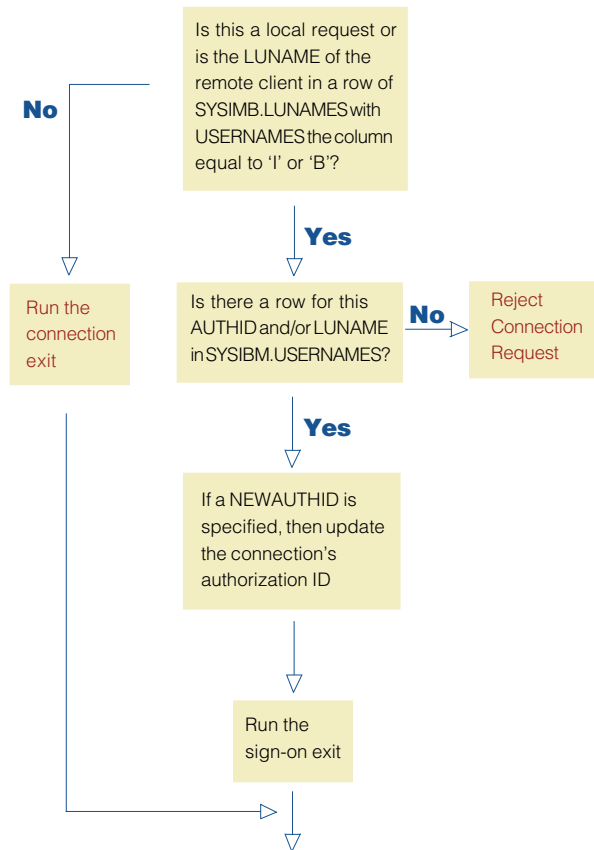
DB2 also provides a useful facility to control access to DB2 tables and to simplify administration. This facility controls access to DB2 tables using RACF groups. In this case, an individual user does not have explicit access to a table. Instead, the user has access by being connected to a RACF group that has table privileges. RACF group checking is available when RACF list of groups access checking is enabled (SETROPTS GRPLIST) and when the sample DB2 connection (DSN3@ATH) and/or signon (DSN@SGN) exits are enabled. The connection exit is invoked for a request from a TSO user or for a remote client when the LUNAME is not in SYSIBM.LUNAMES. The signon exit is invoked for a remote client when the LUNAME is in SYSIBM.LUNAMES. Note: The installation job (DSNTIJEX) that installs samples of these exits is optional and may not have been submitted during the installation of DB2.

The following scenarios demonstrate how the use of RACF list of groups access checking along with DB2 authorization translation can be used to simplify administration. Note that if the sample signon exit is used in conjunction with SQL userid translation, the translated authorization ID must be a valid RACF user.

In this example we have 4 RACF userids which have the following RACF group access:

Userid	Default Group	Group List
MITCH	TSTVGEN	TSTVGEN
MATT	VISGEN	VISGEN,TSTVGEN
JIM	VISGEN	VISGEN,TSTVGEN
ORION	VISGEN	VISGEN

In this scenario, we have granted select privileges on table TESTSQL.EMPLOYEE to RACF group TSTVGEN and we have enabled RACF list of groups access checking. The following diagram represents the flow.



Flow diagram of the SQL authorization process

### Default connection and signon exits

The following examples of default connection and signon exits are based on the sample input to the DB2 tables provided in the DB2 Authorization ID Translation section of this article.

On TSO, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE` produces the following result: RACF list of groups access checking is not enabled with the default connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	MATT	-551 (not authorized to SELECT from table)
JIM	JIM	-551 (not authorized to SELECT from table)
ORION	ORION	-551 (not authorized to SELECT from table)

Using a remote client with LUNAME NR50C16I, the SQL command, `SELECT * FROM EMPLOYEE`, produces the following result: MATT, JIM, and ORION have access to the table because their authorization ID's have been translated to TESTSQL.

**An alternative to sharing an SQL userid is to use a DB2 facility. . .**

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50C16I, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, results in a SQL -551 for MITCH because RACF list of groups access checking is not enabled with the default signon exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50506I, the SQL command, `SELECT * FROM EMPLOYEE`, produces the following result: MATT has access to the table because his authorization ID has been translated to TESTSQL.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME NR50506I, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, results in a SQL error -551 for MITCH and JIM, because RACF list of groups access checking is not enabled with the default signon exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	-551 (not authorized to SELECT from table)
ORION	ORION	-551 (not authorized to SELECT from table)

## DB2 for OS/390 hints and tips, continued

Using a remote client with LUNAME NRR5N600, the SQL command, `SELECT * FROM EMPLOYEE`, results in authorization ID translations since there is no provision for LUNAME NRR5N600 in SYSIBM.LUNAMES.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	MATT	MATT.EMPLOYEE is an undefined name
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME NRR5N600, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, results in error SQL -551 for all users since RACF Group access is not enabled with the default connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	MATT	-551 (not authorized to SELECT from table)
JIM	JIM	-551 (not authorized to SELECT from table)
ORION	ORION	-551 (not authorized to SELECT from table)

### Sample connection and signon exits examples

The following examples of DB2 user exits are based on the sample input to the DB2 tables provided in the DB2 Authorization ID Translation section of this article.

On TSO, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, produces the following result: RACF list of groups access checking is enabled with the sample connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	MATT	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	Rows retrieved from TESTSQL.EMPLOYEE
ORION	ORION	-551 (not authorized to SELECT from table)

Using a remote client with LUNAME NR50C16I, the SQL command, `SELECT * FROM EMPLOYEE`, produces the following result: MATT, JIM, and ORION have access because their authorization IDs have been translated to TESTSQL.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50C16I, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, produces the following result: MITCH now has access to the table for users MITCH, MATT, and JIM because RACF group is enabled with the sample connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50506I, the SQL command, `SELECT * FROM EMPLOYEE`, results in access to the table for MATT since his authorization ID has been translated to TESTSQL.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME NR50506I, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, results in MITCH and JIM having access to the table because RACF list of groups access checking is enabled with the sample signon exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	Rows retrieved from TESTSQL.EMPLOYEE
ORION	ORION	-551 (not authorized to SELECT from table)



Using a remote client with LUNAME NRR5N600, the SQL command, SELECT \* FROM EMPLOYEE, results in no authorization ID translations since there is no entry for LUNAME NRR5N600 in SYSIBM.LUNAMES.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	MATT	MATT.EMPLOYEE is an undefined name
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME NRR5N600, the SQL command, SELECT \* FROM TESTSQL.EMPLOYEE, results in access to the table for users MITCH, MATT, and JIM because RACF group is enabled with the sample connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	MATT	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	Rows retrieved from TESTSQL.EMPLOYEE
ORION	ORION	-551 (not authorized to SELECT from table)

### Sample connection and default signon exits

The following examples using the sample connection exit and the default signon exit are based on the sample input to the DB2 tables provided in the DB2 authorization ID Translation section of this article.

On TSO, the SQL command, SELECT \* FROM TESTSQL.EMPLOYEE, produces the following result: RACF list of groups access checking is enabled with the sample connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	MATT	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	Rows retrieved from TESTSQL.EMPLOYEE
ORION	ORION	-551 (not authorized to SELECT from table)

Using a remote client with LUNAME NR50C16I, the SQL command, SELECT \* FROM EMPLOYEE, produces the following result: MATT, JIM, and ORION have access because their authorization IDs have been translated to TESTSQL SQL

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50C16I, the SQL command, SELECT \* FROM TESTSQL.EMPLOYEE, results in no access to the table for MITCH because RACF list of groups access checking is not enabled with the default signon exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50506I, the SQL command, SELECT \* FROM EMPLOYEE, results in success only for MATT because his authorization ID has been translated to TESTSQL.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME LU NR50506I, the SQL command, SELECT \* FROM TESTSQL.EMPLOYEE, results in no access for MITCH and JIM because RACF list of groups access checking is not enabled with the default signon exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	-551 (not authorized to SELECT from table)
ORION	ORION	-551 (not authorized to SELECT from table)

## DB2 for OS/390 hints and tips, continued

Using a remote client with LUNAME NRR5N600, the SQL command, `SELECT * FROM EMPLOYEE`, does not translate any of the authorization IDs since there is no entry for LUNAME NRR5N600 in SYSIBM.LUNAMES.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	MATT	MATT.EMPLOYEE is an undefined name
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME NRR5N600, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, results in access to the table for user MITCH, MATT, and JIM because RACF list of groups access checking is enabled with the sample connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	MATT	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	Rows retrieved from TESTSQL.EMPLOYEE
ORION	ORION	-551 (not authorized to SELECT from table)

### Default connection and sample signon exits

The following examples using the sample connection exit and the default signon exit are based on the sample input to the DB2 tables provided in the DB2 authorization ID Translation section of this article.

On TSO, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, produces the following result: RACF list of groups access checking is not enabled with the default connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	MATT	-551 (not authorized to SELECT from table)
JIM	JIM	-551 (not authorized to SELECT from table)
ORION	ORION	-551 (not authorized to SELECT from table)

Using a remote client with LUNAME NR50C16I, the SQL command, `SELECT * FROM EMPLOYEE`, results in access to the table for user MATT, JIM, and ORION because their authorization IDs have been translated to TESTSQL.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50C16I, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, produces the following result: authorization ID Mitch now has access to the table because RACF list of groups access checking is enabled with the sample signon exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
ORION	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE

Using a remote client with LUNAME NR50506I, the SQL command, `SELECT * FROM EMPLOYEE`, results in access to the table for user MATT because his authorization ID has been translated to TESTSQL.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME NR50506I, the SQL command, `SELECT * FROM TESTSQL.EMPLOYEE`, results in access to the table for MITCH and JIM because RACF group access is enabled with the sample signon exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	Rows retrieved from TESTSQL.EMPLOYEE
MATT	TESTSQL	Rows retrieved from TESTSQL.EMPLOYEE
JIM	JIM	Rows retrieved from TESTSQL.EMPLOYEE
ORION	ORION	-551 (not authorized to SELECT from table)

Using a remote client with LUNAME NRR5N600, the SQL command, SELECT \* FROM EMPLOYEE, results in no authorization ID translations since there is no provision for LUNAME NRR5N600 in SYSIBM.LUNAMES.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	MITCH.EMPLOYEE is an undefined name
MATT	MATT	MATT.EMPLOYEE is an undefined name
JIM	JIM	JIM.EMPLOYEE is an undefined name
ORION	ORION	ORION.EMPLOYEE is an undefined name

Using a remote client with LUNAME NRR5N600, the SQL command, SELECT \* FROM TESTSQL.EMPLOYEE, results in no access because RACF list of groups access checking is not enabled with the default connection exit.

SQL Userid	DB2 Auth. ID	Results
MITCH	MITCH	-551 (not authorized to SELECT from table)
MATT	MATT	-551 (not authorized to SELECT from table)
JIM	JIM	-551 (not authorized to SELECT from table)
ORION	ORION	-551 (not authorized to SELECT from table)

### Binding VisualAge Generator Developer

The VisualAge Generator access module must be bound with the DB2 subsystem before tables can be accessed during development and test of your application. To grant the minimum DB2 authority required to perform this binding, the DB2 system administrator should issue the following commands:

```
GRANT BINDADD TO userid;
GRANT CREATE IN COLLECTION DEVELOP TO userid
```

After this authority has been granted, you can bind the VisualAge Generator's access module by issuing the following commands in a DB2 Command Window:

```
db2 connect to dbname user userid using password
db2 bind pathname\bindfile blocking all sqlerror continue grant public
      messages vgbind.msg datetime xxx
db2 connect reset
```

Parameters **messages vgbind.msg** and **datetime xxx** are optional, **pathname\** is the directory where you installed VisualAge Generator, and **bindfile** is the name of the bind file for your release (e.g., hptbnd231.bnd for V3.1 and hptbnd240.bnd for V4.0).

If VisualAge Generator is not bound to the DB2 subsystem prior to the first SQL access, VisualAge Generator will prompt you to prepare the access module. If you do not bind the access module prior to the first SQL access, you cannot specify the **datetime** parameter, which controls the format of date values. This might lead to SQL errors when accessing DB2 tables that contain date columns. If this occurs, the package must be freed to correct any bind-related errors. The

```
SELECT NAME FROM SYSIBM.SYSPACKAGE WHERE
      COLLID='DEVELOP'
```

SQL command will provide a list of VisualAge Generator package names:

```
DSN SYSTEM(DSN)
FREE PACKAGE(DEVELOP.HPT2W311)
END
```

To free or drop the package from DB2, invoke the following sequence of commands:

The package name depends on the version, release and fix pack level of the VisualAge Generator (e.g., HPT2W310 for V3.1, HPT2W311 for V3.1 at fix pack 1, and HPT2W400 for V4.0).■

# Some frequently asked questions for VisualAge Generator on OS/400

by Audra Downey, VisualAge Generator Developer and Ying Chen, VisualAge Generator Developer

**Q1: I have successfully generated and transferred an application to a remote AS/400 machine, but I didn't receive a job execution message or spool files to indicate a compilation of these programs. What happened?**

If you generated an application for the AS/400 target environment, but did not receive any job messages from the spool file (WRKSPLF) or did not receive any job run completion message (DSPMSG), it is possible that you might have incorrectly set up your compilation CL program from the template `efk24pbj.tpl`.

First, you should determine if you can manually submit the generated job. At an AS/400 interactive terminal type:

```
SBMDBJOB FILE(destlib/QVGNJOB) MBR(appname)
```

where `destlib` is the target library (defaults to `QGPL`) and `appname` is the name of the application. If you get the following message:

```
Job jobnum/QSPLJOB/jobname not scheduled. Error in
BCHJOB command.,
```

there is an error in the batch job submitted. One common problem is that the `JOBID` contained in the generated batch program was not set correctly. In the default template `efk24pbj.tpl`, `%ezeuserid%` is used to indicate the `JOBID` to use. `%ezeuserid%` uses the current workstation `userid`, which might not be the same as the remote AS/400 machine, which would then cause an error. You can replace `%ezeuserid%` in the template with a fixed value, or use a variable in place of `%ezeuserid%` and set its value in the generation option. For more information on templates, refer to the *VisualAge Generator Generation Guide (SH23-0263)*.

Next, if the `SBMDBJOB` command was successful, verify that you are successful in submitting the job. In the generated `appname.PRP` file, look for the following line:

```
:TYPE PART_TYPE='EZECLJ' WORKSTATION_EXT='.CLJ'
PREP_SUBMIT='Y'
```

At the same time, look for `TYPE='EZECLJ'` under the last `CONTROL` tag. This should be present to indicate that this file needs to be submitted.

Last, if you are running VisualAge Generator V4.0, then you should also check that you have the correct version of `ObjRexx` (1.0.3.0 or later) installed on your workstation. The required version of `ObjRexx` is included with VisualAge Generator and must be installed.

**Q2: How do I include debug view to the compile?**

You can modify your template so that it will always create debug information on compilation. Alternatively, you can modify the CL program submitted for the compile, recompile it, and obtain debug information. To modify the CL template so you always have debug information, first make a backup of the original. Depending on the type of program you have generated, the names of the templates to be modified are:

```
efk24pmn.tpl - VAGened Main application without SQL
efk24pcl.tpl - VAGened Called application without SQL
efk24psm.tpl - VAGened Main application with SQL
efk24psc.tpl - VAGened Called application with SQL
```

Then, include the line:

```
DBGVIEW(*SOURCE)
```

in all `CRTCBLMOD` and `CRTSQLCBLI` (SQL application contains both) commands. Use '+' to continue the command. For example:

```
CRTCBLMOD MODULE(%zedestlib%/%zembr%) +
SRCFILE(%zedestlib%/QVGNCLBL) +
SRCMBR(%zembr%) +
OUTPUT(*PRINT) +
OPTION(*NOSRC) +
DBGVIEW(*SOURCE) +
AUT(*LIBCRTAUT) +
OPTIMIZE(*BASIC) +
TEXT('VAGEN MAIN application')
```

Alternatively, if you do not always want debug information in your generated programs, then modify the generated CL program by adding the line, `DBGVIEW(*SOURCE)`. The CL program is contained in file `QVGNCLS`, and its member name is the same as the application you generated. Resubmit the job after you update the program using the following command:

```
SBMDBJOB FILE(destlib/QVGNJOB) MBR(appname)
```

**Q3: How do I debug a server application?**

First, you need to compile the server application with the debug view. (See question #2)

Then, start the client/server program so that a connection is established and maintained. A connection is established when a server program is first requested by the client. To maintain this connection, ensure that the client program does not exit.

Next, on your AS/400 terminal (separate from the client), type WRKACTJOB. Look for the job QZRCRSRVS under subsystem QSYSWRK or QUSRWRK depending on the installed OS/400 version on your machine. First, verify that this is the right job for this client connection by selecting 5 - *work with job option*, then selecting 10 - *display job log*. Look for "Servicing user profile" in the job log. This should be followed by your profile name. Look for "Client request - run program QVGN/QVGNRVR" and check the time stamp on the job as well.

If this is the correct job, write down its job number. You can obtain the job number on top of the Display Job Log panel. Exit WRKACTJOB.

On the command prompt, type:

```
STRSRVJOB JOB( jobnum/QUSER/QZRCRSRVS )
```

where jobnum is the number you wrote down associated with the job QZRCRSRVS.

Then, type:

```
STRDBG appname
```

to add breakpoints in the server program. Exit the server program source.

Continue running your client program so that it calls the server program. When the program reaches a breakpoint, your AS/400 terminal will display source code for you to step through.

When finished debugging, type:

```
ENDDBG
```

followed by:

```
ENDSRVJOB
```

**Q4: I have followed the instruction from *Running VisualGen Applications on OS/400* to move a prepared application to other OS/400 Systems, but my program doesn't run correctly.**

One common problem is moving the prepared application from a V3R1/V3R2 machine to a V3R6 (or later) machine, or vice versa. V3R1/V3R2 are IMPI machines, while V3R6 (or later) are RISC machines. Thus, a program compiled on one machine level will not run properly on the other.

The best way to move the application in this case is to generate the application directly to the target machine, and compile it there. You can also create a SAVF file from one machine to another - but you must include all generated data. After you restore your SAVF file on the target machine, you should submit the CL job that will trigger the compile. The CL job should be in your library, contained in the file QVGNJOB - with the name the same as the application name. If you don't

use the same JOBD or library for your file, you will need to modify the CL programs to reflect those values.

**Q5: How do I find out what is the latest PTF available for VisualAge Generator on the AS/400?**

You should contact your IBM customer representative for the latest PTF for VisualAge Generator on the AS/400. There are two information APARs that keep track of the latest PTFs for **VisualGen Runtime Services for OS/400**. The APARs are as follows:

```
II08904 for 5763VG1
```

```
II09438 for 5716VG1
```

**Q6: An application retrieves data from a database whose date is stored in the format yyyy-mm-dd and the map mask is yyyy-mm-dd. In test mode the date field appears in the correct format but when the program is run on the AS/400 the TUI displays the date in the yy-mm-dd format.**

You need to set an option when the SQL ILE COBOL object is created. The default for the date format is \*JOB. For the date to be retrieved in the yyyy-mm-dd format, the date format needs to be \*ISO.

You can do this in two ways:

1) You can modify the templates efk24psc.tpl and efk24psm.tpl to include the statement DATEFMT(\*ISO). This statement is inserted in the CRTSQLCBLI command. After this is done, you need to re-issue the submit job command.

For example:

```
CRTSQLCBLI OBJ(%ezedestlib%/%ezembr%) +
SRCFILE(%ezedestlib%/QVGNCLBLS) +
SRCMBR(%ezembr%) +
OBJTYPE(*MODULE) +
DATEFMT(*ISO) + OUTPUT(*NONE) +
OPTION(*SQL /* or *SYS /* +
      *QUOTE *APOSTSQL +
      *NOGEN /* enables 2-step compile */ +
      ) +
COMMIT(*CHG) +
CLOSQCSCR(*ENDACTGRP) +
/* DFTRDBCOL(default collection) /* +
TEXT('VAGEN CALLED application w/SQL')
```

2) On the AS/400, modify the CRTSQLCBLI command in the member destlib/QVGNCLS.appname (where destlib is the destination library and appname is the name of the generated application) to include DATEFMT(\*ISO). After modifying the member, type the following command:

```
SBMDBJOB FILE(destlib/QVGNJOB) MBR(appname)
```

where destlib is the destination library and appname is the name of the generated application. ■

# FUNCTIONS - Here at last!!

by Frieda Dollar, VisualAge Generator Developer

So, have you been a VAGen or CSP customer for some time now and been frustrated by not being able to pass arguments to a process or statement group? Have you been confused about the differences between processes and statement groups? Good news! VAGen V4.0 solves both of those problems for you! We have merged processes and statement groups into a single part type called a function.

