



VisualAge[®] Generator

A Powerful New Vision of Programming™

Volume 3, Number 2
April/May 1998

Contents

New Kid on the VisualAge Generator Block	2
Client/Server Application Development with VisualAge Generator at Deutsche Ausgleichsbank	3
SQL Joins, Unions, and Subselects	5
Building Portable Views in VisualAge Generator	6
Monitoring Java Client Sessions	10
Interoperability—Calling 3GL Programs	14
Introduction to Automated Testing of VisualAge Generator Clients with SilverMark's Smalltalk Test Mentor	19
Moving Your Application to the Internet	22



New Kid on the VisualAge Generator Block

by Peter Spung, VisualAge Generator Product Manager

If you've read previous articles on this page, you'll notice I'm a new kid on the block. My name is Peter Spung, and I've recently been named the VisualAge Generator Product Manager. Sandra Johnson has taken a new position in IBM, managing the development of our Web server product line—Lotus Domino Go Webserver. I certainly have some big shoes to fill but, thankfully, Sandra and the team of incredibly talented folks working on VisualAge Generator have been very helpful to me in my transition to this new assignment. I want to thank Sandra for her leadership, vision, and inspiration and wish her continued success.

I come to this job with a background in big systems integration and development, having spent my formative years in a part of IBM that later became IBM Global Services. Developing and integrating large, complex software systems for customers taught me a lot about the kind of application development issues and challenges you face every day. And the importance of development tools, like VisualAge Generator, that help you attack these problems and deliver the application systems your users expect while maximizing your productivity and allowing you to reuse existing programs and skills. For the last eight years, I've been developing various products in IBM's line of application development products: ISPF/SCLM, VisualAge C++, VisualAge Smalltalk, and VisualAge Java. I've learned that we can provide you with very powerful tools by combining the flexibility and ease-of-use of PC-based development tools, with robust, high-performing, highly scalable server solutions like CICS, IMS, DB2 on platforms like OS/2,

Windows NT, AIX, and MVS. That's what really excites me about VisualAge Generator, and the prospect of working with people like you who understand these issues and the value that a product like VisualAge Generator brings to you.

As you probably have read, the Nagano Olympics was a big success. What you may not know is that two critical systems, the Accreditation Enterprise Database and the Games Staffing System, were developed using VisualAge Generator. The Accreditation system provided around-the-clock badging—validating credentials and giving the right people access to the right venue at the right time. The Games Staffing System electronically distributed results to the international trade press—automatically gathering results from multiple databases, reformatting the data, and sending this information to awaiting sports journalists who got the story out. These systems performed flawlessly, thanks in large part to VisualAge Generator and the powerful underlying middleware from IBM. To read more about these systems, or how customers are using VisualAge Generator to create solutions to their business problems, check out our Web page at <http://www.software.ibm.com/ad/visgen/>.

One intriguing item you'll find on the Web page is a fact sheet about our recent technology demonstration at the JavaOne conference. We showed how VisualAge Generator can be used to develop Enterprise JavaBeans (EJB) compliant objects. Enterprise JavaBeans provide a single component model that allows programmers to build new distributed transactional server systems, leveraging new and

existing function and data, in the same plug-and-play fashion used on some smaller desktop systems (like client JavaBeans and ActiveX). VisualAge Generator's support for EJB generation, a prototype of which we demonstrated at JavaOne, provides programmers the same easy-to-use high-level specification approach for development for EJBs that has proved so productive for the development of your systems. Once again, we are demonstrating our commitment to continue to enhance VisualAge Generator in order to incorporate and support new technologies without requiring you to learn all the details and nuances of those systems while, at the same time, allowing you to leverage your existing programs and skills, and adopt the new technology at your own rate and pace.

Our team here in the Lab is very busy working on the next release of VisualAge Generator. Since this is work in progress, we will describe it in detail in a subsequent edition of the newsletter. You will be pleased to know we are expanding our native database support to another database manager, expanding the platforms we cover with our ODBC and JavaBeans support, and improving the performance of our server runtimes.

Let me close by thanking you for your interest in VisualAge Generator. It is through your use and experience we learn how to improve our product, and make it even better. If you have suggestions for me about the product, or topics you'd like me to cover in future letters, you can write to me in care of this newsletter using the comment form near the back. Or send me e-mail; I'd like to hear from you: paspung@us.ibm.com.

Client/Server Application Development with VisualAge Generator at Deutsche Ausgleichsbank

Who is the “Deutsche Ausgleichsbank” (DtA)?

Although a financial institution, Deutsche Ausgleichsbank (DtA) is not a typical bank, but, rather, a service-oriented development agency for business start-ups of small- to medium-size enterprises: it promotes entrepreneurial ideas, investments, and innovations on behalf of the German government by providing financial products at favorable conditions, risk capital, guarantees, and individual consultancy. DtA finances investments in the fields of innovative technology, environmental protection technology, and professional training. With its 680 employees and 83 billion deutsche marks of assets, it has helped hundreds of thousands of people establish new businesses, mostly in central and eastern Europe.

The Challenge

In the early 1990's, DtA's Information Technology was facing a classic “legacy” challenge. While business pressures demanded many new automated solutions, delivering these solutions within the existing IT infrastructure was a daunting task. More than 80 percent of the programming staff was required to maintain a rigid, batch-oriented, VSAM-based set of PL/I programs that represented the organization's core application system. The use of PCs as a tool to enhance knowledge workers' productivity was sporadic and isolated, and the system's 3270-based OLTP systems severely limited users' productivity.

After careful analysis and several technology studies, DtA established a set of key objectives and an associated IT strategy.