We have further enhanced the new function part to allow arguments to be passed into a function and to allow that function to return a value. In addition, you now have the opportunity to define areas of storage that are known only to that function. We call those local storage areas. A side effect of all of this is that you can now qualify an item name used as a subscript! We even changed the syntax of our statements to use square brackets instead of round brackets to denote subscripts. Terminology has changed too. Scope now means what you have always thought it should. Wow! Let's back up and take a look at these thoughts one at a time to better understand functions and what their introduction into the product will mean to you.

## Scope

In the past, the term scope was used to determine where the characteristics of a data item were located. Were they local to the record or table that contained the item or were they global to the library and shared by multiple records and/or tables? This terminology proved confusing for all of us. You tried to make it mean what it means in other programming languages - who has access to the data in those items? a single process? the entire program? So, all existing references to global and local scope are changed to use the terminology shared and non-shared definition. Scope is now used the way you would expect. The scope of a data item or record indicates whether the data area is known to the entire program or whether it's use is restricted to the function in which it is named.

## Square brackets

As computer languages evolved, the newer languages chose to use square brackets (Ex: [ ]) to denote subscripts where older languages had used round brackets (Ex: ( )), also known as parentheses). Our use of round brackets has frustrated many of our customers who are familiar with the newer computing languages. Since we needed a way to denote the argument list when invoking a function, it seemed like a good time to 'bite the bullet' and change our syntax. Therefore, round brackets are now used to denote the argument list and square brackets are now used to denote the subscripts. Migration will automatically change them for you. In the long term this will make it easier to switch between VAGen and other languages.

## Qualified subscript names

Subscript names no longer have to be unique across an entire program. Instead, you have the ability to qualify a subscript item name with a record, table, or map name.

## Processes and statement groups merge

Our use of both processes and statement groups may have confused many of you. If you never really figured out all of the differences, you don't have to bother any more. We have merged these two very similar part types into a single part type - a function. Migration understands how to read old processes and statement groups and turn them into functions, changing the statement syntax as it goes. You don't have to re-code these unless you want to take advantage of the new capabilities provided.

## Local storage areas

For a function, you may specify any number of working storage records or items that are to be used as local storage areas for that function. The scope of reference for a record or item

named as local storage for a function is limited to that function only. If it is to be known by any other function, it must be passed to that function as an argument. If it is to be known to the caller, then it should not be a local storage definition. The same record or item can be named in the local storage list for more than one function. Each function gets a separate copy of the storage mapped by the definition. Local storage is not initialized upon entry into a function. Therefore, you should make no assumptions as to any of the local storage data area values.

## Functions with parameters and return values

All of the above enhancements were really outgrowths of our original design goal: to be able to pass arguments into a function, have that function receive those arguments as parameters, have the statements in the function reference the arguments by parameter names, and then optionally return a value to the calling function. Basically, we want to help you write reusable code. In order to do that, you need to be able to have a piece of code (a function) that uses generic names (parameter names) to access the data that is passed to it as arguments. We have given you the capability to pass items, working storage records, or literals as arguments. Within the function, you reference the data areas by the parameter names. Since parameters are passed by reference, we modify the storage areas for the arguments.

Let's look at how you invoke a function. In the past, you either PERFORMed a process or you invoked a statement group simply by naming it. Since there isn't a difference between processes and statement groups any more, we have dropped the PERFORM keyword. Will we forget and type it in? Yes! Will statement validation tell us that it is invalid syntax? Yes! To invoke a function, you simply state it's name and follow the name by the argument list.

The argument list must be included in round brackets and individual arguments are separated by commas. The round brackets are required even if you don't have any arguments. That's so we can tell the difference between a function name and a data item name. Without the round brackets, there were some cases that we had trouble with. Oh, in case you're worried about all of your old code, migration took care of removing the PERFORM keyword for you and it also put in empty argument lists on all of your old process and statement group invocations.

OK, that's all very nice, you say. But, sometimes what you really want to do is to invoke a function and, when all is said and done, have some resulting value assigned to a variable. You could pass that variable in as an argument, but that is clumsy and doesn't make for very readable code. To solve that problem, we have given you the ability to define a return value for a function. You can then specify an argument on the EZERTN function word and the value of that argument is what gets assigned to the variable on exit from the function. That means that you can say `BIGGEST = MAX(A, B, C, D, E);` we call invoking a function in this way using a function "inline".

### Accessing state information for parameters

State information for a record received as a parameter is not available. However, if you define an item parameter as an SQL item or as a map item and you pass an SQL item or map item as an argument, then the SQL state information or the map item state information is available to the logic of the receiving function. This is so you can have reusable routines that test for and modify the state of the item. For

example, you can code the following statements in the receiving function:

```
TEST sql-item TRUNC true, false;  
or SET map-item MODIFIED;
```

### Generic data typing

Defining item parameters with one of the normal data types specifies strong typing of the parameter. For these data types, you specify the bytes and decimals values. Test and generation will require exact matches between arguments and parameters when strong typing is used.

But "Wait a minute," you say. "I want to be able to say `BIGGEST = MAX (A, B, C, D, E);` and sometimes pass binary values as arguments and sometimes pass packed values. Or maybe I want a function to do string manipulation on a string that is sometimes 5 bytes long and sometimes 10 bytes. Can I do that?" Sure you can! We have relaxed the typing restrictions by introducing a number of ANY item types that can be used only in the definition of parameters for a function. For each of the string data types, there is a corresponding ANY data type that will accept any length string of that basic type. There is also an ANYNUMERIC data type that will accept any length, any number of decimal places, any numeric data type as input. Using ANY data types specifies loose typing of parameters.

What is really happening is that we are allowing you to write a single function and we are figuring out all of the combinations of ways that you are invoking that function within a program. Where necessary, we are generating multiple copies of the function code, one copy for each unique combination of loosely typed argument definitions used within the program. If you invoke the same function many different ways in the same program, use care because your generated program could get quite large.

### EZEwords become functions

Much of the support that we have supplied in the past, via the special EZEwords available on the CALL statement, can really be viewed as VAGen supplied functions. In keeping with that thought, we have converted their invocation syntax to match that of the functions that you would define. Most of these functions are string and math functions, but there are a few others as well. Some are appropriate to use as "inline" while others are "standalone". A new EZEword, EZEREPLY, is provided for use with the EZEword functions. It controls the setting of EZERT8 after the function invocation in the same way as the REPLY option on the CALL statement. During migration, the CALL statements for these EZEword invocations are converted to a function invocation for you. Statements to set the value of EZEREPLY are added as necessary.

### The future

Hmmmm . . . so now you can write reusable functions, passing in arguments and getting back a return value. Not bad! By looking at the parameter list definition and the return value definition, you can determine how to invoke the function without having to study the logic of the function to decide what you need to add to your program. Not bad! Do you want more? Sure you do! We do too. We have lots of ideas for additional capability to add. There are numerous restrictions that we had to enforce in this first pass at functions in order to keep the scope and size of the release under control. We'd like to lift those restrictions. The list of possibilities for the future is long, but instead of trying to do everything all at once, we'd like your input to help us set priorities. Let us know using the online reader comment form available on the VisualAge Generator website, <http://www.ibm.com/software/ad/visgen/> ■

**Hmmmm . . . so now you can write reusable functions, passing in arguments and getting back a return value.**

# Functions, local storage, parameters: everything you wanted to know and then some

by Guy Slade, VisualAge Generator Developer

This article consolidates several important topics regarding the VAGen Function part into one easy-to-read document. The Function part replaces the Statement Group and Process parts used in all previous releases of VisualAge Generator (VAGen) and Cross System Product. There are some concepts new to VAGen that are part of the Function part such as passed parameters and local storage. To help you work effectively with these new concepts, this article addresses the following topics: name resolution, argument passing, and the function return value.

## Name resolution

If you reference an unqualified data item name or record/table name within 4GL code in a function, how does VAGen decide which object you are referencing? The answer is obvious if it is a data item that exists in a single record but what happens if it exists within two records? Version 4.0 of VAGen adds to the complexity of this scenario with the introduction of functions. With the implementation of functions, you have to think about passed parameters and local function storage. In this model, a data item can exist in several different working storage records. It can also be specified in the function's passed parameter list or its local storage list, and it can be defined as a data item that exists in the I/O object (for example, if the function conversed a map then the I/O object would be the map).

## Name resolution rules

The following is a search order list starting from the first place that VAGen looks to resolve the name through to the last place VAGen looks. The name could

be a record, table, or unqualified data item.

1. VAGen checks to see if the name corresponds to a data item specified in the functions local storage list or passed parameters list. Parameter and local storage data items are not allowed to have the same name so VAGen can check both of these areas without fear of ambiguous reference. For example, you can't define a parameter data item with a name of BOB and also define a local storage data item called BOB. If the name is resolved to a data item in the functions local storage or parameters list, then the resolution is successful.

2. VAGen checks to see if the name is:

- The function's I/O object or a data item contained within the function object.
- A record specified in the functions parameter list or a data item contained within the record
- A record specified in the functions local storage list or a data item contained within the record.

If the name is not unique across this category, then it is an ambiguous reference and an error is issued.

If the name is found (this could have been resolved to either a data item, record, or map) and is not ambiguous, then the resolution is successful.

3. VAGen checks to see if the data item is:

- Any other function's I/O object or a data item contained within any other function's object
- The program's main working storage record or a data item in this record
- A record or table specified in the program's list of tables and additional records

- A data item within a record or table specified in the program's list of tables and additional records

If the name is not unique across this category, then it is an ambiguous reference and an error is issued.

If the name is found (this could have been resolved to either a data item, record, or map) and is not ambiguous, then the resolution is successful.

When writing 4GL code, it is important to know and understand these resolution rules so that when you reference a data item or record, the action that you have coded executes against the appropriate object. It is a good coding practice to qualify any data items used in 4GL scripting; however, this is not always possible. For example, data items that are specified as a function's parameter or local storage would not be fully qualified.

## Passing arguments

When invoking a function, you can pass arguments to the function. This, along with the fact that you can specify local storage for a function, allows you to write a truly reusable, self-contained function.

You also have the ability to specify ANY data types when defining a function's parameter list. For example, you may have a function that accepts two numbers as arguments, adds them together and returns the result. To provide the function with the flexibility to accept any type of number (i.e. bin, pack num) of any length, you would define the two parameters as ANYNUMERIC types.

Arguments are passed by reference. For example, program PROGA has working storage record, WREC1, defined in its additional tables and records list and this record has a data item defined,



named BOB. PROGA has a main function FUNCA defined. FUNCA has the following code:

```
/* invoke funcb passing a
parameter
WREC1.BOB = 'XXX';
FUNCB(WREC1.BOB);
WREC1.TIM = WREC1.BOB;
```

FUNCB is defined to accept the argument, SAM. FUNCB has the following code:

```
SAM = 'YYY'
```

Because arguments are passed by reference, this means that when FUNCB assigns 'YYY' to SAM, it also assigns 'YYY' to WREC1.BOB. So the last statement in FUNCA, "WREC1.TIM = WREC1.BOB;", results in 'YYY' being assigned to WREC1.TIM.

## Function return value

Rules for return values include the following:

1. If you invoke an inline function ( e.g. X = FUNCA(); ) then you must specify a return value for the function. The characteristics that you specify for the function's return value must match the characteristics of the data item that will be returned (i.e. the argument used in the EZERTN statement). For example, if the function's return value is defined as char 12 and the following EZERTN statement is used: "EZERTN(WS.ITEM1);", then ITEM1 would have to be defined as char 12 in the working storage record, WS. Currently you can not use the ANY data type when defining a function's return value; therefore, tight type-matching is performed.

2. If you specify a return value for a function and then exit the function via an EZERTN statement, you must supply an argument in the EZERTN call (e.g. EZERTN(TOTAL); ). The argument must either be a data item with characteristics that match those of the function's return value or a literal that matches the function's return value characteristics. If you are returning a numeric literal, the function's return value must be defined as a NUM type . You can also exit from a function that has a return value specified without the use of an EZERTN statement. In this case, a temporary data item is returned with the characteristics of the return value set to a default value (e.g., a blank for CHAR and a zero for NUM).

3. If you do not specify a return value for a function and you exit the function via an EZERTN statement, you cannot specify an argument in the EZERTN call.

## Functions and segmented transactions

The new flexibility that functions bring to the 4GL is a great step forward. Currently there are certain restrictions regarding functions, some of which are already mentioned in this article. One further restriction concerns segmented transactions. The restriction is that if you have generated a program as a main segmented transaction or have a segmented converse in a non-segmented program by using the EZESEGTR EZEWord, when the map is CONVERSEd, functions in the current processing stack cannot have parameters, local storage, or a return value defined. For example, suppose there are three functions: FUNCA, FUNCB, and FUNCC. FUNCA invokes

FUNCB, which in turn invokes FUNCC. Now suppose FUNCB has a local storage data item defined and FUNCC converses a map in segmented mode. If this occurs when you are testing in ITF, a hard error is issued. If this occurs when you are generating the program, a warning message is issued. If this occurs during runtime, an unrecoverable error occurs.

If you want to include reusable functions within a segmented program, then you must make sure that at any CONVERSE these functions are no longer in the processing stack. The code in the example could be designed so that FUNCA invokes FUNCB, FUNCB finishes processing and returns to FUNCA, and FUNCA invokes FUNCC which performs the CONVERSE without error.

The introduction of function parts to the VAGen 4GL language is an important step in allowing the coding of truly reusable functions. The new rules involved in name resolution and argument passing are more complex than before. However, armed with the information in this article, the documentation that comes with the VAGen product, and, of course, your own common sense, you should be able to code reusable parts and understand their behavior. ■

# VisualAge Smalltalk vs. VisualAge for Java: A simple comparison

by Jon Shavor, VisualAge Generator Developer

Now you have a choice! Prior to Version 4.0, VisualAge Generator was available only on the VisualAge Smalltalk environment. Now, with Version 4.0 you can also run VisualAge Generator on the VisualAge for Java environment. While there are some similarities, there are also some subtle, but important, differences between the two environments. If you are already familiar with VisualAge Generator V3.0 or V3.1, understanding the terminology differences will help you become more comfortable with the new VisualAge for Java environment.

## Library basics

Both VisualAge Smalltalk and VisualAge for Java have a library management system, ENVY. In VisualAge Smalltalk, ENVY components are stored in the manager or library. In VisualAge for Java, it is called the repository. The terms *manager*, *library*, and *repository* are synonyms. In VisualAge Smalltalk, your “working set” of objects is called an Image (abt.icx by default). In VisualAge for Java, it is called a Workspace (ide.icx by default). VisualAge Smalltalk uses “abt” as a prefix in many of its files and VisualAge for Java uses the prefix “ide”. For example, the VisualAge Smalltalk abt.ini file is similar to the VisualAge for Java ide.ini file.

VisualAge Smalltalk	VisualAge for Java	Description
manager, library	repository	ENVY library management system
Image (abt.icx)	Workspace (ide.icx)	working set of objects

## Windows

After starting VisualAge Smalltalk, the first window displayed is the Organizer. The corresponding main window in VisualAge for Java is the Workbench. The Workbench is the main window that you will use to navigate within VisualAge for Java. In VisualAge Smalltalk, there is a System Transcript window which, among other things, logs ENVY messages. In VisualAge for Java, the corresponding window is the Log. VisualAge Smalltalk’s Workspace dialog is displayed when you select File>>New or when a file is opened. The corresponding dialog in VisualAge for Java is the Scrapbook dialog. You can use these dialogs to execute code or edit a text file. Both VisualAge Smalltalk and VisualAge for Java provide browsers to look at components contained in the ENVY repository. In VisualAge Smalltalk there are the Configuration Map Browser and the Application Editions Browser. In VisualAge for Java there is a browser called the Repository Explorer.

VisualAge Smalltalk	VisualAge for Java	Description
Organizer	Workbench	main window
System Transcript	Log	ENVY library message log
Workspace dialog	Scrapbook dialog	Dialog to execute code or edit a file
Configuration Map browser, Application Editions browser	Repository Explorer	Facility to view components in the ENVY repository

## Library components

VisualAge Smalltalk has Configuration Maps to organize Applications. VisualAge for Java has Projects to organize Packages. In VisualAge for Java, a package cannot be loaded independently like a VisualAge Smalltalk application can. A package must be contained in a project. A Configuration Map is a list of applications that can be conveniently acted upon as a unit (load, import, export, etc). There is a “loose” relationship between a Configuration Map and an Application. An Application can be contained by multiple Configuration Maps. If these Configuration Maps are loaded, the edition of the Application released into the last Configuration Map loaded is in the image.

A Project contains one or more Packages. There is a “tight” connection between a Package and its containing Project. A Package can be in multiple Projects, but only one of these Projects can be in the workspace at a time. If you try to add another Project, containing a Package that is already in the workspace, an error occurs.

Another way to look at the “loose” vs. “tight” relationship is to recognize that in VisualAge Smalltalk, an application can exist by itself with no containing Configuration Map. However, in VisualAge Java, a Package cannot exist outside of a Project. It has to be contained in a Project.

In VisualAge Smalltalk, applications can have configuration expressions, prerequisites, and SubApplications. There are no equivalent constructs in VisualAge for Java. For example, there is no such thing as a “SubPackage” in VisualAge for Java. Configuration Maps can have configuration expressions and required maps. There are no equivalent constructs of these in VisualAge for Java.

**Note: See the “VisualAge Generator” section for a VAGen extension to VisualAge for Java that is similar to required maps.**

VisualAge Smalltalk	VisualAge for Java	Description
Application	Package	a group of functionally-related parts
Configuration Map	Project	a collection of related applications or packages

### Library management

The VisualAge for Java Workbench and the VisualAge Smalltalk Organizer have some similarities. For example, various library-related tasks (e.g. version, release, and copy) can be performed from both of them. However, some of the terminology has changed. For example, “Load Available” in VisualAge Smalltalk is “Add” in VisualAge for Java. In VisualAge for Java, you can use Add to load an existing component from the repository or create a new component. VisualAge Smalltalk uses “New” to create a component that does not already exist in the library. When loading a different edition of a component, VisualAge Smalltalk uses “Load Another Edition.” VisualAge for Java’s equivalent is “Replace With.” “Unload” is used in VisualAge Smalltalk to remove a component from the image. The equivalent function in VisualAge for Java is “Delete.” Like “Unload,” “Delete” removes the component from the workspace, not the repository. To remove something from the repository, use “Purge” for both VisualAge Smalltalk and VisualAge for Java. Anything purged can be restored with “Salvage” in VisualAge Smalltalk (“Restore” in VisualAge for Java) at a later time. Purging does not really remove the component from the repository, it just hides it from view.

VisualAge Smalltalk uses “Browse Changes” or “Browse Differences” to compare components. VisualAge for Java uses “Compare With.” In VisualAge Smalltalk, Configuration Maps and Applications have a “manager” and Classes have an “owner.” In VisualAge for Java, Projects, Packages, and Classes all have an “owner.” There is no concept of managing a component in VisualAge for Java. VisualAge for Java has combined the “Copy,” “Move,” and “Rename” actions under a cascaded menu called “Reorganize.” Also, “Version,” “Release,” and “Create Open Edition” have been combined under a cascaded menu called “Manage.”

VisualAge Smalltalk	VisualAge for Java	Description
Load Available, New	Add	load an existing component or create a new component
Load Another Edition	ReplaceWith	load another edition of a component
Unload	Delete	remove a component from the image/workspace
Purge	Purge	hide a component from view
Salvaged	Restore	restore a purged component to view
Browse Changes, Browse Differences	CompareWith	compare components
manager, owner	owner	Ownership of components

### VisualAge Generator

VAGen parts are stored conceptually as methods in both VisualAge Smalltalk and VisualAge for Java. Thus, VAGen parts have the same ownership issues. All VAGen Programs in one Application must be owned by the same user. Similarly, all VAGen Programs in one Package must be owned by the same user.

After the VAGen feature is loaded into VisualAge Smalltalk, the Organizer window includes a third pane. This pane lists the VAGen parts. The VAGen parts can be manipulated from the third pane of the VisualAge Organizer. There is no equivalent in VisualAge for Java. The Workbench is not modified to show VAGen parts. If you select a VisualAge for Java method, which represents a VAGen part in the Workbench, its corresponding External Source Format is displayed. This is similar to VisualAge Smalltalk. In VisualAge for Java, the primary window to work with VAGen parts is the VAGen Parts Browser. In VisualAge Smalltalk, you can navigate from the VisualAge Organizer or the VAGen Parts Browser. When the VAGen feature is added to the VisualAge for Java Workspace, the Open VAGen Parts Browser action, is added to the Workspace menu of the Workbench. This action opens the VAGen Parts Browser so you can work with the VAGen parts.

In VisualAge Smalltalk, when generating in batch, you can specify a Configuration Map to load. This Configuration Map and any required maps are loaded prior to generation. Because VisualAge for Java does not have the concept of required map or required project, this was a potential problem for VAGen batch generation on VisualAge for Java. To solve the problem, VAGen has added an extension to VisualAge for Java to provide a similar function. This extension is called Project List Part (PLP). The PLP is a generation options part that specifies the projects required by this project. When you start a batch generation in VisualAge for Java, you will specify a Project to load. If this project has a PLP, that PLP will be processed (required Projects will be loaded) prior to loading the Project. The PLP in the specified Project potentially could require other Projects that have PLPs. This extension enables you to generate in batch using VisualAge for Java in a similar way to VisualAge Smalltalk.

### Miscellaneous

The GUI builder in VisualAge Smalltalk and VisualAge for Java are similar. It is called the Composition Editor in VisualAge Smalltalk and Visual Composition in VisualAge for Java. There are many similar constructs and actions. There are also differences. Part categories and parts in the palette are displayed differently. Also, because there is not a direct correspondence between VisualAge Smalltalk parts and VisualAge for Java parts, the available parts are different between VisualAge Smalltalk and VisualAge for Java. Dropping and connecting parts are the same, but some of the available connectable feature names are different. For example, a button-clicked event in VisualAge Smalltalk is a button actionPerformed in VisualAge for Java. Another example is that “self” in VisualAge Smalltalk is “this” in VisualAge Java. ■

# I'll never forget . . . Whatshisname

By Jon Shavor, VisualAge Generator Developer  
and John Ormerod, VisualAge Generator Product Specialist

Do you ever have trouble remembering names, dyslexic fingers, typing long names over and over again? If you recognize any, or perhaps all of the above, then help is at hand.

A new feature that was introduced in VisualAge Generator V3.1 has been improved in V4.0 and in V3.1 FixPak 3. This feature enables you to paste the names of parts in your image into the Function Editor statements area. Some performance and usability issues were addressed to improve this feature, including the ability to control it using only keyboard commands.

To use this feature from the Function Editor, select **Paste Part Name** from the context menu. From the cascading menu, select the part type you want to paste. A window containing a list of those parts is displayed.



To filter the list, select the **Visibility** option. The three choices are: VAGen part, Package, and Image. The VAGen part choice limits the selection to what is visible within the context of the VisualAge Generator part. For example, if you open the Program Editor and double-click on a function, the list of records displayed in the Part Names window includes the function object (if it is a record), the program's working storage record, and any records in the program's table and additional records list. The Package choice displays all the parts of this type in the Java package where the function part is defined. The Image choice displays all the parts of this type that are loaded in the image.



The Paste Part Name feature also provides you the ability to paste a fully qualified part name. Fully qualified name is enabled for records, tables, and maps and allows you to paste a part referenced by the part, for example, an item within a record. The item can be pasted with or without the qualified record name (for example: EX99RSUPP.SUPPNO, which includes the record name).

## Performance improvements

In previous versions, when you opened the Part Names window, all the part definitions from the library were read in order to show the containing parts (eg: items in records). Reading all the parts requires a lot of resources. Now, the reading of the part definition is delayed until you expand the part (using Ctrl+E, or from the context menu). So, when you list records, you will not see the data items unless you expand the part or expand all the parts. This "on-demand" approach to displaying information significantly reduces the time it takes to display the Part Names window, especially if you have a large number of parts loaded into your image.

**. . . as you enter statements in the Function Editor using the keyboard, you can use Paste Part Name without touching the mouse.**

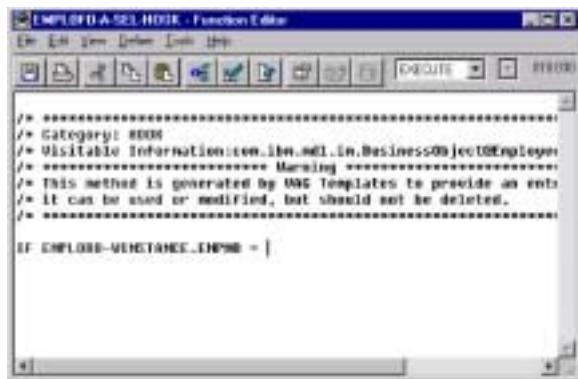


### Usability enhancements

If you prefer to use your mouse to paste part names, you still can. But now, as you enter statements in the Function Editor using the keyboard, you can use Paste Part Name without touching the mouse.

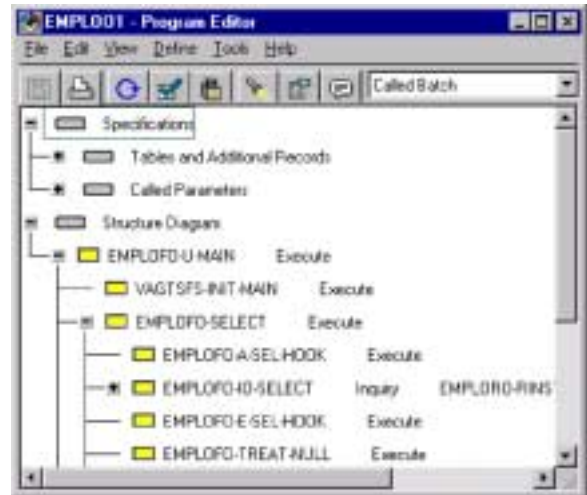
Other enhancements include the addition of Accelerator keys to expand a record, table, etc. The tabbing order on the Part Names window has also been improved. A setting is now available from the Part Names window to give you the option of automatically closing the window after you paste the part name. In addition, the settings on the Paste Part Name settings are saved and used the next time the dialog is shown.

To add the selected part name to the statement area, pressing Enter or clicking the Paste button adds the selected part name to the statement area. In the example shown, it adds the fully qualified item name after the IF statement.



### Example

In this example, you are editing a Function part that you opened from the Program Editor. You have already used the Part Names window and saved the appropriate settings: a Visibility setting of VAGen part, Close window, and Fully qualified name. These settings are saved when you close the Part Names window.



You need the name of an item from the working storage record. Hmm, you wonder, was that item named EX99RSUPP.SUPPNUM or EX99RSUPP.SUPPNO? While your cursor is in the statement area of the Function part, press Alt+R to open the Parts Name window for record. Use the down arrow to locate the record you want. Press the space bar to select the record. Press Ctrl+E to expand the record. Use the down arrow key to locate the item you want to paste. Press the space bar to select the item. Press Enter. The fully qualified data item is pasted into the Statement area and the Part Names window is closed. ■

# SHARE in Long Beach, February 25 - March 2, 2001

***Keeping with SHARE's exceptional tradition of outstanding education and professional connections for users of IBM technologies, SHARE in Long Beach is a week-long event that will deliver nearly 1,000 hours of learning opportunities across these high-interest categories:***

- Applications Development and Enablement
- Core Competencies for the IT Professional
- Domino on S/390 User Experiences
- e-Business Access to and from the Mainframe
- ERP on the Mainframe with DB2 and Oracle
- Linux in the Enterprise
- Linux on VM
- Linux on S/390
- OS/390 Release 10 User Experiences
- Software Asset Management
- Storage Area Networks (SANs)
- Systems Management
- Using Windows 2000 for e-Business

**On Thursday, March 1, the following VAGen sessions will be held:**

Session	Time	Speaker	Title
8385	9:30	Rusty Edmister	VisualAge Generator - Your Application Development Solution
8386	11:00	Rusty Edmister/ Pat Adams	VisualAge Generator - A Strategic IBM Application Development Environment
8387	1:30	To Be Determined	VisualAge Generator - Customer Experience
8388	3:00	Denise Hendriks	VisualAge Generator - Rapidly Building Web-enabled Java Apps
8389	4:30	Denise Hendriks	VisualAge Generator - User Record Technology Demo

Watch <http://www.shareinlongbeach.org> for the latest event, registration, travel and housing information.

# Planning for migration to VisualAge Generator 4.x

by *Debbie Prevost, VisualAge Generator Information Developer*  
and *Jeri Petersen, VisualAge Generator Services Consultant*

With VisualAge Generator 4.x, the development environment has been ported from Smalltalk to the Java platform on Windows NT. With VisualAge Generator on Java, you can easily enter the e-business arena with web applications that mix your own application components with reusable Java components being developed by other vendors and business partners.

Note: This article does not address migration from VisualAge Generator 4.0 to 4.5. Refer to the VisualAge Generator 4.5 and corresponding VisualAge for Java and VisualAge Smalltalk Readme documents and Migration Guides for information on migration from VisualAge Generator 4.0 to 4.5.

## Migration options available

Regardless of your current VisualAge Generator or Cross System Product release, you will need to migrate your applications to be able to use them with VisualAge Generator 4.x. Depending on your current platform and whether you want to migrate to 4.x on Smalltalk or Java, different migration paths are available and different considerations apply. The following table gives a brief overview of the migration options available.

Migrating from	Migrating to	Tools to Use
VAGen 3.x	VAGen 4.x on Smalltalk or Java (see notes 1 and 2)	<ul style="list-style-type: none"> <li>V3 to V4 Migration Tool</li> <li>VAGen Import</li> </ul>
VAGen 2.x	VAGen 4.x on Smalltalk (see notes 1 and 3)	<ul style="list-style-type: none"> <li>MSL Migration Assistance Tool</li> <li>VAGen Import</li> </ul>
Cross System Product	VAGen 4.x on Smalltalk or Java	<ul style="list-style-type: none"> <li>MSL Migration Assistance Tool</li> <li>VAGen Import</li> </ul>

### Notes:

1. None of the VisualAge Generator tools can be used to migrate existing GUIs to Java. You will need to recreate the GUIs using VisualAge Generator 4.x on Java. Then you can add the recreated GUI parts into the migrated packages containing your non-GUI code.

2. If you are migrating to VisualAge Generator 4.x on Smalltalk from VisualAge Generator 3.x, you can use the V3 to V4 Migration Tool to migrate both GUI and non-GUI code, but VAGen Import can only be used to migrate non-GUI code.
3. If you are migrating to VisualAge Generator 4.x on Smalltalk from VisualAge Generator 2.x, you can use the MSL Migration Assistance Tool or VAGen Import to migrate both GUI and non-GUI code.

## Automatic conversions using all migration tools

When you use any of the VisualAge Generator 4.x migration tools, some changes are made to your applications by the tools. These changes are needed to accommodate changes in the product. Without these conversions, you would not be able to use your existing applications with VisualAge Generator 4.x. Some conversions are only made if you are migrating from a specific release, while others are made regardless of your existing platform.

### Conversions for all existing applications

- Process and statement group parts are converted to function parts.
- References to processes and statement groups are converted to the new syntax requirements for functions with no parameters.

- PERFORM statements and unconditional branch statements are no longer supported and are migrated to function invocation statements.
- Subscript parentheses are changed to brackets in VisualAge Generator item names in the following places:
  - 4GL statements in functions (processes and statement groups)
  - Host variable names in SQL statements
  - Comparison value item in DL/I specifications
  - EZEDLPCB is used in a called parameter list

Calls to EZE service routines are converted to the corresponding function invocation statement. A statement to set the value of EZEREPLY is also added before the function invocation.

### Additional conversion only for VisualAge Generator 3.x applications

- VisualAge Generator-supplied string and math functions are converted from the CALL statement to a function invocation statement or to an assignment statement that contains the function as the source of the assignment. A statement to set the value of EZEREPLY is also added before the function invocation.

### Additional conversions only for VisualAge Generator 2.x and cross system product applications

- Members become VAGen parts.
- Applications become VAGen programs. VisualAge Generator 2.x GUIs become Smalltalk visual parts and are converted to the format needed for Smalltalk interoperability.
- For migration from releases earlier than CSP 3.3, additional syntax changes occur.

## Planning for migration to VisualAge Generator 4.x, continued

### V3 to V4 migration tool for 3.x migrations

The new V3 to V4 Migration Tool shipped with VisualAge Generator 4.x allows you to easily migrate 3.x applications and configuration maps to 4.x on Java or Smalltalk. You can set migration options for the V3 to V4 Migration Tool, which are applied when you later select applications and configuration maps for migration. For example, you can choose whether to maintain the existing ownership for all migrated applications and configuration maps, and you can set a naming convention for automatic versioning of the migrated code.

#### Special considerations for 3.x to Java

You can reduce your total migration time by considering these issues when you are planning for migrations to the Java platform:

- The basic containers for storing code are different on Java and Smalltalk. What the Smalltalk platform refers to as applications are called packages on Java. Similarly, *configuration maps* on Smalltalk are replaced by projects on Java. The V3 to V4 Migration Tool automatically converts *applications* to *packages* and *configuration maps* to *projects*. You can set an option for a naming convention to be used in naming these packages and projects, or you can specify the names yourself during the tool's execution.
- Java does not support a concept similar to subapplications. Therefore, the V3 to V4 Migration Tool enables you to set an option specifying whether to convert subapplications to separate packages or to merge subapplications (or migrated "subpackages") into the containing application (or migrated package).
- Although configuration maps are optional on Smalltalk, every Java package must be assigned to a project. However, you cannot assign

required projects on Java the way you can assign required maps to a configuration map. Instead, VisualAge Generator on Java introduces a new part, called the Project List Part, that enables you to specify all the projects that need to be loaded together in your workspace or generated together to produce a runtime application. Project List Parts are created automatically by the V3 to V4 Migration Tool when you migrate 3.x configuration maps that contain required maps to Java. (For 4.0, You will need to apply FixPak 2 for the automatic creation of Project List Parts to be available.)

- When applications containing GUI code are selected for migration, the GUI code is ignored and only the non-GUI code is migrated. After you recreate the GUI code on Java, you can merge the GUI parts back into the migrated package containing the non-GUI code. Alternatively, put the Java GUI code in different packages from your 4GL code.
- Java expects package names to be in all lowercase characters and to use "dot notation." You can set an option in the V3 to V4 Migration Tool so that the tool will automatically convert application names to lowercase and insert a period to the left of each character (except the first) that it changes from uppercase to lowercase. For example, if an existing application is named XYZpayrollAppl, the converted Java package is named x.y.zpayroll.appl. Some common acronyms are preserved.
- You cannot use the V3 to V4 Migration Tool to migrate an open edition of a 3.x application or configuration map. You must use VisualAge Generator 3.x to version and release each application and configuration map that you want to migrate to 4.x.

**Note:** Before you can start the V3 to V4 Migration Tool on Java, you must start VisualAge Generator using the

**VAGen Developer on Java with Migration option. This special option is not required on Smalltalk.**

#### Special considerations for 3.x to Smalltalk

Migrations from VisualAge Generator 3.x to 4.x on Smalltalk are mostly automated when you use the V3 to V4 Migration tool. However, there are a couple of issues to consider:

- You cannot use the V3 to V4 Migration Tool to migrate an open edition of a 3.x application or configuration map. You must use VisualAge Generator 3.x to version and release each application and configuration map that you want to migrate to 4.x.
- Before you can use the V3 to V4 Migration Tool to migrate 3.x code to 4.x on Smalltalk, you must import each versioned application or configuration map from your 3.x library into 4.x. Although the tool performs this step for you when you migrate to the Java platform, you must do it manually for Smalltalk migrations.

### MSL migration assistance tool for VisualAge Generator 2.x and Cross System Product migrations

The MSL Migration Assistance Tool provides a work area, or "sandbox," in which you can organize your VisualAge Generator 2.x or Cross System Product code into units usable by VisualAge Generator 4.x. When you are satisfied with the organization, you can then use the tool to commit your code to the 4.x repository.

**Note:** The MSL Migration Assistance Tool is sometimes called the MSL Migration Tool.

#### Special considerations for 2.x and Cross System Product to Java

You can reduce your total migration time by considering these issues when you are planning for migrations to the Java platform:



- The basic containers for storing code on Java are *projects* and *packages*. The MSL Migration Assistance Tool assists you in organizing your packages. When you commit a group of packages to the repository, the tool prompts you for the name of the project that will contain those packages.
- Java expects package names to be in all lowercase characters and to use “dot notation.”
- In VisualAge Generator 2.2, separate files were used to store control information such as linkage tables, resource association files, generation options, bind commands, and linkage editor control statements. In VisualAge generator 4.x, control information is stored in parts in the repository. The MSL Migration Assistance Tool does not create these parts. You must create them manually in VisualAge Generator 4.x.
- Performance is best when you limit a package to a maximum of 600-700 VAGen parts per VAGen part type. It is also recommended that you use no more than 50 classes per package. When considering this maximum, remember that processes and statement groups are both collapsed into the *VAGenFunctions* class.
- The MSL Migration Assistance Tool identifies duplicate parts. You need to resolve most duplicates by renaming one of the parts and changing all references to the renamed part. However, there are two situations in which you do **not** need to resolve the duplicate part names:
  1. Duplicates in an MSL concatenation that are a result of work-in-progress: For example, the same part might exist in both a test MSL and a production MSL. In this case, migrate the production MSL first and then migrate the test MSL to create new editions for the test-level parts.
  2. Duplicates that exist in MSLs that were never concatenated together (such as MSLs from unrelated subsystems): In this case, migrate one subsystem and then delete the code from your Java

workspace. Then migrate the second subsystem. You will be able to work with one migrated subsystem or the other, but not both at the same time.

- You will need to assign an owner for each VAGen class within a package. You will also need to assign an owner for each package and project. In planning your ownership strategy, you might want to group parts into packages that reflect how development and maintenance responsibilities are divided in your company. In this way, you can ensure that few people will need to work in the same package at the same time and access conflicts will be minimized.
- Early releases of Cross System Product allowed invalid data to be stored in the MSLs. You must correct the invalid data in all members before migrating to VisualAge Generator 4.x. Examples of invalid data include a data item that has a length of 0 and a map that contains a NUM field which is intended to store dates but is not long enough to store a date.

**Special considerations for 2.x and Cross System Product to Smalltalk**

Special considerations for migrations to the Smalltalk platform include:

- The basic containers for storing code in Smalltalk are *applications* and *configuration maps*. However, VisualAge Generator 2.x and Cross System Product applications are not the same as Smalltalk applications. Therefore, VisualAge Generator 2.x and Cross System Product applications are now called *programs*. The MSL Migration Assistance Tool assists you in organizing your MSL members into Smalltalk applications. After you commit the migrated applications into the library, you have the option of grouping the applications.
- VisualAge Generator 4.x on Smalltalk allows you to define a group of application editions that should be loaded together into your image. These groups are called *configuration maps*. For example, you

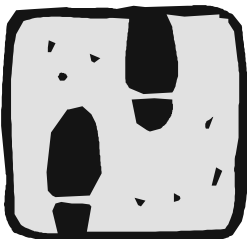
might create a configuration map that represents all the application editions that are currently used in your production environment. If you decide to use configuration maps, you should avoid listing the same application in multiple configuration maps, and you should consider establishing a naming convention for your configuration maps.