1. Improve end-user productivity by installing a networked personal computer for each end user and providing applications for personal productivity, access to workgroup data, and access to enterprise data and applications.
2. Improve the reliability, flexibility, and maintainability of enterprise data by storing and managing it in a relational database.
3. Deliver applications faster, while ensuring that they are easier to use, of higher quality, and easier to maintain. This will be made possible through the use of data-model-driven application development methodologies and the use of a RAD solution to replace PL/I-based development.

Strategy Implementation Decisions

DtA quickly decided on an implementation approach. First they defined the hardware and software standards for each user's workstation. Next they selected DB2 as their strategic relational database system. And then they selected their primary tools suite to support their RAD development approach.

The choice of a suitable RAD application-generation tool was the most difficult step. The tool had to meet a rigorous set of technical criteria and be able to:

- Deliver client/server systems that could reuse and integrate COBOL and PL/I code components easily and without major need for re-architecture
- Generate applications for Windows, OS/2, AIX, and MVS
- Support several data management systems including DB2, DB2/2, DB2/6000, and VSAM
- Provide superior programmer productivity
- Be easy for programmers to learn and use
- Deliver state-of-the-art workstation-based development facilities for applications intended for execution on local network environments, as well as those intended for execution on host environments.

After evaluating AD tools from several vendors, DtA chose IBM's VisualAge Generator, the only RAD solution that met all their requirements.

The introduction of PCs on the employees' desktops quickly resulted in a high level of acceptance of the graphical user interface, and as a direct consequence all requirements for new business applications demanded standard GUI-based access.

VisualAge Generator provided DtA with the ideal LAN-based single programming tool to create the required variety of system execution topologies: from traditional Host batch programs, to OLTP transactions, to PC or LAN systems. DtA programmers were particularly pleased with the ease of the 4GL which, unlike PL/I, significantly simplified the tedious task of low-level programming to execution environment APIs and did not require the need to develop specialized skills for the new DtA execution platforms.

Because of the breadth of the enabled solutions it provides, VisualAge Generator requires the right mix of skills. A critical success factor in DtA adoption of VisualAge Generator was the gradual and mentored introduction of the technology. A management sponsor was nominated for the initial deployment of the new AD environment, and a pilot project was identified and staffed with a core team of developers that consisted of both DtA employees and external consultants.

The pilot project was extremely successful and DtA proceeded in deploying the new development technology throughout the organization. The knowledge gained during the pilot project was then extended to other applications areas while the core team became the in-house mentoring staff necessary for large-scale training and consulting.

It is with Customers such as DtA in mind that IBM designed and brought to market VisualAge Generator, and DtA's success is a clear confirmation that the product strengths combined with intelligent planning and serious commitment are a true recipe for success.

Acronyms

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

SQL Joins, Unions, and Subselects

by Fiona Mader and Jeri Petersen

In many cases in VisualAge Generator, the default SQL created for you based on the SQL row records is what you need. However, joins, unions, and subselects have special requirements.

Joins

For a join, create a SQL row record that includes all the tables. Then use that record in a process. The process is automatically created with default SQL for all the columns in the record and with a FROM statement for all the tables listed in the properties of that record. This is fairly basic, and information about it can be found in the VisualAge Generator online Help.

Unions and Subselects

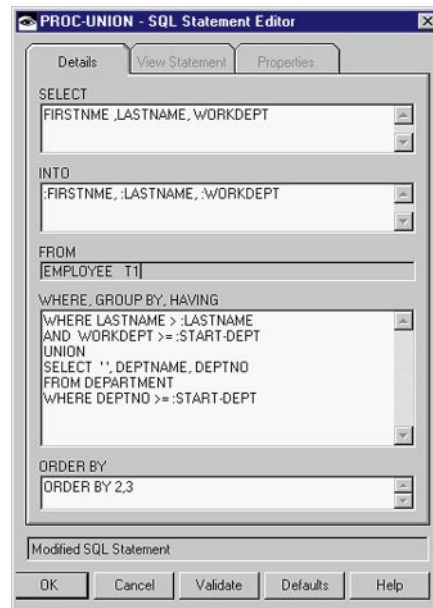
Unions may not be as obvious as joins, but they are very simple. As shown in the diagram, the Union statement is coded into the space that has been preallocated to the WHERE clause. This space is not restricted to being used only for the WHERE clause. Selecting the View Statement tab to display the entire SQL statement should make this evident.

The main point is that with a union there is only one INTO clause used. Both SELECT statements will return rows into the same host variables as specified in the INTO clause.

The Union must follow the rules of the database. For example, the number and definition of selected columns must match. Use of the **Validate** push button will ensure that the SQL statement will validate against the database and all rules have been adhered to.

Similar rules can be applied for the creation of subselects.

Ninety percent of the time, the default SQL statement that is created will be what you need. However, when there is a requirement to use unions and subselects, you should code these in the space reserved for the WHERE clause.



Building Portable Views in VisualAge Generator

by Jay Cagle, Rebecca Schaller, and Hayden Lindsey, *VisualAge Generator Development*

VisualAge Generator supports the creation of Graphical User Interfaces (GUIs) by exploiting the construction-from-parts paradigm common to the VisualAge family of products. In fact, VisualAge Generator V3.0 is built on top of VisualAge for Smalltalk. Thus, all of the visual programming facilities provided by VisualAge Smalltalk are also available in VisualAge Generator. While VisualAge Generator provides much in the way of visual programming, creating visually appealing, usable, and portable GUIs requires more than simply dropping controls inside a window shell in the Composition Editor. In this article, we describe techniques that can be employed to ensure portability of the GUIs you produce with VisualAge Generator. This portability allows you to deploy GUIs on machines that vary from the machine you used to develop the GUI, while maintaining their full function and attractive appearance.