- In VisualAge Generator 2.2, separate files were used to store control information such as linkage tables, resource association files, generation options, bind commands, and linkage editor control statements. In VisualAge Generator 4.x, control information is stored in parts in the library. The MSL Migration Assistance Tool does not create these parts. You must create them manually in VisualAge Generator 4.x.
- Performance is best when you limit an application to a maximum of 600-700 VAGen parts per VAGen part type. It is also recommended that you use no more than 50 classes per application. When considering this maximum, remember that processes and statement groups are both collapsed into the *VAGenFunctions* class.
- The MSL Migration Assistance Tool identifies duplicate parts. You need to resolve most duplicates by renaming one of the parts and changing all references to the renamed part. However, there are two situations in which you do **not** need to resolve the duplicate part names:
  1. Duplicates in an MSL concatenation that are a result of work-in-progress: For example, the same part might exist in both a test MSL and a production MSL. In this case, migrate the production MSL first and then migrate the test MSL to create new editions for the test-level parts.
  2. Duplicates that exist in MSLs that were never concatenated together (such as MSLs from unrelated subsystems): In this case, migrate one subsystem and then delete the code from your Java workspace or Smalltalk image. Then migrate the second

## Planning for migration to VisualAge Generator 4.x, continued

subsystem. You will be able to work with one migrated subsystem or the other, but not both at the same time.

- You will need to assign an owner for each class within an application (including visual parts and VAGen part classes). You will also need to assign an owner for each application and configuration map. In planning your ownership strategy, you might want to group parts into applications that reflect how development and maintenance responsibilities are divided in your company. In this way, you can ensure that few people will need to work in the same application at the same time and access conflicts will be minimized.
- Early releases of Cross System Product allowed invalid data to be stored in the MSLs. You must correct the invalid data in all members before migrating to VisualAge Generator 4.x. Examples of invalid data include a data item that has a length of 0 and a map that contains a NUM field which is intended to store dates but is not long enough to store a date.
- VisualAge Generator 4.x on Smalltalk allows you to break an application into smaller components, called *subapplications*, for organizational purposes. You should not use subapplications if the code contained in them might need to be shipped separately or if you plan to later migrate the application to the Java environment.



- When GUIs are migrated to VisualAge Generator 4.x on Smalltalk, the GUIs are called *views* or *visual parts*. Each GUI becomes a class in VisualAge Smalltalk. Therefore, the GUIs are not visible in the VAGen Parts Browser or in the VAGen Parts pane shown on several windows. It is best to put the GUI code into different applications from your 4GL code.
- Both the MSL Migration Assistance Tool and VAGen Import write a list of important GUI changes to a file named **hptguicv.log**. The warning messages in this log indicate changes you must make to the view after migration.
- After GUIs are migrated into VisualAge Generator 4.x views, they cannot be exported back into VisualAge Generator 2.2 or earlier releases. Therefore, you should do all maintenance on the earlier release before you migrate the GUIs. Otherwise, you will need to maintain the source in both 4.x and the earlier release if you need to keep both versions current.
- GUI parts and features are renamed during conversion to VisualAge Generator 4.x views. See the *VisualAge Generator Migration Guide* for a list of name conversions.

### VAGen import

It is highly recommended that you use either the V3 to V4 Migration Tool or the MSL Migration Assistance Tool to migrate your code to VisualAge Generator 4.x. VAGen Import is intended only for importing single parts or small groups of parts that will be stored in the same Java package or Smalltalk application. Before you use VAGen Import for migration, consider these issues:

- You must use your existing release to export your code to external source format files before you can use VAGen Import to migrate the external source format files to VisualAge Generator 4.x. If you decide to use VAGen Import, you could organize your code by exporting one external source format file for each Java package or Smalltalk application that you want to create.
- Before you use VAGen Import, you will need to use VisualAge Generator 4.x to manually create Java projects and packages or Smalltalk applications to contain your migrated code.
- VAGen Import tool can only be used for GUIs coming from VisualAge Generator 2.2 going to VisualAge Generator 4.x on Smalltalk.

### More information

For more information on migrating to VisualAge Generator 4.x, see these books:

- *VisualAge Generator Migration Guide* (Version 4.0), SH23-0267-00
- *VisualAge Generator Version 4 System Development Guide*, SG24-5467-00
- *Migrating Cross System Product Applications to VisualAge Generator* (Version 3.1), SH23-0244-01

The IBM VisualAge Generator Consulting Services group can provide education, help you plan for migration, and even perform the entire migration for you. To contact a VisualAge Generator expert in the IBM VisualAge Generator Consulting Services group:

- Call (919) 254-2196
- Send e-mail to [vaconsul@us.ibm.com](mailto:vaconsul@us.ibm.com)
- Go to <http://www.ibm.com/software/ad/visgen/services> on the web. ■

# Client/Server configurations for VisualAge Generator version 4.0

by Kristine Heaton, VisualAge Generator Tester and John Snyder, VisualAge Generator Developer

As VisualAge Generator is enhanced to include new functions and additional platforms, the matrices and tables presented in this article can help you determine the many possible configurations. The following matrices and charts can assist in providing answers to questions, such as:

- Which environments are supported for clients?
- What are the supported server platforms?
- What protocols are supported and where are they used?
- What is valid for 2- and 3-tier configurations?
- What are the current options available for VisualAge Generator applications?

You can use the matrix and chart to determine valid configurations. For example, if you have a SmallTalk GUI and Windows NT is your client platform and you would like to implement a 3-tier configuration using a CICS for NT platform to connect to an IMS system, you would first use the Usage vs Platform

Matrix table to determine that you can execute a GUI on Windows NT. Then you can verify that you can call to CICS for NT using CICS Client using the Standard Client/Server From/To Protocol Matrix. From this same matrix (Standard Client/Server From/To Protocol Matrix) you can determine that CICS for NT cannot connect to an IMS system. By using the tables provided, you can determine that the desired configuration is not supported.

Here is another example. If you have a Java Application running on Windows NT and you want to access VM as a VisualAge Generator application server, you can verify that Java Applications are supported on Windows NT using the Usage vs. Platform Matrix. Then, using the Enterprise Java Bean chart you can verify that VM/ESA is a valid VisualAge Generator application server. Looking at the Standard Client/Server From/To Protocol Matrix you can then verify that Windows NT can access VM/ESA via TCP/IP. By using the tables provided, you can determine that this configuration is supported in VisualAge Generator.

With new releases of VisualAge Generator these matrices will change with additions and updates as new platforms and functions are added. For information on platforms and functions added in VisualAge Generator 4.5, see the VisualAge Generator website at <http://www.ibm.com/software/ad/visgen/>

Editor's Note: Would you be interested in a web based application that would make the job of planning runtime configurations easier? Let us know using the online reader comment form available on the VisualAge Generator website.

## GUI/TUI/Servlet Client/Server Chart

This following table shows the GUI, TUI, and Java Servlets that are supported in VisualAge Generator in a traditional client/server environment. It includes the 2nd-tier and 3rd-tier server configurations and the VAGen code (as HS, WGS,CSO) that is required at runtime. The supported communications protocols used for the various client and server platforms is referenced and specifically listed on the Standard Client/Server From/To Protocol Matrix (SCSFTPM) which follows.

Client	Communications	2nd Tier Server	Communications	3rd Tier Server
ST GUI (CSO)	Standard Client/Server From/To Protocol Matrix (SCSFTPM) (see below)	Windows NT (WGS)	Standard Client/Server From/To Protocol Matrix(SCSFTPM) (see below) Note: this does not imply that all clients can get to all servers. Must use the Usage vs. Platform Matrix to confirm what you want to do.	Windows NT (WGS)
Java Application (WGS/HS/CSO)*		OS/2 (WGS)		OS/2 (WGS)
Java Servlet (WGS/HS/CSO)*		AIX (WGS)		AIX (WGS)
C++ TUI (WGS)		Solaris (WGS)		Solaris (WGS)
COBOL TUI (WGS/HS)**		HP-UX (WGS)		HP-UX (WGS)
ITF (Developer)		VM/ESA (HS)		VM/ESA (HS)
		IMS (HS)		IMS (HS)
		AS/400 (HS)		AS/400 (HS)
		CICS for Windows NT (WGS)		CICS for Windows NT (WGS)
		CICS for OS/2 (WGS)		CICS for OS/2 (WGS)
		CICS for AIX (WGS)		CICS for AIX (WGS)
		CICS for Solaris (WGS)		CICS for Solaris (WGS)
		CICS for MVS/ESA (HS)		CICS for MVS/ESA (HS)
		CICS for VSE/ESA (HS)		CICS for VSE/ESA (HS)

**Chart Legend:** WGS = VisualAge Generator Server  
 HS = VisualAge Generator Host Services  
 CSO = VisualAge Generator Communications Services  
 ITF = VisualAge Generator Interactive Test Facility  
 Developer = VisualAge Generator Developer

**Platforms not allowed to be a client or 2nd tier of a 3 tier configuration:** (can't call out to someplace else) HP-UX, VM/ESA, IMS, Solaris, AS/400

## Client/Server configurations for VisualAge Generator version 4.0, continued

### Java Applet to Server Chart

The Java Applet to Server chart lists the current supported configurations for Java Applets to VisualAge Generator application servers.

Client	Communications	2nd Tier Server	Communications	VG Application Server
Web Browser - any platform	RMI	Windows NT (CSO*) OS/2 (CSO*) AIX (WGS)  * CSO code ships with developer and can be redistributed on the Windows and OS/2 platforms.	Standard Client/Server From/To Protocol Matrix  Note: this does not imply that all clients can get to all servers. Must use the Usage vs. Platform Matrix to confirm what you want to do.	Windows NT (WGS) OS/2 (WGS) AIX (WGS) Solaris (WGS) HP-UX (WGS) VM/ESA (HS) IMS (HS) AS/400 (HS) CICS for Windows NT (WGS) CICS for OS/2 (WGS) CICS for AIX (WGS) CICS for Solaris (WGS) CICS for MVS/ESA (HS) CICS for VSE/ESA (HS)
<p><b>Platforms where Web Server and VG Application Server can be the same physical machine:</b> Windows NT (Native C++ and CICS for Windows NT) OS/2 (Native C++ and CICS for OS/2) AIX (Native C++ and CICS for AIX)</p>				

**Chart Legend:** RMI = Remote Method Invocation

### Enterprise Java Beans Chart

This chart shows the supported EJB server environments and current supported VisualAge Application Server environments. Note that there is no matrix associated with the Enterprise Java Server (EJS) Communications. In the chart the EJB Servers listed are valid for all the listed clients.

Client	Communications	EJB Server	Communications	VG Application Server
Java Application (WGS/HS/CSO*) Java Servlet (WGS/HS/CSO*) Java Applet** (WGS/HS/CSO*)  * WGS for AIX, HS for OS/390 Unix System, and CSO for Windows and OS/2.  ** For this scenario, the "Client" is the Web Server where the EJB call is being made, not the browser running the Applet.  Note: We are not using the CSO code for the communications. CSO is for the VG Java runtime support.	Enterprise Java Server (EJS) Communications  Note: Actual protocol used varies depending on the EJS being used. WebSphere is currently using IIOP.	Windows NT (CSO*) OS/2 (CSO*) AIX (WGS)  * CSO code ships with developer and can be redistributed on the Windows and OS/2 platforms.	Standard Client/Server From/To Protocol Matrix  Note: this does not imply that all clients can get to all servers. Must use the Usage vs. Platform Matrix to confirm what you want to do.	Windows NT (WGS) OS/2 (WGS) AIX (WGS) Solaris (WGS) HP-UX (WGS) VM/ESA (HS) IMS (HS) AS/400 (HS) CICS for Windows NT (WGS) CICS for OS/2 (WGS) CICS for AIX (WGS) CICS for Solaris (WGS) CICS for MVS/ESA (HS) CICS for VSE/ESA (HS)
<p><b>Platforms where the EJB Server and the Server can be the same physical machine:</b> Windows NT (Native C++ and CICS for Windows NT) OS/2 (Native C++ and CICS for OS/2) AIX (Native C++ and CICS for AIX)</p>				

## WebSphere RAD (UI Record) Chart

The Websphere RAD (UI Record) table lists the currently-supported WebServer environments and runtime server environments for VisualAge Generator 4.0 and the supported protocols to access the runtime servers.

Client	Communications	EJB Server	Communications	VG Application Server
Web Browser - any platform	HTTP	Windows NT (WGS) OS/2 (WGS) AIX (WGS) Solaris (WGS) HP-UX (WGS)	TCP/IP to non-CICS platforms. CICS Transaction Gateway (CTG) to CICS platforms.	Windows NT (WGS) OS/2 (WGS) AIX (WGS) Solaris (WGS) HP-UX (WGS) IMS (HS) AS/400 (HS) CICS for Windows NT (WGS) CICS for AIX (WGS) CICS for Solaris (WGS) CICS for MVS/ESA (HS) CICS for VSE/ESA (HS)

### Platforms where Web Server and Server can be the same physical machine:

Windows NT (Windows NT or CICS for Windows NT)  
OS/2  
AIX (AIX or CICS for AIX)

## Usage vs. Platform Matrix

The Usage vs. Platform Matrix shows which platforms support a specific type of execution. For example, you would use this matrix to find the platforms which will support runtime execution of a SmallTalk GUI.

Platform Use	Windows 95	Windows 98	Windows NT	OS/2	AIX	Solaris	HP-UX	VM/ESA	IMS	AS/400	CICS for Windows NT	CICS for OS/2	CICS for Solaris	CICS for AIX	CICS for MVS/ESA	CICS for VSE/ESA	OS/390 Unix
ST GUI	X	X	X	X													
Java Application	X	X	X	X	X	X	X				X		X	X			
C++ TUI and Called Batch Application			X	X	X	X	X				X		X	X			
ITF			X	X													
COBOL TUI and Called Batch Application								X	X	X		X			X	X	
Java Servlet	X	X	X	X	X												
AppletWeb Server			X	X	X												
Enterprise Java Server			X	X	X												
UI Web Server			X	X	X	X	X										

**Client/Server configurations  
for VisualAge Generator version 4.0, continued**

**Standard Client/Server From/To Protocol Matrix**

The Standard Client/Server From/To Protocol Matrix (SCSFTPM) lists the supported server platforms, the supported client platforms, and the communications protocols used to connect to the server platforms.

If a platform is listed in the From (or client) column and also listed in the To (or server) column and it has a supported protocol at the intersection of the columns, then it can be a 2nd-tier server of a three-tier configuration. (A 2nd-tier server has the capability to act as a client and call out to another server).

Note that although HP-UX, VM/ESA, IMS, AS/400, OS/390, and Solaris are listed in the client column, they are not currently valid 2nd-tier platforms for a three-tier solution. They are listed under the client column of the Standard Client/Server From/To Protocol matrix and under the 2nd-Tier Server column of the GUI/TUI/Servlet Client/Server chart for possible future reference only.

**Standard Client/Server From/To Protocol Matrix  
(SCSFTPM)**

TO \ FROM	Windows NT	CICS for Windows NT	OS/2	CICS for OS/2	AIX	CICS for AIX	Solaris	CICS for Solaris	HP-UX	VM/ESA	IMS	AS/400	CICS for MVS/ESA	CICS for VSE/ESA
Windows 95	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP	CICS Client	TCP/IP	TCP/IP	APPC	CA/400	CICS Client	CICS Client
Windows 98	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP	CICS Client	TCP/IP	TCP/IP	APPC	CA/400	CICS Client	CICS Client
Windows NT	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP	CICS Client	TCP/IP	TCP/IP	APPC	CA/400	CICS Client	CICS Client
OS/2	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP	CICS Client	TCP/IP	TCP/IP	APPC	CA/400	CICS Client LU2	CICS Client
AIX	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP DCE	CICS Client	TCP/IP	CICS Client	TCP/IP	TCP/IP			CICS Client	CICS Client
Solaris														
HP-UX														
VM/ESA														
IMS														
AS/400														
CICS for Windows NT		CICS DPL		CICS DPL		CICS DPL		CICS DPL					CICS DPL	CICS DPL
CICS for OS2		CICS DPL		CICS DPL		CICS DPL		CICS DPL					CICS DPL	CICS DPL
CICS for AIX		CICS DPL		CICS DPL		CICS DPL		CICS DPL					CICS DPL	CICS DPL
CICS for Solaris		CICS DPL		CICS DPL		CICS DPL		CICS DPL					CICS DPL	CICS DPL
CICS for MVS/ESA		CICS DPL		CICS DPL		CICS DPL		CICS DPL					CICS DPL	CICS DPL
CICS for VSE/ESA		CICS DPL		CICS DPL		CICS DPL		CICS DPL					CICS DPL	CICS DPL
OS/390 Unix System														

# Accessing VSAM files on OS/390 from VisualAge Generator

by Chuck Proffer, VisualAge Generator Developer, Kristine Heaton, VisualAge Generator Tester and Mitch Johnson, VisualAge Generator Consulting Services

## I. Overview

VisualAge Generator provides the capability to access VSAM files that reside on an OS/390 system or any other remote system that supports the Distributed Data Management architecture. VSAM files on OS/390 systems are typically stored as EBCDIC data, although you can store ASCII data in them as well. VSAM files on the workstation are stored as ASCII data. If you use the OS/390 system only for data storage, you do not need to convert the data to EBCDIC. However, if you want to access the data from VisualAge Generator programs on your workstation and from programs on the host system, you need to store the data in EBCDIC. This means that your VisualAge Generator programs running on the workstation must convert the data between ASCII and EBCDIC. VisualAge Generator provides an automatic data conversion enhancement for remote VSAM files in V3.1 FixPak 3 and V4.0. The data will be automatically converted to EBCDIC before any write operations and converted to ASCII after any read operations.

## II. Software prerequisites

Before accessing remote VSAM files, there are several products that must be installed and configured. VisualAge Generator interfaces with Distributed File Manager (DFM) on the workstation and on the OS/390 to provide the remote VSAM access capability. DFM uses APPC for communications between the workstation and the OS/390.

IBM eNetwork Personal Communications Version 4.21 or later	Provides the APPC communications support on Windows NT and OS/2
IBM DFSMS/MVS Version 1.2 or later	Provides the DFM support on the OS/390

## III. Setup of Distributed File/Manager/MVS (DFM/MVS) on OS/390

Remote access to MVS/ESA and OS/390 VSAM data sets is provided by Distributed FileManager/MVS (DFM/MVS), a component of DFSMS/MVS. DFM/MVS communicates with remote clients using MVS/APPC, which is a part of the base control program (BCP) of each release of MVS/ESA and OS/390. For detailed information on DFM/MVS, refer to the DFM/MVS Guide and Reference - SC26-4915.

Using DFM/MVS for remote VSAM access requires the following:

- MVS/APPC must be active - DFM/MVS uses the MVS/APPC's base LU specified in member APPCPMxx in SYS1.PARMLIB.

```
LUADD ACBNAME(NRAPCVS1)
  BASE
  TPDATA(SYS1.APPCTP)
  TPLEVEL(SYSTEM)
  SIDEINFO DATASET(SYS1.APPCSI)
```

### SYS1.PARMLIB(APPCPMxx)

Start MVS/APPC with the MVS command:

```
START APPC, SUB=MSTR, APPC=xx
```

- An APPC/MVS transaction scheduler - The DFM/MVS transaction program (TP) is initiated when a remote request is received. The scheduling information is provided in member ASCHPMxx in SYS1.PARMLIB.

### CLASSADD CLASSNAME(DEFAULT)

```
MAX(20)
MIN(1)
MSGLIMIT(5000)
RESPGOAL(1)
```

### OPTIONS DEFAULT(DEFAULT)

### TPDEFAULT REGION(2M)

```
TIMES(5)
MSGLEVEL(1,1)
OUTCLASS(T)
```

### SYS1.PARMLIB(ASCHPMxx)

Start the MVS/APPC transaction scheduler with the following MVS command:

```
START ASCH, SUB=MSTR, ASCH=xx
```

```
//ATBSDFMU EXEC PGM=ATBSDFMU
//SYSPRINT DD SYSOUT=*
//SYSSDLIB DD DISP=SHR,DSN=SYS1.APPCTP
//SYSSDOUT DD SYSOUT=*
//SYSIN DD DATA,DLM=XX
TPADD TPNAME(^X'07'001) ACTIVE(YES)
TPSCHED_DELIMITER(##)
CLASS(DEFAULT)
JCL_DELIMITER(ENDJCL)
//GDEDFM JOB MSGCLASS=T,MSGLEVEL=(1,1),CLASS=A
//GDEDFM EXEC PGM=GDEISASB
ENDJCL
##
XX
```

### Adding a DFM/MVS TP Profile

A transaction program (TP) profile - A DFM/MVS agent is started when MVS/APPC receives a request. The TP profile is added to SYS1.APPCTP dataset with the ATBSDDMU utility.

Start the DFM task with the MVS command:

```
START DFM, SUB=MSTR
```

The following example shows the VTAM definition for the MVS/APPC LU name.