Portability of user interfaces is an area that is often misunderstood or overlooked. When a GUI is built on a particular machine, it may be sized and positioned appropriately. However, when the end-user sees this GUI, controls may be overlapped, truncated, or completely hidden. This can happen because the portability of a GUI is affected by the following factors: differences in screen resolutions, fonts, operating systems, and end-user language translations. To achieve portability, one must understand the basic techniques of attachments in order to eliminate these factors.

Attachments specify the positional relationship between controls within a window. When a control is dropped in a VisualAge Generator window frame, it is actually being placed within another control—its parent. Each side of a control can be attached to its parent or another control. The nature of these attachments are the primary factor in determining the relative portability of the resulting GUI.

Each side of a control can be attached in one of the following ways:

- Fixed position within the parent form.

This is the default behavior within VisualAge Generator. When a push button is dropped, the exact position as seen in the Composition Editor is the same position that will be used at runtime. While this is truly WYSIWYG, this does not accommodate portability. Thus, this default attachment style should be appropriately altered using one of the following methods.

- Relative position with the parent form

A somewhat more flexible manner of attachment is to associate the sides of a control to relative positions (that is, 1–100) within the parent. For example, if a list box is included within a window and attached to the parent form on the left to position 5 and on the right to position 95, it will grow and

shrink whenever the parent window frame is resized. This is a very useful form of attachment, especially when dividing the client region of the window between more than one control (that is, if two list boxes need to be displayed side-by-side). Within VisualAge Generator, this attachment type is known as ATTACHPOSITION.

- To the parent form

When a control needs to be positioned close to the boundary of the parent control, you can attach the control to its parent. Additionally, an offset can be specified. In fact, this is the same attachment type (fixed position within the parent form) as in the first bullet above. The difference is the offset is used as a margin specification, rather than an absolute position within the parent. In the list box example above, a similar appearance could be achieved by attaching the left and right sides to the parent form, and then specifying a common offset such as 10 pixels. On initial display, this may be indistinguishable from a window built using relative positioning. However, when the window is resized, the margin (10 pixels) will not change, whereas the margin will change when using relative positioning. This type of attachment is known as ATTACHFORM.

- To a peer control

When a control is positioned close to another control, you can attach one or more of the control's sides to the adjacent control. As when attaching to the parent, an offset can be specified. This form of attachment is particularly useful when building more complex GUIs where a control is adjacent to another control and not the parent. For example, if a window is to have two side-by-side list boxes, then the rightmost list box could be attached on the left to the right side of the leftmost list box, with an offset to improve the appearance. Once done, if the leftmost list box should grow, the rightmost one will simply slide over, making room for its larger peer control. This method of attachment is known as `ATTACHWIDGET`. The term widget comes from Motif, on which the VisualAge GUI architecture is based.

- To itself

This attachment type applies to the height and width of a control. For example, if a push button is dropped and resized, by default the bottom and right sides will be attached to its top and left sides, respectively. The offset for the attachments will be fixed, so that at runtime the push button will be sized exactly as it was in the Composition Editor. As with the first bullet, this will not be portable. This attachment type is called `ATTACHSELFOPPOSITE`.

- To nothing

To allow controls to grow, two of the sides generally need to be unattached. If controls are being attached from top to bottom and left to right, then the bottom and right sides of a control should be unattached, thus determining the directions in which the control will grow or shrink should its contents change. With no attachment, the system will size the control appropriately. In the case of a push button, it will be sized to contain its label. Within VisualAge Generator, this type of attachment is known as `ATTACHNONE`.

Finally, when attaching a control to its parent or to another control, there are two modes worth mentioning: adjacent and opposite. `ATTACHFORM` and `ATTACHWIDGET` specify that the side of the control is attached to the adjacent side of the parent or peer. For instance, if `ATTACHFORM` is specified for the left side of a push button, it will be attached to the left side (the adjacent side) of the parent form, using the given offset. If two buttons are side by side, and `ATTACHWIDGET` is specified for the left side of the right button, it will be attached to the right side (the adjacent side) of the left button. You may also use `ATTACHOPPOSITEFORM` and `ATTACHOPPOSITEWIDGET`. These two attachment types specify that the side of the control is attached to the opposite side of the parent or peer. With two buttons side by side, specifying `ATTACHOPPOSITEWIDGET` for the left side of the right button will attach it to the left side (the opposite side) of the left button. While this doesn't make sense if the buttons are side by side, it could be used if one button were below the other. With an offset of zero, the left sides of the buttons would be aligned.

Since no single mode of attachment is appropriate for all the controls within a GUI, several attachment styles listed above can be used in combination. However, the styles of bullets 1 and 5 should be voided if at all possible. Within VisualAge Generator, attachments are specified using the `framingSpec` dialog box from a control's property sheet.

Now that we have introduced attachments as the general architecture upon which GUIs in VisualAge Generator are built, the remainder of this article provides suggestions about how to use this information and the associated capabilities to build portable GUIs.

The basic idea of attachments is that you describe the relationships between parts in your GUI, and let the system take care of sizing and positioning them. For the most part, you should not make assumptions about the size of a part, because this size will vary as a result of NLS translation, fonts, screen resolution, and operating system. When adding attachments to a view, in general you should start in the upper left corner and work your way down and to the right. As you proceed, you attach each part to one or more parts surrounding it.