## Accessing VSAM files on OS/390 from VisualAge Generator, continued

```
*****
*MVS/APPC Application
*****

NRAPCVS1 APPL ACBNAME=NRAPCVS1,          C
  APPC=YES,                               C
  AUTOSSES=0,                              C
  DDRAINL=NALLOW,                          C
  DRESPL=NALLOW,                            C
  DSESLIM=32,                               C
  EAS=32,                                    C
  MODETAB=ISTINCLM,                          C
  SECACPT=CONV,                              C
  SRBEXIT=YES,                               C
  VERIFY=NONE
```

### IV. Setup on Windows NT

#### Install IBM Personal Communications

- Select the Communications APIs component
- Ensure that IBM LLC2 network protocol has been installed.

#### Configure an APPC session with the OS/390 host

To configure Personal Communications to support APPC connectivity, do the following:

1. Select the **start** button.
2. Select **Programs**.
3. Select **IBM Personal Communications**.
4. Select **SNA Node Configuration**, which opens the following window.



5. Select **Configure Node** in the **Configuration options** pane and then click **New...** to open the **Define the Node** window.



6. Define the host physical unit (PU) information for this workstation. In this example, the **Fully qualified CP name** should be USIBMNR.NRR50506. The **Block ID** is 05D and the **Physical Unit ID** is 50506 (Note: Physical Unit ID can be all zeros if IDNUM is not specified in the PU definition or if VTAM dynamic LU support is enabled). The **CP alias** is NRR50506 in this example but can be any value. This information was obtained from SYS1.VTAMLST(APPCPUS).
7. Click **OK** when finished.

Next we need to define a LAN Device.

1. In the **Configuration Options** pane, select **Configure Devices** and then click **New...** This opens the **Define a LAN Device** window.



2. Normally the default settings are okay. If the machine has multiple LAN cards then the **Adapter number** might have to be specified to correspond to the correct adapter.
3. Click **OK** when finished.



- Next we need to configure connections to the host. In the **Configuration options** pane, select **Configure Connections** and then click **New...** This opens the **Define a LAN Connection** window.



- On the **Basic** tab we need to specify the **Destination address**. This should be the LAN adapter address of the host (e.g. the TIC address of the communication controller). Also select the appropriate LAN type (Token-Ring or Ethernet).
- On the **Adjacent Node** tab, specify the **Adjacent CP Name**. This is obtained from SYS1.VTAMLST(ATCSTR00), which in this example is USIBMNR.NRMCMC2.



- Click **OK** when finished.
- Next we need to define the partner applications.

- Return to the **Configuration options** pane, select **Configure Partner LU 6.2**, and click **New...**



- On the **Define a Partner LU 6.2** window, type the fully qualified control point name (CP name) of the MVS/APPC subsystem. The fully qualified CP name is network ID plus the LU name of the MVS/APPC subsystem. The network ID is obtained from SYS1.VTAMLST(ATCSTR00) and the LU name is obtained from SYS1.PARMLIB(APPCPMxx). The **Partner LU alias** will be used as the TARGET\_SYSTEM in the dfm.rc configuration file and the **Fully qualified CP name** is obtained from SYS1.VTAMLST(ATCSTR00), which in this example is USIBMNR.NRMCMC2.
- Click **OK** when finished.

Finally, the workstation needs to know the LU name to use when connecting to the host applications.

- This LU name is specified by selecting **Configure Local LU 6.2**. The Local LU name is identified in SYS1.VTAMLST(APPCPUS) by the LU which has a LOCADDR equal to 0 (LOCADDR=0). This local LU name should also be specified via the environment variable APPCLLU. See the section entitled Sample VTAM Definitions for an example.



- Click **OK** when finished and save the configuration.
- If you installed Personal Communications in directory c:/PComm and saved the configuration as file config.acg, then the command **csstart -a /PComm/private/config.acg** can be used to load Personal Communications automatically when the system is started.

## Accessing VSAM files on OS/390 from VisualAge Generator, continued

### Configure DFM

The workstation support for DFM is provided by the SmartData Utilities component of IBM VisualAge for COBOL. If you have IBM VisualAge for COBOL installed, specifically the SmartData Utilities component, then you will have an `\ibmcobw\samples\hostdata` directory with sample configuration files. If you do not have IBM VisualAge for COBOL, you can download `vsamnt.zip` from <ftp://ps.software.ibm.com/ps/products/visualagegen/info/v4.0> and perform the following steps:

1. Create a directory called `vgdfm`
2. Copy the zip file to the `vgdfm` directory and unzip it using options `-o` and `-d`.

Modify the DFM configuration file (`dfm.rc`) from either the IBM COBOL directory or the `vgdfm` directory as follows:

1. There should be one `DFM_TARGET` statement for each server system that you plan to access.
2. Specify the LU alias of the server system in the `TARGET_SYSTEM` parameter as defined in the IBM Personal Communications configuration files, **Partner LU alias**.
3. Specify a `MODE_NAME` (LOG MODE) that is known on both the OS/390 and the workstation, such as `#INTER`.
4. Specify a `MAX_SEND_LIMIT` of 32767.

Modify the start DFM command (`strtsdu.cmd`) as follows:

1. Make sure that the configuration file name specified on the `dfmcfg` command is the same as the one previously modified (`dfm.rc`).
2. Replace the string `machine-name` on the `dfmlogon` command with the LU alias of the server system.
3. Replace the string `userid` on the `dfmlogon` command with the `userid` that will be used to access files on the server system.

If you do not have IBM VisualAge for COBOL installed, you will need to set the following environment variables:

```
CDRASRV=X:\VGDFM\CONVTABL
PATH=X:\VGDFM;%PATH%
```

### Start DFM

To start DFM, issue the `strtsdu` command from a command window. You will be prompted to enter the password for the `userid` specified on the `dfmlogon` command. To verify that everything is set up correctly, run the `dfmtry` command. It is shipped in the VisualAge Generator Server samples directory. Enter: `dfmtry machine-name userid`

It will prompt you for a password to the `userid` and then issue messages indicating whether or not it was able to successfully connect to the server system.

### Stopping DFM

To stop DFM, shutdown and restart your workstation. When you restart your workstation, do not start DFM.

Additional information on configuring, starting, and stopping DFM can be found in *Distributed FileManager User's Guide*, SC26-7134. There is also information in the online documentation that ships with IBM VisualAge for COBOL.

## V. Accessing VSAM files from ITF on Windows NT

Before you can access VSAM files from ITF, you must modify the VisualAge Generator Preferences. Select the **Options** menu on the **VisualAge Organizer** window. Select **Preferences** and the **VisualAge Preferences** notebook is displayed. Select the **VAGen - Test General** tab. At the bottom of the page, select the **Remote VSAM** radio button. This will cause ITF to use remote VSAM files for all file accesses (on Windows NT, there is only remote VSAM file support). In a later FixPak, this option will be moved to the Resource Association Editor so that the type of file accessed can be specified on a file basis. If you want the data converted to EBCDIC, a conversion table name must be specified in the **VSAM translation file name** field. The field should be preset with a default value based on your language setting. The list of valid conversion tables is documented in the VisualAge Generator Client/Server Communications Guide, Appendix B. When you have finished modifying the preferences, click the OK button. If you have changed your preferences to use **Remote VSAM** and you don't have APPC set up and working, you will receive error F207, which indicates a communications error.



In addition to changing your preferences, you also need to specify the physical name and path in the ITF Resource Association File editor. In the **Physical name** field, specify the file name as it is on your OS/390 system but without the high level qualifier. In the **Path** field, specify the machine name or a shortcut name using a technique similar to the Windows

Universal Naming Convention used for network file access. In the example below, the dataset name on the OS/390 is PROFFER.FIO1IR1.DATA. In the **Physical name** field, FIO1IR1.DATA is specified and \\CARMVS1\PROFFER is specified in the **Path** field. CARMVS1 is the machine name and PROFFER is the userid/high level qualifier.



If the file doesn't already exist on your OS/390 system, VisualAge Generator will create it for you the first time your program tries to add a record.

## VI. Accessing VSAM files from C++ generated programs on Windows NT

Access to VSAM files from a C++ generated program is determined by the resource association file (RSC). Specify / FILETYPE=VSAM in the ASSOCIATE entry for a VSAM file. Because there is no local VSAM support on Windows NT, all VSAM file access is remote. Specify the file name using a technique similar to the Windows Universal Naming Convention used for network file access. If you want the data converted to EBCDIC, specify the /CONTABLE option, otherwise the data will be written in ASCII. See the following example:

```
ASSOCIATE FILE=FIO1IR1 /
  SYSNAME=\\CARMVS1\PROFFER.FIO1IR1
  /FILETYPE=VSAM /CONTABLE=ELACNENU
```

If you do not have APPC set up and working, you will receive error F207, which indicates a communications error. Refer to the VisualAge Generator Server Guide for OS/2, Windows NT, HP-UX, and AIX for more information on using VSAM and resource association files.

## VII. Setup on OS/2 Warp

### Install IBM Personal Communications - OS/2 Access Feature

Ensure that IBM IEEE 802.2 protocol has been added to the LAN adapter

### Configure an APPC session with the OS/390 host

To configure OS/2 Access Feature for APPC connectivity, do the following:

1. Open the **Personal Communications** folder.
2. Select the **Access Feature Configuration** tool.
3. On the **Communications Manager Setup** window, select Setup and then open the configuration to added or updated, which opens the following window.



4. Select **APPC APIs over Token-ring** or **APPC APIs over Ethernet** and then click Configure to display the **APPC APIs over ...** window.



5. Define the the host physical unit (PU) information for this workstation. In this example, the **Network ID** is LSIBMNR and the **Local node name** is NRR50C16. This information was obtained from SYS1.VTAMLST(APPCPUS) and SYS1.VTAMLST(ATCSTR00). Select **Advanced...** to display the **Communication Manager Profile List** window.
6. From the **Communications Manager Profile List** window, SNA APPC profiles are configured.



## Accessing VSAM files on OS/390 from VisualAge Generator, continued

7. Select the **SNA local node characteristics** profile and then click **Configure...** to bring up the **Local Node Characteristics** window.
8. Confirm that the **Network ID** matches the NETID from SYS1.VTAMLST(ATCSTR00) and that the **Local Node name** (PU label) and the **Local node ID** (IDBLK and IDNUM) match the values from SYS1.VTAMLST(APPCPUS).

9. Click **OK** to return to the **Communications Manager Profile List** window. On this window select the **SNA Connections** profile, then **Configure...**

10. If there are already LU2 sessions defined, then a host connection might already exist. If this is true, then select this connection and then select **Change...** to continue, otherwise select **Create...**
11. Select the appropriate adapter to display the **Connections to a Host** window.

12. For the Partner LU definitions, specify the **Partner network ID** and **Partner node** name, obtained from

SYS1.VTAMLST(ATCSTR00), which in this example are USIBMNR and NRMCMC2. Specify the **LAN Destination address**, which should be the LAN adapter address of the host (e.g. the TIC address of the communication controller). Also select the appropriate Address format (Token-Ring or Ethernet). Next, click **Define Partner LUs...** to display the **Partner LUs** window.

13. On the **Partner LUs** window, type the **Network ID** and **LU name** of the MVS/APPC subsystem. The **Network ID** is obtained from SYS1.VTAMLST(ATCSTR00) and the **LU name** is obtained from SYS1.PARMLIB(APPCPMxx). The **Alias** will be used as the TARGET\_SYSTEM in the dfm.rc configuration file. When finished, **select ADD**.

14. Select **OK** or **Close** until the **Communication Manager Profile List** window is displayed. On this window select the **SNA features** profile and select **Configure...**

The workstation needs to know which LU name it should use when connecting to the host applications. This LU name is specified by creating a local LU profile by selecting the **Local LUs** feature and then clicking **Create...** The local LU is identified in SYS1.VTAMLST(APPCPUS) by the LU which has a LOCADDR equal to 0 (LOCADDR=0). This local LU name should also be specified via the environment variable, APPCLLU. See the section entitled *Sample VTAM Definitions* for an example.

15. Select **OK** and **Close** until OS/2 Access Features verifies the configuration. After the verification is complete, follow the instructions to activate the changes.
16. Open the Personal Communications folder and start the Access Feature by double-clicking on the icon.

### Configure DFM

The workstation support for DFM is provided by the SmartData Utilities component of IBM VisualAge for COBOL. If you have IBM VisualAge for COBOL installed, specifically the SmartData Utilities component, then you will have an `\ibmcobol\samples\sdu` directory with sample configuration files. If you do not have IBM VisualAge for COBOL, you can download `vsamos2.zip` from `ftp://ps.software.ibm.com/ps/products/visualagegen/info/v4.0` and perform the following steps:

1. Create a directory called `vgdfm`
2. Copy the zip file to the `vgdfm` directory and unzip it using options `-o` and `-d`.

Modify the DFM configuration file (`config.dfm`) from either the IBM COBOL `samples\sdu` directory or the `vgdfm` `samples` directory as follows:

1. Locate the `DFM_TARGET` statement. There should be one `DFM_TARGET` statement for each server system that you plan to access.
2. Specify the LU alias of the server system in the `REMOTE_LU` parameter as defined in the IBM Personal Communications configuration files, **Partner LU alias**.
3. Specify a `MAX_SEND_LIMIT` of 32767.
4. Specify the userid that will be used to access files on the server system in the `USERID` parameter.
5. Locate the `LOCAL_LU` statement and specify the LU name of the local system as defined in the IBM Personal Communications configuration files, **Local LU**.
6. Locate the `MODE_NAME` statement and specify a mode name (log mode) that is known on both the OS/390 and the workstation, such as **#INTER**.
7. Locate the `DEFAULT_DFM_TARGET` statement and specify the LU alias of the server system (same as in step 2 above).

Modify the start DFM command (`startdfm.cmd`) as follows:

1. Locate the `DFMDRIVE` statement and specify the drive letter of an unassigned drive and the LU alias of the target system you wish to access.

If you do not have IBM VisualAge for COBOL installed, you will need to set the following environment variables:

```
FMTCDRA=X:\VGDFM\CONVTABL
LIBPATH=X:\VGDFM;...
PATH=X:\VGDFM;...
```

### Starting DFM

After the configuration steps are complete, start DFM by running the `startdfm` command file. If everything started successfully, you should be able to issue the `dir` command with the drive letter specified on the `DFMDRIVE` statement and see a list of the files on your OS/390 system.

### Stopping DFM

There are two ways to stop DFM:

1. Shutdown and restart your workstation. When you restart your workstation, do not start DFM.
2. Run the command `DFMDRIVE RELEASE *` to release all of your DFM drive assignments.

Additional information on configuring, starting, and stopping DFM can be found in *Smartdata Utilities for OS/2: VSAM in a Distributed Environment*, SC26-7063. There is also information in the online documentation that ships with IBM VisualAge for COBOL.

### VIII. Accessing VSAM files from ITF on OS/2

Before you can access VSAM files from ITF, you must modify the VisualAge Generator Preferences. Select the Preferences from the **Options** menu on the **VisualAge Organizer** window to display the **VisualAge Preferences** pages. Select the **VAGen - Test** tab. At the bottom of the page, select either the **Local VSAM** or **Remote VSAM** radio button and click **OK** to have ITF use local or remote VSAM files for all file access. (In a later FixPak, this option will be moved to the Resource Association Editor so that the type of file accessed can be specified on a file basis.) If you want the data converted to EBCDIC, a conversion table name must be specified in the **VSAM translation file name** field. The field should be preset with a default value based on your language setting. The list of valid conversion tables is documented in the *VisualAge Generator Client/Server Communications Guide, Appendix B*. When you have finished modifying the preferences, select **OK**. If you have changed your preferences to use **Remote VSAM** and you don't have APPC set up and working, you will receive error F207, which indicates a communications error.

If you have selected **Local VSAM**, ITF will create and access VSAM files residing on your local drive. No translation is required, so the **VSAM translation file name** field is ignored and the data is written in ASCII.



## Accessing VSAM files on OS/390 from VisualAge Generator, continued

In addition to changing your preferences, you also need to specify the physical name and path in the ITF Resource Association File editor. In the **Physical name** field, specify the file name as it is on your OS/390 system but without the high level qualifier. In the **Path** field, specify the drive letter of the DFM drive. In the example below, the dataset name on the OS/390 is PROFFER.FIO1IR1. In the **Physical name field**, FIO1IR1 is specified and the drive letter M is specified in the **Path** field.



If the file doesn't already exist on your OS/390 system, VisualAge Generator will create it for you the first time your program tries to add a record.

### IX. Accessing VSAM files from C++ generated programs on OS/2

Access to VSAM files from a C++ generated program is determined by the resource association file (RSC). Specify / FILETYPE=VSAM in the ASSOCIATE entry for a VSAM file. To access a remote VSAM file, preface the file name with the DFM drive letter. See the following example:

```
ASSOCIATE FILE=FIO1IR1 /FILETYPE=VSAM /
SYSNAME=D:\PROFFER.FIO1IR1
/CONTABLE=ELACNENU
```

If you do not have APPC set up and working, you will receive error F207 indicating a communications error. Refer to the *VisualAge Generator Server Guide for OS/2, Windows NT, HP-UX, and AIX* for more information on using VSAM and resource association files.

### X. Diagnosing error conditions

#### Diagnosing error conditions when using VSAM with ITF

A trace facility has been provided to assist in diagnosing error conditions. The trace is controlled by the HPTTROPT environment variable. Specifying HPTTROPT=1 turns on the trace, specifying HPTTROPT=0 turns off the trace. The trace

output is written to a file named hptrace.out unless you change the name using the HPTTROUT environment variable.

#### Diagnosing error conditions when using VSAM with C++ generated programs

The trace facility for C++ generated programs is controlled by the FCWTROPT environment variable. Specifying FCWTROPT=31 will turn on trace for file I/O as well as other C++ program- related events. The trace output is written to a file named fcwtrace.out unless you change the name using the FCWTROUT environment variable. Refer to the appendix in the *VisualAge Generator Server Guide for OS/2, Windows NT, HP-UX, and AIX* for more information on the trace environment variables.

The I/O return codes found in both the HPT trace and the FCW trace files are VSAM reply messages. They are documented in VSAM in a Distributed Environment, SC26-7064. Some of the more common ones are listed below:

'1207'	Duplicate File Name
'1208'	Duplicate key, different index
'1209'	Duplicate key, same index
'120B'	End of file
'120C'	File is full
'120D'	File in use
'120E'	File not found
'1225'	Record not found
'1250'	Function not supported
'1251'	Parameter not supported
'125A'	File damaged
'125F'	Parameter not supported on target system
'F207'	Communications error

### XI. Sample VTAM definitions

#### SYS1.VTAMLST(ATCSTR00)...

```
SSCPNAME=NRMCMC2,
NETID=USIBMNR,
```

#### SYS1.VTAMLST(APPCPUS) - Sample Static PU Definition

```
NRR50506 PU      ADDR=04,           X
                  IDBLK=05D,         X
                  DBLK=05D,          X
                  IDNUM=50506,        X
                  DISCNT=NO,          X
                  IRETRY=NO,          X
                  ISTATUS=ACTIVE,     X
                  MAXDATA=1024,       X
                  MAXOUT=7,           X
                  PUTYPE=2,            X
                  PACING=0,            X
                  VPACING=0,           X
                  USSTAB=ALLUSS,       X
                  DLOGMOD=ISTINCLM,    X
                  SSCPFM=USSSCS,      X
                  MODETAB=HUBMODE
```

```
NR50506I LU LOCADDR=0,SSCPFM=FSS,USSTAB=
ISTINCDT,DLOGMOD=#INTER
```

# Testing modifications to generated UI Record JSPs

By Henry Koch, VisualAge Generator Architect

Author-Time Values is one of several VisualAge Generator features that help you move a Web transaction from prototype to production. This feature is of particular interest to Web designers, who can now customize a generated UI record JSP and review the Web page that results from that customization, all without deploying the generated program.

In addition to giving Web designers greater control of their work environment, Author-Time Values empowers project managers because it further separates the task of designing a user interface from the task of writing business logic. Web design can even reside at a location separate from code development. To display a web page at web transaction run time, the VisualAge Generator gateway servlet invokes a UI record JSP, which does the following:

- Produces an HTML stream that is based on the characteristics of a UI record.
- Embeds run-time values in the HTML by invoking methods in a UI record bean.