For parts that contain *static text*, such as labels, push buttons, and toggle buttons, one attachment in each direction should be set to `ATTACHNONE`. For example, a label might be attached to a parent form on the top and left, with the bottom and right set to `ATTACHNONE`. This tells the system to make the label as wide and tall as necessary to accommodate the text using the current font.

For parts with *dynamic text*, such as entry fields, drop-down lists, MLEs, and list boxes, ATTACHNONE is also useful. These parts all have attributes to tell the system how to size the part when ATTACHNONE is specified. Entry fields and MLEs have a *columns* attribute, which is a number indicating how many characters wide the part should be. For instance, if you have an entry field with *columns* set to 10, with the left side attached to the form and the right side ATTACHNONE, the entry field will be sized to display 10 characters (based on the average width of the characters in the current font).

List boxes, drop-down lists, and combo-boxes have a *visibleItemCount* attribute, which indicates how many lines to display. So, if you have a list box with *visibleItemCount* set to 5, with top as ATTACHFORM and bottom as ATTACHNONE, the list box will be sized to display 5 lines (again based on the current font). The list parts do not have a *columns* attribute. Instead they will set their width to the size of the widest item. In order for this to happen, however, the items of the list must be set before the view is laid out. If the items are set dynamically, you can do this in a *finalInitialize* Smalltalk method, but *AboutToOpenWidget* and *openedWidget* are too late. In this case, if the list doesn't contain any items when the view is laid out, it will be sized to a default width. In the case of drop-down lists and combo-boxes, the width will be 20 characters. List boxes don't have a default width, so if the list is initially empty, it will be sized very narrow. In this case, you'll have to do something else to set a reasonable initial size.

Container parts, such as container details, do not have either *columns* or *visibleItemCount* attributes, so you must handle sizing both the width and height yourself.

In addition to their use when ATTACHNONE is specified, these sizing algorithms and attributes are also used when a part is attached to the form or another part. In these cases, the calculated size will be used to set the initial size of the part. For instance, suppose you have a window that contains a single list box. The items of the list box are set to ('a' 'bb' 'cccc') in the list box properties. *visibleItemCount* is set to 7, and the top, left, bottom, and right attachments are all set to ATTACHFORM. When the window is initially displayed, the listbox will be sized wide enough to display 'cccc' and tall enough to display 7 items. However, as you size the window, the list box will shrink and stretch.

The last category of parts is what VisualAge calls Canvas parts. These include windows, forms, and group boxes. When ATTACHNONE is specified, these parts will size themselves to fit their contents. If ATTACHFORM or ATTACHWIDGET is used, the parts' initial size will be set to fit their contents. As a simplified example, consider a group box that contains two label parts whose strings are 'aa' and 'bbbb'. The upper label is attached on the top and left to the form (the group box in this case), and the bottom and right are ATTACHNONE. The lower label is attached on top to the upper label, on left to the form, and bottom and right are ATTACHNONE. The group box is attached on top and left to the form, and bottom and right are ATTACHNONE. When displayed, the two labels will size themselves to fit their strings, and the group box will set its width and height just large enough to contain the two labels. In this case, the width of the group box is set to contain the width of the lower label string 'bbbb'. If, during language translation, the upper string is translated to 'xxxxxxx', the group box will now be sized large enough to hold it.

The fact that canvas parts size themselves to contain their largest child part can be used to make a set of parts all the same size. Consider the case where you have three push buttons aligned in a column. Because the labels will change during translation, you want to set the attachments so the buttons determine their own size. However, you would also like the buttons to be the same width. You can solve this by placing the buttons within a form. Each button is attached on the left and right to the form. The bottom and right attachments of the form are ATTACHNONE. When displayed, each button will set its initial size based on its label. The width of the form will then be sized based on the widest button. However, since the buttons are attached to the form, the two narrower buttons will be stretched to the width of the form, which is the width of the largest button.

The equivalent of setting ATTACHNONE for a window is to blank out the *Width* and *Height* entry fields in the window's framingSpec property dialog. If this is done, the window will size itself just large enough to contain its contents. If you blank out the X and Y entry fields, the initial position of the window will be left to the system.

However, due to a limitation of the Composition Editor, if you blank out these fields, when you reopen your view in the editor, the window will be sized very small. In order to get around this limitation, a little Smalltalk coding is required. Leave the fields as is, and implement the following:

```
AbtShellView>>xyzSetSizeAndPositionToDefault
self initWidgetSize: nil.
```

```
AbtBasicView>>xyzSetSizeAndPositionToDefault
self framingSpec: nil.
```

where 'xyz' is some unique prefix to avoid collisions with any future system methods. Then, add the method *finalInitialize* to your view class. Its method body should be simply 'self xyzSetSizeAndPositionToDefault'. Since *finalInitialize* is a message sent to each view when it has been created but before it is opened, this will have the same effect as blanking out the X, Y, Width, and Height fields but will not cause a problem when the view is reopened in the Composition Editor.

Finally, to ensure that canvas parts (such as group boxes and forms) within a window behave properly, you must also do a similar trick to them. You'll want to specify ATTACHNONE for two sides of your canvas parts, so that the system will size them according to their contents. But again, due to the Composition Editor limitation, if you do so these parts will be sized very small when you reopen your view in the editor. Instead, specify ATTACHSELFOPPOSITE where you would normally specify ATTACHNONE, and implement the following:

```
AbtBasicView>>xyzSetCompositeAttachmentsToDefault
"Do nothing. This catches the Message for all non-composite widgets."

AbtCompositeView>>xyzSetCompositeAttachmentsToDefault
| fs |
( fs := self framingSpec ) notNil ifTrue: [
    fs bottomEdge xyzSetAttachmentToDefault.
    fs topEdge xyzSetAttachmentToDefault.
    fs leftEdge xyzSetAttachmentToDefault.
    fs rightEdge xyzSetAttachmentToDefault].
self components do: [ :comp | comp hptSetCompositeAttachmentsToDefault ].

AbtEdgeConstant>>xyzSetAttachmentToDefault
"Not a constraint; don't do anything."

AbtEdgeConstraint>>xyzSetAttachmentToDefault
self attachment = XmATTACHSELFOPPOSITE ifTrue: [
    self attachment: XmATTACHNONE ].
```

You will also need to add

"self xyzSetCompositeAttachmentsToDefault" to your *finalInitialize* method.

xyzSetCompositeAttachmentsToDefault is similar to *xyzSetSizeAndPositionToDefault*. For all composite (canvas) parts in your window, it will change all occurrences of ATTACHSELFOPPOSITE to ATTACHNONE. This is the only case where you should use ATTACHSELFOPPOSITE.

With these attachment techniques you can account for differences in the developer's and the end-user's machines. While creating the proper attachments can be a time-consuming process, it is necessary to solve the portability issues introduced by differing screen resolutions, fonts, operating systems and end-user language translations. By following these recommendations, you can be confident the interfaces you design will be displayed correctly for the end-user.

Monitoring Java Client Sessions

by Henry Koch, VisualAge Generator Development

The Java Wrapper support shipped with VisualAge Generator V3.0 FixPak2 includes a new graphical interface, called the session manager, to help you monitor Java client sessions started on the web server machine. The session manager, written in Java, allows you to:

- Display response time statistics on client sessions started on the web server machine
- Automatically cancel sessions that have been inactive for a specified time, or explicitly cancel any active session
- Trace activity on any active session
- Log requests made through the Web server

Installing the Java Wrapper Support

To install the new features in the Java Wrapper support of FixPak2, run the javao command file for OS/2 or the javaw batch command file for Windows. These command files are found in the directory where you installed VisualAge Generator Common Services. Installing the Java support enhancements causes the session manager to appear when you start the UnitOfWorkServer on the web server machine. You can find details of how to install and use the Java Wrapper in the section "Calling Server Programs via VisualAge Generator JavaBeans" of the *VisualAge Generator Client/Server Communications Guide* (SH23-6602) or by using a web browser to browse file csojava.html. This html file is found in the directory specified as the target directory when running javao or javaw. The csojava.html file contains instructions on how to use the session manager's user interface.

Starting the UnitOfWorkServer

Before you can download an applet from the web server, you must do the following:

1. Start web server software, for example Lotus Go WebServer, to enable downloading html pages that have an <applet> tag referencing the applet.
2. Start the Java RMI registry on the web server machine with the command *rmiregistry*
3. Start the UnitOfWorkServer with the command:

```
java ibm.cso.UnitOfWorkServerImpl trace_level timeout_interval refresh_interval
```

where:

trace_level	Level of request reporting. Controls the reporting of requests handled by the UnitOfWorkServer. 0 = no reporting 1 = report errors (default value) 2 = report all activity
timeout_interval	Interval for automatic closing of inactive sessions in minutes. 0 = no closing. Default is 60 minutes.
refresh_interval	Server session report interval in minutes. 0 = no report. Default is 5 minutes.

The Session List Window

When the UnitOfWorkServer starts, it brings up the session list window as shown in Figure 1.

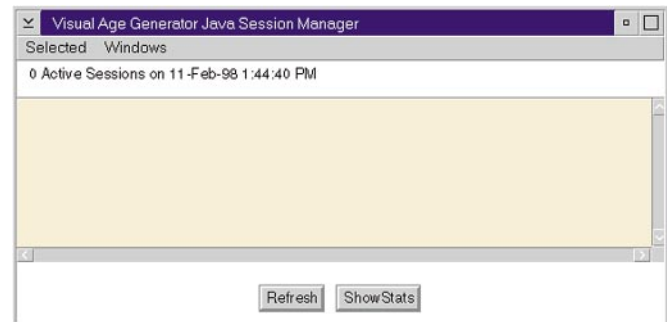


Figure 1. Initial Session List Window

At the end of each refresh interval the session list window is updated with the sessions currently active as shown in Figure 2. Average response time statistics are displayed for all calls made for each session in the list. You can click on the **Refresh** push button at any time to end the current refresh interval and display the active sessions.

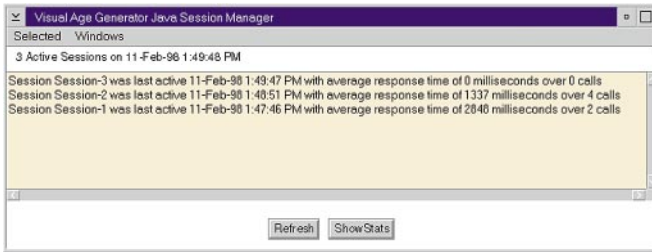


Figure 2. List of Active Sessions

If you want to view average response time statistics for all UnitOfWorkServer requests since the UnitOfWorkServer was started, click on the **Show Stats** push button. The window shown in Figure 3 is then displayed, showing statistics for calls made during the last refresh interval as well as statistics for all calls.

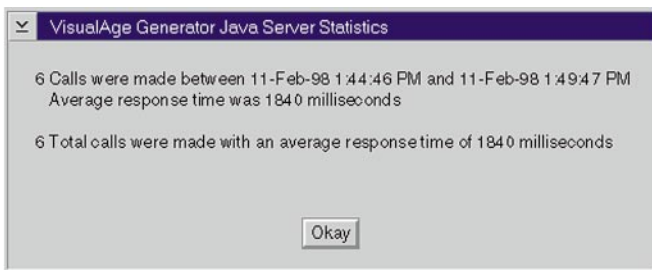


Figure 3. Response Time Statistics for All Sessions

If you want to view statistics for a particular session, select the session in the session list, click on **Selected** in the menu bar, then click on the **Show details** menu item. A dialog appears similar to Figure 4, giving average response times for all requests made for the session.