For example, in the following line from a UI record JSP, the value of field LASTNAME is placed within the HTML `<b>` structure, which directs the browser to display the string in boldface:

```
<b><%=LASTNAME.getTextValue()%></b>
```

Instead of relying on the gateway servlet, however, the web designer can invoke the UI record JSP directly if Author-Time Values is in effect. In this case, the data values in the resulting web page have initial values, and simulated error messages may be displayed.

To make Author-Time Values available, generate the web transaction or UI record with option GENAUTHORTIMEVALUES, which adds both a scriptlet to the UI record JSP and a method, `initAuthorTimeValues()`, to the UI record bean.

The scriptlet invokes `initAuthorTimeValues()` at run time, but the effect depends on how the UI record JSP itself is invoked: If the UI record JSP is invoked by the gateway servlet, the call to `initAuthorTimeValues()` has no effect, and data values are made available to the UI record JSP as usual. If the UI record JSP is invoked directly, however, `initAuthorTimeValues()` initializes the values that are to be returned to the UI record JSP. By default, the initial values reflect only the data item characteristics. For example, the value 99.99 is displayed for an item of UI type Output, data type Num, length 4, and decimals 2. By changing method `initAuthorTimeValues()` in a simple way, however, the web designer can replace the default values with a realistic set.

## Understanding the default initial values

By default, the initial values depend on the UI type and data type of the UI record items.

### UI type Input, Output, or Input/Output

Table 1 shows the default values for data items that are of UI type Input, Output, or Input/Output.

Data Type	Length	Decimals	Author Time Value
BIN, NUM, NUMC, PACK, PACF	N/A	0	A 9 for each integer digit; for example, 999999999 for a numeric item with a length of 9.
BIN, NUM, NUMC, PACK, PACF	N/A	>0	A 9 for each digit, with a decimal point as appropriate; for example, 999999.999 for a numeric data item with a length of 9 and 3 positions after the decimal point.
CHA	<=254	N/A	An A for each character; for example, AAA for a character item with a length of 3.
CHA	>254	N/A	A series of 254 A's.
MIX	<=254	N/A	An M for each character; for example, MMM for a mix item with a length of 3.
MIX	>254	N/A	A series of 254 M's.
DBCS	<=254	N/A	A double-byte D for each character; for example, DDD for a DBCS character with a length of 3.
DBCS	>254	N/A	A series of 254 double-byte D's.
Unicode	<=254	N/A	A U for each character; for example, UUU for a Unicode item with a length of 3.
Unicode	>254	N/A	A series of 254 U's.
HEX	<=254	N/A	An E for each character; for example, EEE for a hex item of length 3.
HEX	>254	N/A	A series of 254 E's.

Table 1. Default values

If a data item of UI type Input, Output, or Input/Output has an occurs value > 1 but has no OCCURS data item defined, a value is set for each occurrence of the data item. If an OCCURS data item is defined, the OCCURS data item value is set to two, and values are set for two occurrences of the data item.

## Testing modifications to generated UI Record JSPs, continued

If a data item of UI type Input, Output, or Input/Output has an occurs value > 1 and has a SELECTED data item defined, the SELECTED data item is set to a default value of 1 to cause the first occurrence to be selected when the page is displayed.

If data item edits are defined that could cause an error message to be displayed for a data item, a comment is added to the `initAuthorTimeValues()` method. The comment encloses Java scriptlet code. If the web designer removes the comment delimiters for a particular data item, the scriptlet causes an error message to be displayed for that data item, despite the fact that no error checking occurs when the UI record JSP is accessed directly.

For example, if data item QUANTITY (data type NUM) has a minimum value of 0 and a maximum value of 1000, the following comment lines are added to the `initAuthorTimeValues()` method:

```
/* Uncomment the following statement to simulate an error
   message being */
/* displayed for QUANTITY when displaying author time values */
/* getQuantity().setErrorMsg(0, "Value entered for data item
   QUANTITY is invalid"); */
```

When the web designer removes the `"/**` and `*/` delimiters of the last comment line, the remaining Java code causes the error message **Value entered for data item QUANTITY is invalid** to be displayed below data item QUANTITY. The web designer can change the method code to display a more specific message.

### UI type Submit or Submit Bypass

If a data item of UI type Submit or Submit Bypass has an initial value, the method `initAuthorTimeValues()` uses that value. The statement making the assignment has the following syntax:

```
<UIRecordName>.<dataItemName>.assign(<occursIndex>,
<initialValue>);
```

If a data item of UI type Submit or Submit Bypass has occurs > 1 and only one initial value is specified, that initial value applies to all occurrences of the data item.

If a data item of UI type Submit or Submit Bypass has no initial value but has a label, the method `initAuthorTimeValues()` uses the label as the initial value. (The web transaction must set the value of that data item to display the button at run time.)

The statement making the assignment is as follows:

```
<UIRecordName>.<dataItemName>.assign(<occursIndex>,
<label>);
```

If the data item of UI type Submit or Submit Bypass has no initial value and no label, the method `initAuthorTimeValues()` sets the data item's value to *No Label*. (The web transaction must set the value of that data item to display the button at run time.) The statement making the assignment is as follows:

```
<UIRecordName>.<dataItemName>.assign(<occursIndex>,
"No Label");
```

### UI type Program Link

If a data item of UI type Program Link has a label, no value is assigned to the data item, and the label is displayed as the value. If a data item of UI type Program Link has no label, the method `initAuthorTimeValues()` assigns a value *A Link* to the data item. The statement making the assignment is as follows:

```
<UIRecordName>.<dataItemName>.assign(<occursIndex>,
"A Link");
```

### Customizing the initial values

To allow review of a web page, web designers may want to change method `initAuthorTimeValues()` to display realistic values rather than values that reflect only the data types.

For example, a statement that assigns a value to UI record EMPLOYEE, data item LASTNAME might be as follows:

```
EMPLOYEE.LASTNAME.assign(0,
"AAAAAAAAAAAAAAAAAAAAA");
```

As shown, the statement includes the UI record and data item names as qualifiers for a method `assign()`. The first parameter of that method refers to an occurrence index, which starts at zero. The first parameter is also zero if the data item is not an array. The second parameter is the value to be displayed.

To display the last name Doe, change the statement as follows:

```
EMPLOYEE.LASTNAME.assign(0, "Doe");
```

If the LASTNAME data item is part of an array, the second occurrence could be changed as follows:

```
EMPLOYEE.LASTNAME.assign(1, "Smith");
```

If a data item of type Input, Output, or Input/Output has an occurs value > 1 and has an OCCURS data item defined, the OCCURS data item is set to a default value of 2 to cause two rows to be displayed. If you want to display more occurrences you need to change the value assigned to the OCCURS data item. For example, if the OCCURS data item for a data item in the EMPLOYEES UI record is NUMBEREMPLOYEES, and you want to display 10 occurrences of employees returned in a UI record, use the following statement:

```
EMPLOYEES.NUMBEREMPLOYEES.assign(0, 10L);
```

You would have to add assignment statements for the third through 10th occurrences of the data item as well.

If a data item of type Input, Output, or Input/Output has an occurs value > 1 and has a SELECTED data item defined, the SELECTED data item is set to a default value of 1 to cause the first occurrence to be selected when the page is displayed. The statement making the assignment is as follows:

```
<UIRecordName>.<selectedDataItemName>.assign(0, 1L);
```



You can modify the second parameter of this assign statement to have a different occurrence selected when the web page is displayed.

To simulate what a web page looks like when a web transaction does not display a SUBMIT button, comment out (enclose in /\* \*/ delimiters) the `initAuthorTimeValues()` code that assigns a value to the data item of UI type Submit or Submit Bypass. For example, change the following entry to the subsequent one:

```
EMPLOYEE.BUTTONS.assign(0, "EXIT");
/* EMPLOYEE.BUTTONS.assign(0, "EXIT"); */
```

It is important that the web designer affect only method `initAuthorTimeValues()` when changing the UI record bean. Otherwise, changes may interfere with run-time processing.

After making changes to `initAuthorTimeValues()`, the web designer must compile the UI record bean. This compilation occurs automatically if the web designer uses VisualAge for Java, which also provides a versioning mechanism by which the web designer can retain the original, generated UI record bean.

### Viewing customized JSPs

To view customized JSPs, do the following:

1. Deploy the UI record bean and the UI record object to the classpath of your web application server. The name of the UI record bean is of the form `<UI_record_name>Bean.class`. The name of the UI record object is of the form `VGUIr<UI_record_name>.class`.

If you are using VisualAge for Java's WebSphere Test Environment, fulfill this step by importing the UI record bean and UI record object into a VisualAge for Java project, making sure that the project is included in the VisualAge for Java Servlet Engine classpath. (When importing into VisualAge for Java, import the .java files for these classes.)

Chapter 4 of the VisualAge Generator *Web Transaction Development Guide (SH23-0281)* describes how to set up the VisualAge for Java WebSphere Test Environment. If you do not have edition 2 of that document, it is strongly recommended that you download the latest version from the following web site:

<http://www.ibm.com/software/ad/visgen/library/v45docs.html>

Deploy the enhanced JSP to the document root of your web server, which may be the WebSphere Test Environment's Servlet Engine.

2. In your browser, enter the URL, which is composed of the document root URL concatenated with the UI record JSP name. For example, if the URL of the web server's document root is `http://www.myCompany.com` and the JSP name is `myUIRecord.jsp`, enter the following:

<http://www.myCompany.com/myUIRecord.jsp>

If you are using the default web server configuration in the VisualAge for Java WebSphere Test Environment, enter the following:

<http://localhost:8080/myUIRecord.jsp>

Accessing a UI record JSP directly causes the UI record bean to be initialized with author time values, if any. Although you can view enhancements to a single JSP without running the gateway servlet or the web transaction, you cannot test links to other web pages in this circumstance.

### Example

Figure 1 and Figure 2 show UI records LISTRW-UI-Page and DETAIRW-UI-Page, respectively. These UI records are used in a web transaction that displays a list of employees, as well as detailed information about an employee selected from the list.

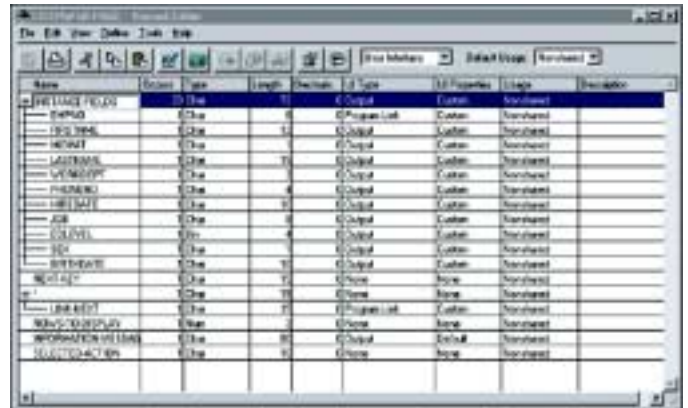


Figure 1. Employee list UI Record



Figure 2. Employee detail UI record

Figure 3 and Figure 4 show the JSPs generated for the LISTRW-UI-Page and DETAIRW-UI-Page UI records.

## Testing modifications to generated UI Record JSPs, continued

```
<% @ page import = "com.ibm.vgj.uibean.VGDataElement" %>
<HTML><HEAD>
<jsp:useBean id="LISTRWx002DUIx002DPAGE"scope="request" class="my.pkg.LISTRWx002DUIx002DPAGEBean"/>
<!--

                This is JAVA code that gets the individual data elements
                from the UI Bean that are to be used by this page to
                access all dynamic data. -->

<% LISTRWx002DUIx002DPAGE.initAuthorTimeValues(); %>
<%
    VGDataElement INSTANCEx002DFIELDS = LISTRWx002DUIx002DPAGE.getInstance();
    VGDataElement EMPNO = LISTRWx002DUIx002DPAGE.getEMPNO();
    VGDataElement FIRSNM = LISTRWx002DUIx002DPAGE.getFIRSNM();
    VGDataElement MIDINIT = LISTRWx002DUIx002DPAGE.getMIDINIT();
    VGDataElement LASTNAME = LISTRWx002DUIx002DPAGE.getLastname();
    VGDataElement WORKDEPT = LISTRWx002DUIx002DPAGE.getWORKDEPT();
    VGDataElement PHONENO = LISTRWx002DUIx002DPAGE.getPHONENO();
    VGDataElement HIREDATE = LISTRWx002DUIx002DPAGE.getHIREDATE();
    VGDataElement JOB = LISTRWx002DUIx002DPAGE.getJOB();
    VGDataElement EDLEVEL = LISTRWx002DUIx002DPAGE.getEDLEVEL();
    VGDataElement SEX = LISTRWx002DUIx002DPAGE.getSEX();
    VGDataElement BIRTHDATE = LISTRWx002DUIx002DPAGE.getBIRTHDATE();
    VGDataElement NEXTDKEY = LISTRWx002DUIx002DPAGE.getNextDKEY();
    VGDataElement LINKDNEXT = LISTRWx002DUIx002DPAGE.getLINKDNEXT();
    VGDataElement ROWSDTOx002DDISPLAY = LISTRWx002DUIx002DPAGE.getROWSx002DTox002DDisplay();
    VGDataElement INFORMATIONx002DMESSAGE = LISTRWx002DUIx002DPAGE.getINFORMATIONx002DMessage();
    VGDataElement SELECTEDx002DACTION = LISTRWx002DUIx002DPAGE.getSelectedx002DAction(); %>
<TITLE><%= LISTRWx002DUIx002DPAGE.getTitle() %></TITLE>
</HEAD>
<BODY><H1><%= LISTRWx002DUIx002DPAGE.getTitle() %></H1>
<TABLE BORDER="4" CELLPADDING="20">
  <TR>
    <TD>
      <TABLE ALIGN="left">
        <TR>
          <TD>
            <FORM METHOD="POST" ACTION="<%= LISTRWx002DUIx002DPAGE.getGatewayURL() %>"
              <!-- No comment defined for item INSTANCE-FIELDS -->
              <% if (INSTANCEx002DFIELDS.notEmpty()) { %>
                <TABLE BORDER="1">
                  <CAPTION ALIGN="top"><b><%= INSTANCEx002DFIELDS.getLabel() %> </b>&nbsp;</CAPTION>
                  <TR>
                    <% { %>
                      <%
                        java.util.Enumeration columns = INSTANCEx002DFIELDS.subElements();
                        while (columns.hasMoreElements()) {
                          VGDataElement column = (VGDataElement)columns.nextElement(); %>
                          <TH><%= column.getLabel() %></TH>
                        <% } %>
                      <% } %>
                    </TR>
                    <% { %>
                      <%
                        java.util.Enumeration rows = INSTANCEx002DFIELDS.occurrences();
                        while (rows.hasMoreElements()) {
                          VGDataElement row = (VGDataElement)rows.nextElement();
                          int i = row.getIndex(); %>
```

Figure 3. Generated employee list JSP

```

<TR>
  <TD ALIGN="left">
    <!-- No comment defined for item EMPNO -->
    <A HREF="<%= EMPNO.getGatewayURL(i) %>"><%= EMPNO.getTextValue(i) %></A></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item FIRSTNAME -->
    <%= FIRSTNAME.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item MIDINIT -->
    <%= MIDINIT.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item LASTNAME -->
    <%= LASTNAME.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item WORKDEPT -->
    <%= WORKDEPT.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item PHONENO -->
    <%= PHONENO.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item HIREDATE -->
    <%= HIREDATE.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item JOB -->
    <%= JOB.getTextValue(i) %></TD>
  <TD ALIGN="right">
    <!-- No comment defined for item EDLEVEL -->
    <%= EDLEVEL.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item SEX -->
    <%= SEX.getTextValue(i) %></TD>
  <TD ALIGN="left">
    <!-- No comment defined for item BIRTHDATE -->
    <%= BIRTHDATE.getTextValue(i) %></TD>
</TR>
<% } %>
<% } %>
</TABLE>
<% } %>
<br>
<!-- No comment defined for item LINK-NEXT -->
  <A HREF="<%= LINKx002DNEXT.getGatewayURL() %>"><b><%= LINKx002DNEXT.getLabel() %></b></A>
<br>
<!-- No comment defined for item INFORMATION-MESSAGE -->
  <b><%= INFORMATIONx002DMESSAGE.getLabel() %> </b><%= INFORMATIONx002DMESSAGE.getTextValue() %>
<br>
<P>
  <!-- VG Gateway control fields - DO NOT MODIFY -->
  <INPUT TYPE=HIDDEN NAME="hptAppId" VALUE="<%= LISTRWx002DUIx002DPAGE.getAppID()%>">
  <INPUT TYPE=HIDDEN NAME="hptSessionId" VALUE="<%= LISTRWx002DUIx002DPAGE.getSessionID() %>">
  <INPUT TYPE=HIDDEN NAME="hptPageId" VALUE="<%= LISTRWx002DUIx002DPAGE.getPageID()%>">
</FORM>
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY></HTML>

```

Figure 3. Generated employee list JSP, continued

## Testing modifications to generated UI Record JSPs, continued

```
<%@ page import = "com.ibm.vgj.uibean.VGDataElement" %>
<HTML><HEAD>
<jsp:useBean id="DETAIRWx002DUIx002DPAGE" scope="request" class="my.pkg.DETAIRWx002DUIx002DPAGEBean" />
<!--
        This is JAVA code that gets the individual data elements
        from the UI Bean that are to be used by this page to
        access all dynamic data. -->

<% DETAIRWx002DUIx002DPAGE.initAuthorTimeValues(); %>
<%
    VGDataElement EMPNO = DETAIRWx002DUIx002DPAGE.getEMPNO();
    VGDataElement FIRSNME = DETAIRWx002DUIx002DPAGE.getFIRSNME();
    VGDataElement MIDINIT = DETAIRWx002DUIx002DPAGE.getMIDINIT();
    VGDataElement LASTNAME = DETAIRWx002DUIx002DPAGE.getLastNAME();
    VGDataElement WORKDEPT = DETAIRWx002DUIx002DPAGE.getWORKDEPT();
    VGDataElement PHONENO = DETAIRWx002DUIx002DPAGE.getPHONENO();
    VGDataElement HIREDATE = DETAIRWx002DUIx002DPAGE.getHIREDATE();
    VGDataElement JOB = DETAIRWx002DUIx002DPAGE.getJOB();
    VGDataElement EDLEVEL = DETAIRWx002DUIx002DPAGE.getEDLEVEL();
    VGDataElement SEX = DETAIRWx002DUIx002DPAGE.getSEX();
    VGDataElement BIRTHDATE = DETAIRWx002DUIx002DPAGE.getBIRTHDATE();
    VGDataElement SALARY = DETAIRWx002DUIx002DPAGE.getSALARY();
    VGDataElement BONUS = DETAIRWx002DUIx002DPAGE.getBONUS();
    VGDataElement COMM = DETAIRWx002DUIx002DPAGE.getCOMM();
    VGDataElement INFORMATIONx002DMESSAGE = DETAIRWx002DUIx002DPAGE.getINFORMATIONx002DMESSAGE();
    VGDataElement SELECTEDx002DACTION = DETAIRWx002DUIx002DPAGE.getSELECTEDx002DACTION(); %>
<TITLE><%= DETAIRWx002DUIx002DPAGE.getTitle() %></TITLE>
</HEAD>
<BODY><H1><%= DETAIRWx002DUIx002DPAGE.getTitle() %></H1>
<TABLE BORDER="4" CELLPADDING="20">
  <TR>
    <TD>
      <TABLE ALIGN="left">
        <TR>
          <TD>
            <FORM METHOD="POST" ACTION="<%= DETAIRWx002DUIx002DPAGE.getGatewayURL() %>">
              <!-- No comment defined for item EMPNO -->
              <b><%= EMPNO.getLabel() %> </b><%= EMPNO.getTextValue() %>
              <br>
              <!-- No comment defined for item FIRSNME -->
              <b><%= FIRSNME.getLabel() %> </b><%= FIRSNME.getTextValue() %>
              <br>
              <!-- No comment defined for item MIDINIT -->
              <b><%= MIDINIT.getLabel() %> </b><%= MIDINIT.getTextValue() %>
              <br>
              <!-- No comment defined for item LASTNAME -->
              <b><%= LASTNAME.getLabel() %> </b><%= LASTNAME.getTextValue() %>
              <br>
              <!-- No comment defined for item WORKDEPT -->
              <b><%= WORKDEPT.getLabel() %> </b><%= WORKDEPT.getTextValue() %>
              <br>
              <!-- No comment defined for item PHONENO -->
              <b><%= PHONENO.getLabel() %> </b><%= PHONENO.getTextValue() %>
              <br>
              <!-- No comment defined for item HIREDATE -->
              <b><%= HIREDATE.getLabel() %> </b><%= HIREDATE.getTextValue() %>
              <br>
              <!-- No comment defined for item JOB -->
```