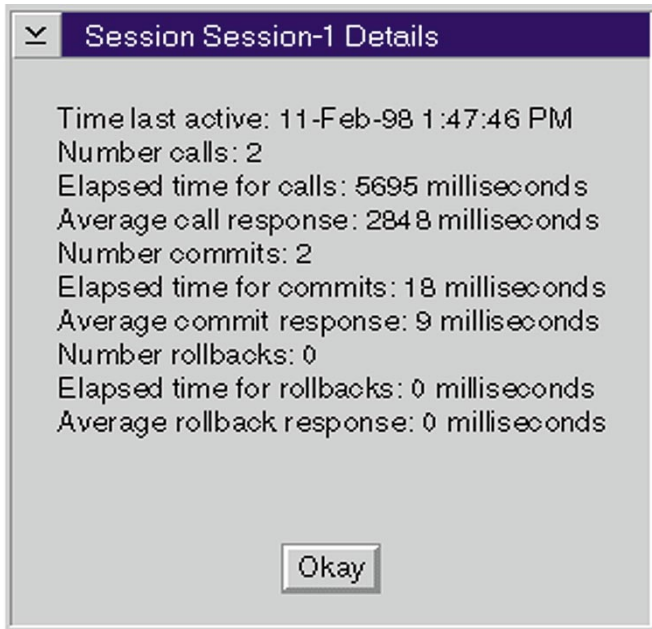


Figure 4. Response Time Statistics for A Session

Changing Session Manager Settings

You can initialize settings for refresh interval, timeout interval, and trace level reporting when you start the UnitOfWorkServer. You can change these options, and set others, using the **Settings** window shown in Figure 5. Open the **Settings** window by clicking on **Windows** in the menu bar, then click on the **Settings** menu item.

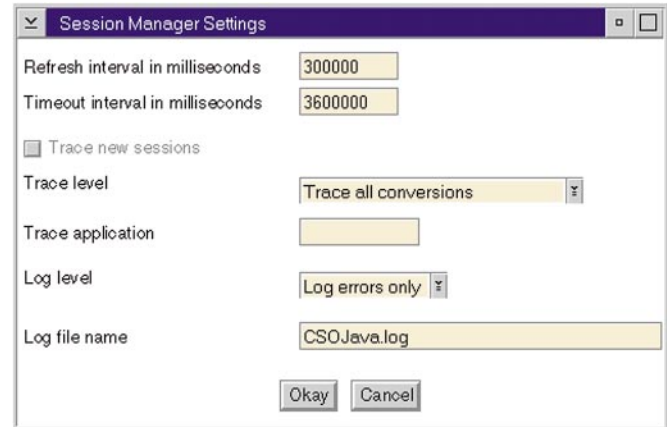


Figure 5. Settings Window

You change refresh interval and timeout interval by entering their new values in their respective entry fields. A value of zero means no automatic refreshing of the session list or no automatic closing of sessions.

You can control the initial trace level set in new trace windows using the **Trace level** drop-down list shown in Figure 6. If **No trace** is selected, you cannot open new trace windows. If **Trace errors only** is selected, you may open new trace windows, and those trace windows will contain entries only for requests that result in an error. If **Trace all** is selected, you may open new trace windows, and those trace windows will contain an entry for each request.

If you set trace level to **Trace conversions by application** and you enter the name of a server program to be traced, parameters for all calls to the specified program are dumped to file CSODUMP.OUT in the directory where the UnitOfWorkServer was started. You can dump all parameters for all calls to all server programs by selecting **Trace all conversions**. The *VisualAge Generator Client/Server Communications Guide* (SH23-6602) provides information on interpreting the data dumped to CSODUMP.OUT. No new trace windows can be opened when the trace level is set to **Trace all conversions** or to **Trace conversions by application**.

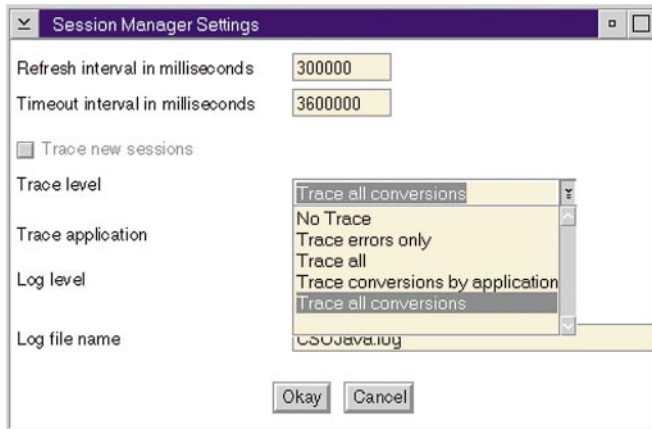


Figure 6. Trace Level Selection In Settings Window

You can have the session manager open a trace window for each new session by clicking on the **Trace new sessions** check box. (This check box is disabled unless the trace level selected is **Trace all** or **Trace errors only**.) Each new trace window is opened with the trace level of the **Settings**. An example of a trace window is shown in Figure 7. Once open, the trace level for a session can be changed at any time using the **Trace level** drop-down list on the session's trace window.

12

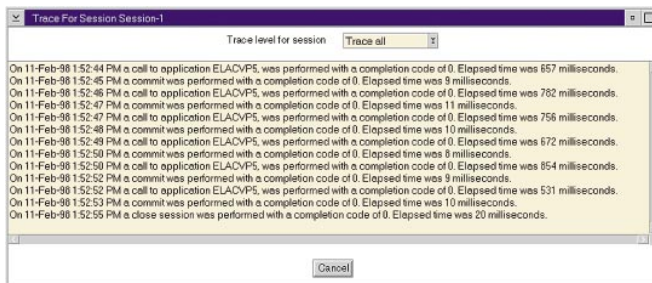


Figure 7. Session Trace Window

You can have UnitOfWorkServer requests logged using the **Log level** drop-down list and the *Log file name* entry field. Select **No log** to log no requests. Select **Log errors only** to log only requests resulting in errors. Select **Log all** to log all requests. You can specify the name of the log file, or let it default to CSOJAVA.LOG in the directory where the UnitOfWorkServer was started.

Changes made to the settings window are not applied until you click on the OK push button.

Tracing Active Sessions

When developing your Java classes it is convenient to see all activity on the web server for your session. The *AppletUnitOfWork* class is provided in the VisualAge Generator Java support package (ibm.cso) to establish a communication session to a server program from a Java applet. The first time you call a server program using a given *AppletUnitOfWork*, the communications session is established. After the first call, if you click on the **Refresh** push button of the session list, you should see your session in the list of active sessions. Session names default to Session-x, where x is a number generated by the session manager. You can name your session to make it more recognizable. You do this when testing your Java applet by using an *AppletUnitOfWork* constructor that includes a session name in its signature.

To open a session trace window like that shown in Figure 7, select your session in the session list, click on **Selected** in the menu bar, then click on the **Trace** menu item. The resulting trace window then displays all requests of that session that meet the trace level criteria. Information about each request includes the time of the request, the type of request, the return code, and the response time for the request. If there were any errors, the error message is added.

Sometimes it is convenient to have the trace window open automatically each time a session is started to indicate whether the web server is connected, or to include the first call in the trace. To do this, use the **Trace new sessions** option as described in the section "Changing Session Manager Settings" earlier in this article.

Explicitly Closing a Session

When developing Java applets, you might make a mistake that causes the applet to terminate abnormally after the communications session is established, and before it is closed. End-users running an applet also might go on to other work without ending the applet. This causes the communications session to remain open on the web server. If you have the timeout interval set, the session manager will eventually close the session because of inactivity. You can close the session immediately by selecting the session in the session list, clicking on **Selected** in the menu bar, then clicking on the **Close** menu item.

Bringing Trace Windows to the Forefront

When you have a lot of trace windows open, you will need to bring a hidden window to the forefront of your desktop. To do this, click on **Windows** in the menu bar, then click on the menu item for the session name of the trace window you want brought to the forefront.

Summary

In general, the new session manager provides:

- Trace facilities that aid in debugging Java applets that call VisualAge Generator server programs
- Response time statistics that help you monitor the performance of the web server
- The ability to monitor the activity of a session on the web server and to cancel a session immediately if desired
- A logging facility that helps in tracking problems that occurred with sessions established on the web server



VisualAge Generator Web Pages

The VisualAge Generator web address is:

www.software.ibm.com/ad/visgen

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

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

Interoperability—Calling 3GL Programs

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

VisualAge Generator provides application programmers a robust language to develop their programs with. There are times when VisualAge Generator programs must interface with legacy applications written in another programming language, usually a 3GL language such as C, COBOL, or PL/I. This article demonstrates how to call 3GL programs from VisualAge Generator programs and call VisualAge Generator programs from 3GL programs. Although VisualAge Generator V3.0 supports many different environments, this article focuses on the environments where C++ code is generated: AIX, HP-UX, OS/2, and Windows NT.

This is the second in a series of articles that will cover the interoperability between VisualAge Generator and programs written in 3GL languages. Follow-on articles will cover advanced debugging, and

conversion between host data files and the various workstation file formats.

The VisualAge Generator language has three ways of transferring control to another program: CALL, DXFR, and XFER. The examples used in this article all use the CALL interface to transfer control to the 3GL program. The CALL interface transfers control to the target program and returns control to the calling program. The DXFR and XFER interfaces transfer control to the target program but do not return control to the calling program.

Each of the following sections will discuss considerations unique to each environment and present examples for several different scenarios. A summary of the examples is presented in the following table. The source code for each example is located on the

following VisualAge Generator ftp site. Its URL is:

<ftp://ps.software.ibm.com/ps/products/visualagegen/info/v30>

The examples can be found in the following files:

- AIX** vagexaix.tar
- HP** vagexhp.tar
- OS/2** vagexos2.zip
- NT** vagexnt.zip

Instructions for compiling, linking, and executing the examples are included in a readme file with the source code.

The AIX Environment

The VisualAge Generator Server for AIX Version 3.0 product provides the runtime support necessary to execute VisualAge Generator programs on the AIX platform. It requires AIX Version 4.1.4 or later. AIX runtime support was also provided in VisualAge Generator Version 2.2. It required AIX version 3.2.5 or later.

VisualAge Generator programs on AIX are subject to the following considerations:

- VisualAge Generator programs on AIX must be compiled and linked into shared libraries. The script file generated for each VisualAge Generator program does this automatically.
- If VisualAge Generator programs are called by a 3GL program and parameters are being passed, the VisualAge Generator program must be a CALLED BATCH type of program. If no parameters are being passed, the VisualAge Generator program can be a MAIN BATCH program.