Figure 4. Generated employee detail JSP

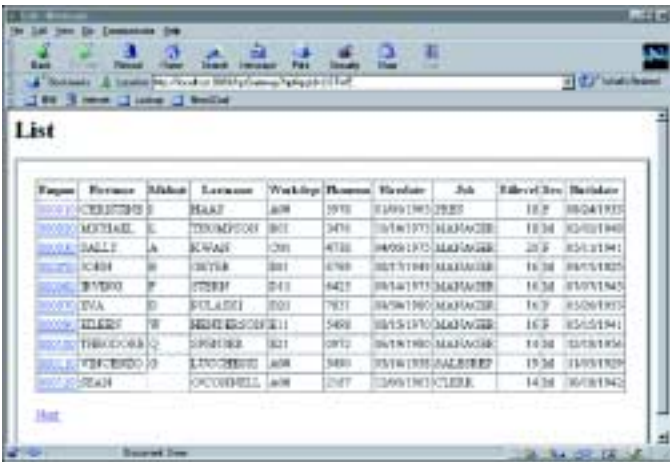
```

        <b><%= JOB.getLabel() %> </b><%= JOB.getTextValue() %>
<br>
<!-- No comment defined for item EDLEVEL -->
        <b><%= EDLEVEL.getLabel() %> </b><%= EDLEVEL.getTextValue() %>
<br>
<!-- No comment defined for item SEX -->
        <b><%= SEX.getLabel() %> </b><%= SEX.getTextValue() %>
<br>
<!-- No comment defined for item BIRTHDATE -->
        <b><%= BIRTHDATE.getLabel() %> </b><%= BIRTHDATE.getTextValue() %>
<br>
<!-- No comment defined for item SALARY -->
        <b><%= SALARY.getLabel() %> </b><%= SALARY.getTextValue() %>
<br>
<!-- No comment defined for item BONUS -->
        <b><%= BONUS.getLabel() %> </b><%= BONUS.getTextValue() %>
<br>
<!-- No comment defined for item COMM -->
        <b><%= COMM.getLabel() %> </b><%= COMM.getTextValue() %>
<br>
<!-- No comment defined for item INFORMATION-MESSAGE -->
<% if (INFORMATIONx002DMESSAGE.notEmpty()) { %>
        <b><%= INFORMATIONx002DMESSAGE.getLabel() %> </b>
<%
        out.println("<pre>");
        java.util.Enumeration lines = INFORMATIONx002DMESSAGE.occurrences();
        while (lines.hasMoreElements()) {
            VGDataElement line = (VGDataElement)lines.nextElement();
            out.println(line.getTextValue());
        }
        out.println("</pre>"); %>
<% } %>
<br>
<P>
<!-- VG Gateway control fields - DO NOT MODIFY -->
        <INPUT TYPE=HIDDEN NAME="hptAppId" VALUE="<%= DETAIRWx002DUIx002DPAGE.getAppID()%>">
        <INPUT TYPE=HIDDEN NAME="hptSessionId" VALUE="<%= DETAIRWx002DUIx002DPAGE.getSessionID() %>">
        <INPUT TYPE=HIDDEN NAME="hptPageId" VALUE="<%= DETAIRWx002DUIx002DPAGE.getPageID()%>">
</FORM>
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY></HTML>

```

**Figure 4. Generated employee detail JSP, continued**

## Testing modifications to generated UI Record JSPs, continued



The screenshot shows a web browser window with a table titled "List". The table has the following columns: Empno, Firtname, Midinit, Lastname, Workdept, Phoneno, Hiredate, Job, Edlevel, Sex, Birthdate. The data rows are as follows:

Empno	Firtname	Midinit	Lastname	Workdept	Phoneno	Hiredate	Job	Edlevel	Sex	Birthdate
00001	CHRISTINE	I	HAAS	A00	3978	01/01/1965	PRES	18	F	08/24/1933
00002	MORTIMER	E	DOUMPSOON	001	3478	03/16/1973	MANAGER	18.38	M	03/01/1940
00003	MALLE	A	SCHWAB	001	4788	04/09/1970	MANAGER	20.5	F	05/21/1941
00004	ICAHN	B	CHYSE	001	4788	05/21/1941	MANAGER	18.38	M	04/25/1925
00005	BYRNO	F	STREIF	010	4421	08/14/1973	MANAGER	18.38	F	01/03/1945
00006	EVA	D	STLADEI	001	7821	04/06/1960	MANAGER	16.3	F	05/26/1913
00007	HELENE	B	ROBERTSON	011	5468	04/15/1970	MANAGER	16.3	F	05/05/1941
00008	THROCKM	<	SPRIBER	001	3972	04/16/1960	MANAGER	18.38	M	03/01/1956
00009	VERFRENDO	D	LANOCHES	A00	3480	03/16/1973	MANAGER	19.38	M	11/03/1929
00010	DEAN		OCCORHELL	A00	2987	03/09/1973	CLERK	14.38	M	06/08/1942

Figure 5. Employee list as displayed by the generated JSP

Figure 5 and Figure 6 show the output when running web transactions that display the employee list and employee detail. The generated UI record JSPs are in use...



The screenshot shows a web browser window with a "Detail" page. The details for Empno 000010 are as follows:

**Empno** 000010  
**Firtname** CHRISTINE  
**Midinit** I  
**Lastname** HAAS  
**Workdept** A00  
**Phoneno** 3978  
**Hiredate** 01/01/1965  
**Job** PRES  
**Edlevel** 18  
**Sex** F  
**Birthdate** 08/24/1933  
**Salary** \$52750.00  
**Bonus** \$1000.00  
**Comm** \$4220.00

Figure 6. Employee detail as displayed by the generated JSP

Figure 7 shows the employee list JSP enhanced to use alternating light and dark background colors for the rows of the employee list. A program link was also added to return to the gateway servlet entry page. Changes are highlighted.

```
<%@ page import = "com.ibm.vgj.uibean.VGDataElement" %>
<HTML><HEAD>
<jsp:useBean id="LISTRWx002DUIx002DPAGE" scope="request" class="my.pkg.LISTRWx002DUIx002DPAGEBean" />
<!--
                This is JAVA code that gets the individual data elements
                from the UI Bean that are to be used by this page to
                access all dynamic data. -->
<% LISTRWx002DUIx002DPAGE.initAuthorTimeValues(); %>
<%
    VGDataElement INSTANCEx002DFIELDS = LISTRWx002DUIx002DPAGE.getInstancex002DFIELDS();
    VGDataElement EMPNO = LISTRWx002DUIx002DPAGE.getEMPNO();
    VGDataElement FIRSNME = LISTRWx002DUIx002DPAGE.getFIRSNME();
    VGDataElement MIDINIT = LISTRWx002DUIx002DPAGE.getMIDINIT();
    VGDataElement LASTNAME = LISTRWx002DUIx002DPAGE.getLastname();
    VGDataElement WORKDEPT = LISTRWx002DUIx002DPAGE.getWORKDEPT();
    VGDataElement PHONENO = LISTRWx002DUIx002DPAGE.getPHONENO();
    VGDataElement HIREDATE = LISTRWx002DUIx002DPAGE.getHIREDATE();
    VGDataElement JOB = LISTRWx002DUIx002DPAGE.getJOB();
    VGDataElement EDLEVEL = LISTRWx002DUIx002DPAGE.getEDLEVEL();
    VGDataElement SEX = LISTRWx002DUIx002DPAGE.getSEX();
    VGDataElement BIRTHDATE = LISTRWx002DUIx002DPAGE.getBIRTHDATE();
    VGDataElement NEXTx002DKEY = LISTRWx002DUIx002DPAGE.getNextx002DKEY();
    VGDataElement LINKx002DNEXT = LISTRWx002DUIx002DPAGE.getLINKx002DNEXT();
    VGDataElement ROWSx002DTox002DDISPLAY = LISTRWx002DUIx002DPAGE.getROWSx002DTox002DDISPLAY();
    VGDataElement INFORMATIONx002DMESSAGE = LISTRWx002DUIx002DPAGE.getINFORMATIONx002DMESSAGE();
    VGDataElement SELECTEDx002DACTION = LISTRWx002DUIx002DPAGE.getSelectedx002DACTION(); %>
<TITLE><%= LISTRWx002DUIx002DPAGE.getTitle() %></TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide script
```

Figure 7. Enhanced employee list JSP

```

var ph = null;
var dirwin = null;
function Popup()
{
    ph = window.open("", 'DefaultPopup', 'toolbar=no,location=no,directories=no,status=no,
        scrollbars=yes,resizable=yes,copyhistory=no,width=350,height=325');
    ph.document.write("<p><br><p><br><ul><font color='blue'><b>Loading data, please wait ...</b></font></ul>");
    if (navigator.appName == "Netscape")
        {
            ph.focus();
        }
}
// End script hiding —>
</SCRIPT>
</HEAD>
<BODY><H1><%= LISTRWx002DUIx002DPAGE.getTitle() %></H1>
<!-- No comment defined for item INSTANCE-FIELDS —>

<% if (INSTANCEx002DFIELDS.notEmpty()) { %>

    <TABLE border=0>
    <CAPTION ALIGN="top"><b><b><%= INSTANCEx002DFIELDS.getLabel() %> </b></b></CAPTION>
    <TR>
    <% { %>
        <%
            java.util.Enumeration columns = INSTANCEx002DFIELDS.subElements();
            while (columns.hasMoreElements()) {
                VGDataElement column = (VGDataElement)columns.nextElement(); %>
                <TH><%= column.getLabel() %></TH>
            <% } %>
        <% } %>
    </TR>

        <% { %>
            <%
                java.util.Enumeration rows = INSTANCEx002DFIELDS.occurrences();
                while (rows.hasMoreElements()) {
                    VGDataElement row = (VGDataElement)rows.nextElement();
                    int i = row.getIndex(); %>
                    <% if (!%2 == 1) { %>
                        <TR bgcolor=#4f60af>
                    <% } else { %>
                        <TR bgcolor=#aaaaaa>
                    <% } %>

                    <TD ALIGN="left">
                        <!-- No comment defined for item EMPNO —>
                        <A HREF="<%= EMPNO.getGatewayURL(i) %>" onClick="Popup()"
                            target="DefaultPopup" ><FONT COLOR="white">
                            <%= EMPNO.getTextValue(i) %></A></TD>

                    <TD ALIGN="left">
                        <!-- No comment defined for item FIRSTNME —>
                        <%= FIRSTNME.getTextValue(i) %></TD>
                    <TD ALIGN="left">
                        <!-- No comment defined for item MIDINIT —>
                        <%= MIDINIT.getTextValue(i) %></TD>
                    <TD ALIGN="left">
                        <!-- No comment defined for item LASTNAME —>
                        <%= LASTNAME.getTextValue(i) %></TD>
                    <TD ALIGN="left">

```

Figure 7. Enhanced employee list JSP, continued

## Testing modifications to generated UI Record JSPs, continued

```
        <%= HIREDATE.getTextValue(i) %></TD>
<TD ALIGN="left">
        <!-- No comment defined for item JOB -->
        <%= JOB.getTextValue(i) %></TD>
<TD ALIGN="right">
        <!-- No comment defined for item EDLEVEL -->
        <%= EDLEVEL.getTextValue(i) %></TD>
<TD ALIGN="left">
        <!-- No comment defined for item SEX -->
        <%= SEX.getTextValue(i) %></TD>
<TD ALIGN="left">
        <!-- No comment defined for item BIRTHDATE -->
        <%= BIRTHDATE.getTextValue(i) %></TD>
        </TR>
    <% } %>
</TABLE>
<% } %>
<br>
<!-- No comment defined for item LINK-NEXT -->
<A HREF="<%= LINKx002DNEXT.getGatewayURL() %>"><b><%= LINKx002DNEXT.getLabel() %> </b></A><br>
<A HREF="<%= LISTRWx002DUIx002DPAGE.getGatewayURL() %>" target="_top"><b>Home</b></A>
<br>

<!-- No comment defined for item INFORMATION-MESSAGE -->
<b><%= INFORMATIONx002DMESSAGE.getLabel() %> </b><%= INFORMATIONx002DMESSAGE.getTextValue() %>
<br>
<P>
</BODY></HTML>
```

**Figure 7. Enhanced employee list JSP, continued**

Figure 8 shows an enhanced JSP for the employee detail. The highlighted changes cause the following:

- Aligned labels and attribute values in columns

```
<%@ page import = "com.ibm.vgj.uibean.VGDataElement" %>
<HTML><HEAD>
<jsp:useBean id="DETAIRWx002DUIx002DPAGE" scope="request" class="my.pkg.DETAIRWx002DUIx002DPAGEBean" />
<!--
        This is JAVA code that gets the individual data elements
        from the UI Bean that are to be used by this page to
        access all dynamic data. -->
<% DETAIRWx002DUIx002DPAGE.initAuthorTimeValues(); %>
<%
    VGDataElement EMPNO = DETAIRWx002DUIx002DPAGE.getEMPNO();
    VGDataElement FIRSNMNE = DETAIRWx002DUIx002DPAGE.getFIRSNMNE();
    VGDataElement MIDINIT = DETAIRWx002DUIx002DPAGE.getMIDINIT();
    VGDataElement LASTNAME = DETAIRWx002DUIx002DPAGE.getLastname();
    VGDataElement WORKDEPT = DETAIRWx002DUIx002DPAGE.getWORKDEPT();
    VGDataElement PHONENO = DETAIRWx002DUIx002DPAGE.getPHONENO();
    VGDataElement HIREDATE = DETAIRWx002DUIx002DPAGE.getHIREDATE();
    VGDataElement JOB = DETAIRWx002DUIx002DPAGE.getJOB();
    VGDataElement EDLEVEL = DETAIRWx002DUIx002DPAGE.getEDLEVEL();
```

**Figure 8. Enhanced Employee Detail JSP**



```

VGDataElement SEX = DETAIRWx002DUIx002DPAGE.getSEX();
VGDataElement BIRTHDATE = DETAIRWx002DUIx002DPAGE.getBIRTHDATE();
VGDataElement SALARY = DETAIRWx002DUIx002DPAGE.getSALARY();
VGDataElement BONUS = DETAIRWx002DUIx002DPAGE.getBONUS();
VGDataElement COMM = DETAIRWx002DUIx002DPAGE.getCOMM();
VGDataElement INFORMATIONx002DMESSAGE = DETAIRWx002DUIx002DPAGE.getINFORMATIONx002DMESSAGE();
VGDataElement SELECTEDx002DACTION = DETAIRWx002DUIx002DPAGE.getSelectedx002DACTION(); %>
<TITLE><%= DETAIRWx002DUIx002DPAGE.getTitle() %></TITLE>
</HEAD>
<BODY>
<b><%= LASTNAME.getTextValue() %>, <%= FIRSTNAME.getTextValue() %></b><br>
<TABLE BORDER=0 bgcolor=#aaaaaa WIDTH=300><TR><TD bgcolor=#4f60af colspan=2 align=center><FONT
color="white"><B><%= DETAIRWx002DUIx002DPAGE.getTitle() %></B></FONT></TD>
<TR>
<!-- No comment defined for item EMPNO -->
<TD><b><%= EMPNO.getLabel() %> </b></TD><TD><%= EMPNO.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item WORKDEPT -->
<TD><b><%= WORKDEPT.getLabel() %> </b></TD><TD><%= WORKDEPT.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item PHONENO -->
<TD><b><%= PHONENO.getLabel() %> </b></TD><TD><%= PHONENO.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item HIREDATE -->
<TD><b><%= HIREDATE.getLabel() %> </b></TD><TD><%= HIREDATE.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item JOB -->
<TD><b><%= JOB.getLabel() %> </b></TD><TD><%= JOB.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item EDLEVEL -->
<TD><b><%= EDLEVEL.getLabel() %> </b></TD><TD><%= EDLEVEL.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item SEX -->
<TD><b><%= SEX.getLabel() %> </b></TD><TD><%= SEX.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item BIRTHDATE -->
<TD><b><%= BIRTHDATE.getLabel() %> </b></TD><TD><%= BIRTHDATE.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item SALARY -->
<TD><b><%= SALARY.getLabel() %> </b></TD><TD><%= SALARY.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item BONUS -->
<TD><b><%= BONUS.getLabel() %> </b></TD><TD><%= BONUS.getTextValue() %></TD>
</TR>
<TR>
<!-- No comment defined for item COMM -->
<TD><b><%= COMM.getLabel() %> </b></TD><TD><%= COMM.getTextValue() %></TD>
</TR>
</TABLE>
</BODY></HTML>

```

Figure 8. Enhanced Employee Detail JSP, continued

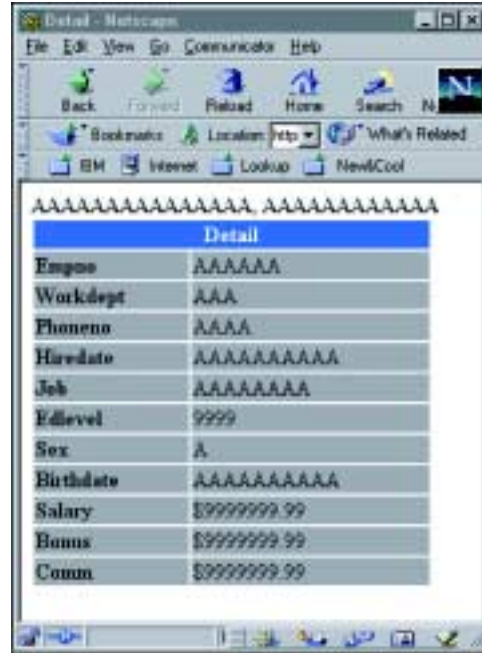
## Testing modifications to generated UI Record JSPs, continued

Figure 9 shows the employee list, which is presented by the enhanced JSP and includes author-time values that were set by the generated method `initAuthorTimeValues()`.



**Figure 9. Enhanced employee list JSP with generated author-time values**

Figure 10 shows the employee detail, which is presented by the enhanced JSP and includes author-time values that were set by the generated method `initAuthorTimeValues()`.



**Figure 10. Enhanced employee detail JSP with generated author-time values**

Figure 11 shows the generated method `initAuthorTimeValues()` for the employee list UI record. This version of the method was used to set the initial values displayed in Figure 9.

```
// Set author time initial values for data items associated with HTML elements.
// The initial values are based only on UI Type and data item of the items.
public void initAuthorTimeValues()
{
    if ( !getGatewayURL().equals( NOGATEWAYURL ))
        return;

    try
    {
        LISTRWx002DUIx002DPAGE.setInitialValues();
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
        LISTRWx002DUIx002DPAGE.EMPNO.assign( 0, "A Link" );
        LISTRWx002DUIx002DPAGE.EMPNO.assign( 1, "A Link" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
        LISTRWx002DUIx002DPAGE.FIRSTNME.assign( 0, "AAAAAAAAAAAA" );
        LISTRWx002DUIx002DPAGE.FIRSTNME.assign( 1, "AAAAAAAAAAAA" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
        LISTRWx002DUIx002DPAGE.MIDINIT.assign( 0, "A" );
        LISTRWx002DUIx002DPAGE.MIDINIT.assign( 1, "A" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
        LISTRWx002DUIx002DPAGE.LASTNAME.assign( 0, "AAAAAAAAAAAAAAA" );
        LISTRWx002DUIx002DPAGE.LASTNAME.assign( 1, "AAAAAAAAAAAAAAA" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
        LISTRWx002DUIx002DPAGE.WORKDEPT.assign( 0, "AAA" );
        LISTRWx002DUIx002DPAGE.WORKDEPT.assign( 1, "AAA" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
        LISTRWx002DUIx002DPAGE.PHONENO.assign( 0, "AAAA" );
        LISTRWx002DUIx002DPAGE.PHONENO.assign( 1, "AAAA" );
    }
}
```

**Figure 11. Generated method `initAuthorTimeValues()` for the Employee List bean**

```

LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
LISTRWx002DUIx002DPAGE.HIREDATE.assign( 0, "AAAAAAAAA" );
LISTRWx002DUIx002DPAGE.HIREDATE.assign( 1, "AAAAAAAAA" );
LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
LISTRWx002DUIx002DPAGE.JOB.assign( 0, "AAAAAAAA" );
LISTRWx002DUIx002DPAGE.JOB.assign( 1, "AAAAAAAA" );
LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
LISTRWx002DUIx002DPAGE.EDLEVEL.assign( 0, 9999L );
LISTRWx002DUIx002DPAGE.EDLEVEL.assign( 1, 9999L );
/* Uncomment the following statement to simulate an error message being */
/* displayed for EDLEVEL when displaying author time values. */
/* getEDLEVEL().setErrorMsg( 0, "Value entered for data item EDLEVEL is invalid" ); */
LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
LISTRWx002DUIx002DPAGE.SEX.assign( 0, "A" );
LISTRWx002DUIx002DPAGE.SEX.assign( 1, "A" );
LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 2L );
LISTRWx002DUIx002DPAGE.BIRTHDATE.assign( 0, "AAAAAAAAA" );
LISTRWx002DUIx002DPAGE.BIRTHDATE.assign( 1, "AAAAAAAAA" );
LISTRWx002DUIx002DPAGE.LINKx002DNEXT.assign( 0, "A Link" );
LISTRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 0,
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
    }
    catch ( Exception e )
    {
        System.out.println( "Exception received trying to initialize author time values: "
+ e.getMessage() );
    }
}

```

**Figure 11. Generated method `initAuthorTimeValues()` for the *Employee List* bean, continued**

Figure 12 shows the generated method `initAuthorTimeValues()` for the employee detail UI record. This version of the method was used to set the initial values displayed in Figure 10.

```

// Set author time initial values for data items associated with HTML elements.
// The initial values are based only on UI Type and data item of the items.
public void initAuthorTimeValues()
{
    if ( !getGatewayURL().equals( NOGATEWAYURL ) )
        return;

    try
    {
        DETAIRWx002DUIx002DPAGE.setInitialValues();
        DETAIRWx002DUIx002DPAGE.EMPNO.assign( 0, "AAAAAA" );
        DETAIRWx002DUIx002DPAGE.FIRSTNME.assign( 0, "AAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.MIDINIT.assign( 0, "A" );
        DETAIRWx002DUIx002DPAGE.LASTNAME.assign( 0, "AAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.WORKDEPT.assign( 0, "AAA" );
        DETAIRWx002DUIx002DPAGE.PHONENO.assign( 0, "AAAA" );
        DETAIRWx002DUIx002DPAGE.HIREDATE.assign( 0, "AAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.JOB.assign( 0, "AAAAAA" );
        DETAIRWx002DUIx002DPAGE.EDLEVEL.assign( 0, 9999L );
        /* Uncomment the following statement to simulate an error message being */
        /* displayed for EDLEVEL when displaying author time values. */
        /* getEDLEVEL().setErrorMsg( 0, "Value entered for data item EDLEVEL is invalid" ); */
        DETAIRWx002DUIx002DPAGE.SEX.assign( 0, "A" );
        DETAIRWx002DUIx002DPAGE.BIRTHDATE.assign( 0, "AAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.SALARY.assign( 0, new VGJBigNumber( "9999999.99" ) );
        /* Uncomment the following statement to simulate an error message being */
        /* displayed for SALARY when displaying author time values. */
        /* getSALARY().setErrorMsg( 0, "Value entered for data item SALARY is invalid" ); */
    }
}

```

**Figure 12. Generated method `initAuthorTimeValues()` for the *Employee Detail* bean**



Figure 15 shows an enhanced method `initAuthorTimeValues()` for the employee list UI record. Changes are highlighted. This version of the method was used to set the initial values displayed in Figure 13.

```
// Set author time initial values for data items associated with HTML elements.
// The initial values are based only on UI Type and data item of the items.
public void initAuthorTimeValues()
{
    if ( !getGatewayURL().equals( NOGATEWAYURL ) )
        return;

    try
    {
        LISTRWx002DUIx002DPAGE.setInitialValues();
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.EMPNO.assign( 0, "000010" );
        LISTRWx002DUIx002DPAGE.EMPNO.assign( 1, "000020" );
        LISTRWx002DUIx002DPAGE.EMPNO.assign( 2, "000030" );
        LISTRWx002DUIx002DPAGE.EMPNO.assign( 3, "000050" );
        LISTRWx002DUIx002DPAGE.EMPNO.assign( 4, "000060" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.FIRSTNME.assign( 0, "CHRISTINE" );
        LISTRWx002DUIx002DPAGE.FIRSTNME.assign( 1, "MICHAEL" );
        LISTRWx002DUIx002DPAGE.FIRSTNME.assign( 2, "SALLY" );
        LISTRWx002DUIx002DPAGE.FIRSTNME.assign( 3, "JOHN" );
        LISTRWx002DUIx002DPAGE.FIRSTNME.assign( 4, "IRVING" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.MIDINIT.assign( 0, "I" );
        LISTRWx002DUIx002DPAGE.MIDINIT.assign( 1, "L" );
        LISTRWx002DUIx002DPAGE.MIDINIT.assign( 2, "A" );
        LISTRWx002DUIx002DPAGE.MIDINIT.assign( 3, "B" );
        LISTRWx002DUIx002DPAGE.MIDINIT.assign( 4, "F" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.LASTNAME.assign( 0, "HAAS" );
        LISTRWx002DUIx002DPAGE.LASTNAME.assign( 1, "THOMPSON" );
        LISTRWx002DUIx002DPAGE.LASTNAME.assign( 2, "KWAN" );
        LISTRWx002DUIx002DPAGE.LASTNAME.assign( 3, "GEYER" );
        LISTRWx002DUIx002DPAGE.LASTNAME.assign( 4, "STERN" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.WORKDEPT.assign( 0, "A00" );
        LISTRWx002DUIx002DPAGE.WORKDEPT.assign( 1, "B01" );
        LISTRWx002DUIx002DPAGE.WORKDEPT.assign( 2, "C01" );
        LISTRWx002DUIx002DPAGE.WORKDEPT.assign( 3, "E01" );
        LISTRWx002DUIx002DPAGE.WORKDEPT.assign( 4, "D11" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.PHONENO.assign( 0, "3978" );
        LISTRWx002DUIx002DPAGE.PHONENO.assign( 1, "3476" );
        LISTRWx002DUIx002DPAGE.PHONENO.assign( 2, "4738" );
        LISTRWx002DUIx002DPAGE.PHONENO.assign( 3, "6789" );
        LISTRWx002DUIx002DPAGE.PHONENO.assign( 4, "6423" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.HIREDATE.assign( 0, "01/01/1965" );
        LISTRWx002DUIx002DPAGE.HIREDATE.assign( 1, "10/10/1973" );
        LISTRWx002DUIx002DPAGE.HIREDATE.assign( 2, "04/05/1975" );
        LISTRWx002DUIx002DPAGE.HIREDATE.assign( 3, "08/17/1949" );
        LISTRWx002DUIx002DPAGE.HIREDATE.assign( 4, "09/14/1973" );
        LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, "5L" );
        LISTRWx002DUIx002DPAGE.JOB.assign( 0, "PRES" );
        LISTRWx002DUIx002DPAGE.JOB.assign( 1, "MANAGER" );
        LISTRWx002DUIx002DPAGE.JOB.assign( 2, "MANAGER" );
    }
}
```

Figure 15. Enhanced method `initAuthorTimeValues()` for the Employee List bean

## Testing modifications to generated UI Record JSPs, continued

```
LISTRWx002DUIx002DPAGE.JOB.assign( 3, "MANAGER" );
LISTRWx002DUIx002DPAGE.JOB.assign( 4, "MANAGER" );
LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 5L );
LISTRWx002DUIx002DPAGE.EDLEVEL.assign( 0, 18L );
LISTRWx002DUIx002DPAGE.EDLEVEL.assign( 1, 18L );
LISTRWx002DUIx002DPAGE.EDLEVEL.assign( 2, 20L );
LISTRWx002DUIx002DPAGE.EDLEVEL.assign( 3, 16L );
LISTRWx002DUIx002DPAGE.EDLEVEL.assign( 4, 16L );
/* Uncomment the following statement to simulate an error message being */
/* displayed for EDLEVEL when displaying author time values. */
/* getEDLEVEL().setErrorMsg( 0, "Value entered for data item EDLEVEL is invalid" ); */
LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 5L );
LISTRWx002DUIx002DPAGE.SEX.assign( 0, "F" );
LISTRWx002DUIx002DPAGE.SEX.assign( 1, "M" );
LISTRWx002DUIx002DPAGE.SEX.assign( 2, "F" );
LISTRWx002DUIx002DPAGE.SEX.assign( 3, "M" );
LISTRWx002DUIx002DPAGE.SEX.assign( 4, "M" );
LISTRWx002DUIx002DPAGE.ROWSx002DTOx002DDISPLAY.assign( 0, 5L );
LISTRWx002DUIx002DPAGE.BIRTHDATE.assign( 0, "08/24/1933" );
LISTRWx002DUIx002DPAGE.BIRTHDATE.assign( 1, "02/02/1948" );
LISTRWx002DUIx002DPAGE.BIRTHDATE.assign( 2, "05/11/1941" );
LISTRWx002DUIx002DPAGE.BIRTHDATE.assign( 3, "09/15/1925" );
LISTRWx002DUIx002DPAGE.BIRTHDATE.assign( 4, "07/07/1945" );
LISTRWx002DUIx002DPAGE.LINKx002DNEXT.assign( 0, "Next" );
/*
LISTRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 0,
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
*/
}
catch ( Exception e )
{
    System.out.println( "Exception received trying to initialize author time
values: " + e.getMessage() );
}
}
```

Figure 15. Enhanced method `initAuthorTimeValues()` for the *Employee List* bean, continued

Figure 16 shows an enhanced method `initAuthorTimeValues()` for the employee detail UI record. Changes are highlighted. This version of the method was used to set the initial values displayed in Figure 14.

```
// Set author time initial values for data items associated with HTML elements.
// The initial values are based only on UI Type and data item of the items.
public void initAuthorTimeValues()
{
    if ( !getGatewayURL().equals( NOGATEWAYURL ) )
        return;

    try
    {
        DETAIRWx002DUIx002DPAGE.setInitialValues();
        DETAIRWx002DUIx002DPAGE.EMPNO.assign( 0, "000010" );
        DETAIRWx002DUIx002DPAGE.FIRSTNME.assign( 0, "CHRISTINE" );
        DETAIRWx002DUIx002DPAGE.MIDINIT.assign( 0, "I" );
        DETAIRWx002DUIx002DPAGE.LASTNAME.assign( 0, "HAAS" );
        DETAIRWx002DUIx002DPAGE.WORKDEPT.assign( 0, "A00" );
        DETAIRWx002DUIx002DPAGE.PHONENO.assign( 0, "3978" );
        DETAIRWx002DUIx002DPAGE.HIREDATE.assign( 0, "01/01/1965" );
        DETAIRWx002DUIx002DPAGE.JOB.assign( 0, "PRES" );
        DETAIRWx002DUIx002DPAGE.EDLEVEL.assign( 0, 18L );
    }
}
```

```

        /* Uncomment the following statement to simulate an error message being */
        /* displayed for EDLEVEL when displaying author time values. */
        /* getEDLEVEL().setErrorMsg( 0, "Value entered for data item EDLEVEL is invalid" ); */
        DETAIRWx002DUIx002DPAGE.SEX.assign( 0, "F" );
        DETAIRWx002DUIx002DPAGE.BIRTHDATE.assign( 0, "08/24/1933" );
        DETAIRWx002DUIx002DPAGE.SALARY.assign( 0, new VGJBigNumber( "52750.00" ) );
        /* Uncomment the following statement to simulate an error message being */
        /* displayed for SALARY when displaying author time values. */
        /* getSALARY().setErrorMsg( 0, "Value entered for data item SALARY is invalid" ); */
        DETAIRWx002DUIx002DPAGE.BONUS.assign( 0, new VGJBigNumber( "1000.00" ) );
        /* Uncomment the following statement to simulate an error message being */
        /* displayed for BONUS when displaying author time values. */
        /* getBONUS().setErrorMsg( 0, "Value entered for data item BONUS is invalid" ); */
        DETAIRWx002DUIx002DPAGE.COMM.assign( 0, new VGJBigNumber( "4220.00" ) );
        /* Uncomment the following statement to simulate an error message being */
        /* displayed for COMM when displaying author time values. */
        /* getCOMM().setErrorMsg( 0, "Value entered for data item COMM is invalid" ); */
        /*
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 0,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 1,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 2,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 3,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 4,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 5,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 6,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 7,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 8,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        DETAIRWx002DUIx002DPAGE.INFORMATIONx002DMESSAGE.assign( 9,
        "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" );
        */
    }
    catch ( Exception e )
    {
        System.out.println( "Exception received trying to initialize author time values: "
        + e.getMessage() );
    }
}

```

Figure 16. Enhanced method *initAuthorTimeValues()* for the *Employee List* bean

**Conclusion**

Web page designers who customize UI record JSPs do not need to access a web transaction to view the web pages produced by those JSPs. The web pages can include initial values (either default or realistic), as well as error messages. ■

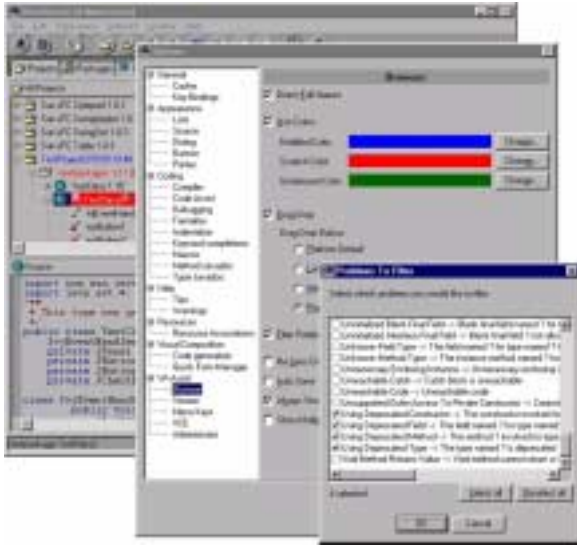
## Acronyms

<b>3GL</b>	third-generation language
<b>4GL</b>	fourth-generation language
<b>AIX</b>	Advanced Interactive Executive
<b>API</b>	Application Programming Interface
<b>AS/400</b>	Application System/400
<b>CAE/2</b>	Client Application Enabler/2
<b>CASE</b>	Computer-aided Software Engineering
<b>CICS</b>	Customer Information Control System
<b>CICS OS2</b>	Customer Information Control System Operating System/2
<b>CPU</b>	central processing unit
<b>CSP</b>	Cross System Product
<b>DB2</b>	Database 2
<b>DBCS</b>	double-byte character set
<b>DBMS</b>	database management system
<b>DCE</b>	distributed computing environment
<b>DRDA</b>	distributed relational database architecture
<b>EMEA</b>	Europe/Middle East/Africa
<b>GUI</b>	graphical user interface
<b>IBM</b>	International Business Machines
<b>IMS</b>	Information Management System
<b>LAN</b>	Local Area Network
<b>MSL</b>	member specifications library
<b>MVS</b>	Multiple Virtual Storage
<b>NT</b>	Microsoft Windows NT
<b>OS/2</b>	Operating System/2
<b>OS/390</b>	Operating System/390
<b>OS/400</b>	Operating System/400
<b>RAD</b>	rapid application development
<b>SQL</b>	Structured Query Language
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>VM</b>	Virtual Machine
<b>VSE</b>	Virtual Storage Extended
<b>WWW</b>	World Wide Web



# Instantiations, Inc. announces the release of VA Assist Enterprise/G™ Version 2

VA Assist Enterprise/G is a product that follows in the rich tradition of providing major new capabilities to IBM's popular VisualAge® Generator. In addition to providing the extensive set of power tools for repository management, GUI building and code development - VA Assist Enterprise/G supports both the Java and Smalltalk language options in VisualAge Generator! VA Assist Enterprise/G integrates seamlessly into the familiar parts browsers and visual composition editors, adding hundreds of new features and commands.



In summary, VA Assist Enterprise/G lets new and experienced VisualAge Generator developers easily harness the power that Java brings to the environment.

Developers will find that VA Assist Enterprise/G encourages them to use a much broader spectrum of VisualAge Generator's capabilities, increasing their productivity and ease of use along the way.

Instantiations is an IBM Business Partner and has been providing products and services to VisualAge customers for years. Thousands of developers are currently using Instantiations' VisualAge productivity tools.

VA Assist Enterprise/G Expands on our Triple Your Productivity Claim by Providing VisualAge Generator Users with Powerful Repository Management, Code Development and GUI Building Capabilities.

– Eric Clayberg, Sr. Vice President of Product Development, Instantiations, Inc.

"We are delighted that Instantiations has developed VA Assist Enterprise/G that fully supports Java

"From the powerful library administration aides to the fast-path GUI building capabilities, I have found VA Assist Enterprise/G to be a valuable add-on to VisualAge Generator. Check it out!"

– Gary Johnston, Product Manager, VisualAge Generator.

## Some of the Repository Management features included:

- Version renaming commands – baseline app names with ease!
- Version name templates allow the developer to specify the scheme used for generating new version names from prior releases.
- Super user features – manipulate editions and version regardless of ownership.
- Super group - enables the Administrator to set up special ownership and modification rules for specific packages.
- New commands to copy projects, compare different projects and packages, create new editions of projects, packages and types.
- Replace a project with an arbitrary instance of any other (unloaded) project.

## Some of the Code Development features included:

- Color highlighting of modified, scratch and unreleased editions.
- Enhanced import and export Wizard for efficient naming and saving.
- Powerful nesting capabilities with batch export sets.
- Intuitive and integrated drag-and-drop for moving or copying classes or projects.
- Keyboard shortcuts (accelerators) that can be assigned to any menu items. Multiple menu key profiles may be created!
- Handy view filtering allows the developer to filter project and package lists to show only the items of interest.
- Problem filtering allows developers to filter the problems page by project and type of problem.

## Some of the GUI Building features included:

- Unravel the visual complexity by filtering connections.
- Gain workspace clarity by auto- hiding all connections except those originating from or terminating at selected components.
- Alt-click on a widget to change its label or title with direct editing.
- Developers are able to easily select all components or all component of the same type.
- Any visual component may be auto-sized with newly added commands.
- Move or adjust the size of selected components in pixel increments.
- Set the size of the VCE script editor according to your preference!



**You can download a free, fully functional 30-day evaluation version of VA Assist Enterprise/G from Instantiations' web site:**

<http://www.instantiations.com/assist>

**For more information, contact Instantiations at 800-808-3737.**

# 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. E9WB/503  
P.O. Box 12195  
3039 Cornwallis Road  
RTP, NC 27709  
USA  
ahawk@us.ibm.com

© Copyright International Business Machines Corporation 2001. All rights reserved. Printed in U.S.A.

The following terms used in this publication are trademarks or service marks of the IBM Corporation in the United States or other countries or both: AIX, AS/400, CICS, CICS OS2, COBOL, Database 2, DataJoiner, DB2, DB2/2, DB2/400, DB2/6000, IBM, IMS, LE/370, MQSeries, MVS, VM, VSE, Operating System/2, OS/2, OS/390, OS/400, RISC System/6000, SQL/DS, WebSphere, VisualAge, and VisualGen.

The following terms and phrases used in this publication are trademarks or service marks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U. S. and other countries.

Informix is a trademark of the Informix Corporation.

Oracle is a trademark of Oracle Corporation.

ENVY is a trademark of Object Technology International, Inc.

HP is a trademark of Hewlett-Packard Company.

Microsoft, Windows, Windows NT, the Windows 95 logo, Visual Basic, and ActiveX are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names may be trademarks or service marks of others.

IBM has made reasonable efforts to ensure the accuracy of the information contained in this publication. However, this publication is presented "as is" and IBM makes no warranties of any kind with respect to the contents hereof, the products listed herein, or the completeness or accuracy of this publication. Customer experiences may be different from those described here. IBM does not warrant any non-IBM programs or products, which are described in this newsletter. These articles are for information only, and you should contact the stated company with your questions.

G242-0315-13