Example	Description
	AIX
1	AIX C calling VA/G called program, dynamic loading
2	AIX VA/G program calling C, dynamic loading
3	AIX VA/G program calling script file via RUNKSH
4	AIX VA/G program calling IBM COBOL shared library
	HP-UX
5	HP-UX C calling VA/G program, dynamic loading
6	HP-UX VA/G program calling C, dynamic loading
	OS/2
7	OS/2 C calling VA/G called program, dynamic loading
8	OS/2 VA/G program calling C, dynamic loading
9	OS/2 VA/G program calling REXX via RUNCMD
10	OS/2 VA/G program calling IBM COBOL
11	OS/2 VA/G program calling PL/I
	Windows NT
12	NT 'C' calling VA/G called program, dynamic loading
13	NT VA/G program calling C, dynamic loading
14	NT VA/G program calling BAT file via RUNBAT

- VisualAge Generator programs on AIX can call only programs (3GL or otherwise) that are compiled and linked into shared libraries. They cannot directly call script files or executable files directly. These types of programs must be called through a stub program that is compiled and linked as a shared library. See example 3.
- When calling a script file via RUNKSH from a VisualAge Generator program (as shown in example 3), only character parameters can be passed to the script file.
- When calling programs written in COBOL or any other 3GL language (except C++) on AIX, you must specify a linkage table at generation time that contains the LINKTYPE=AIXLOAD parameter. This causes the VisualAge Generator runtime to use the AIX subroutine **load** instead of **loadAndInit** to dynamically load the program. The **loadAndInit** subroutine is used to load C++ programs. See example 4 for a sample linkage table.

Examples on AIX

The examples in this section were compiled and executed on an AIX Version 4.1.5 system. The C examples were compiled using the IBM C Set ++ for AIX compiler Version 3.1.4. The COBOL example was compiled using the IBM COBOL Set for AIX compiler Version 1.1.

Example 1

In this example, a C program is calling a VisualAge Generator Called program. The VisualAge Generator program is dynamically loaded by the C program using the AIX routine **loadAndInit**. To be consistent with the behavior of VisualAge Generator Server programs running on AIX, the C

program is coded so that the environment variable FCWLIBPATH must be set to contain the directories that will be searched to locate the VisualAge Generator program (that is, export FCWLIBPATH=/home/aixuser1/genout). A structure (working storage record) containing most of the VisualAge Generator data types is passed to the VisualAge Generator program. The VisualAge Generator program initializes each data item, sets a return code, and returns to the C program. The C program formats and displays the value of each data item.

Example 2

In this example, a VisualAge Generator Main program is calling a C program. The C program is dynamically loaded by the VisualAge Generator Server. The C program is located using the contents of the FCWLIBPATH environment variable. This example passes each data item to the C program individually instead of passing a structure. The VisualAge Generator program initializes each data item. The C program formats and displays the value of each data item.

Example 3

In this example, a VisualAge Generator Main program is calling a Korn shell script file. Since a VisualAge Generator program cannot directly call script files or executables, an intermediate C program (RUNKSH) must be called. The name of the script file and any parameters that the script file needs are passed to RUNKSH. RUNKSH then invokes the script file. In this case, one parameter is passed. Only character parameters may be passed using this technique.

Example 4

In this example, a VisualAge Generator Main program is calling a COBOL program. The COBOL program is dynamically loaded by the VisualAge Generator Server.

The COBOL program is located using the contents of the FCWLIBPATH environment variable. This example individually passes each data item to the COBOL program instead of passing a structure. The VisualAge Generator program initializes each data item. The COBOL program formats and displays the value of each data item in a file called AVG2B1B.dat.

The HP-UX Environment

The VisualAge Generator Server for HP-UX Version 3 product provides the runtime support necessary to execute VisualAge Generator programs on the HP-UX platform. It requires HP-UX Version 10.10 or later and HP's Advanced C++ compiler (aCC).

VisualAge Generator programs on HP-UX are subject to the following considerations:

- VisualAge Generator programs on HP-UX must be compiled and linked into shared libraries. The script file that is generated for each VisualAge Generator program does this automatically.
- If they are called by a 3GL program and parameters are being passed, the VisualAge Generator program must be a CALLED BATCH program. If no parameters are being passed, the VisualAge Generator program can be a MAIN BATCH program.
- VisualAge Generator programs on HP-UX can call only programs (3GL or otherwise) that are compiled and linked into shared libraries. They cannot call script files or executable files directly. These types of programs must be called through a stub program that is compiled and linked as a shared library. See example 3; the HP-UX environment is very similar to AIX.

- When calling a script file via RUNKSH from a VisualAge Generator program, only character parameters can be passed to the script file.
- The C++ compiler on HP-UX does not have an option to pack structures. Therefore, when you pass data to 3GL programs using structures, ensure that numeric data items are properly aligned; otherwise, bus errors will occur. Bus errors are synonymous with data exceptions. Character data items are not a problem because they are aligned on a byte boundary; however, the 2-, 4-, and 8-byte integer data items (short, int, double in C) might need pad bytes inserted to ensure that each data type is aligned properly. For more information on passing data to 3GL programs, refer to the technical report *VisualAge Generator Interoperability—Passing Data to 3GL Programs*.

Examples on HP-UX

The examples in this section were compiled and executed on an HP-UX Version 10.20 system using the HP aCC compiler Version A.01.00.

Example 5

In this example, a C program is calling a VisualAge Generator Called program. The VisualAge Generator program is dynamically loaded by the C program using the HP-UX routine `shl_load`. To be consistent with the behavior of VisualAge Generator Server programs running on HP-UX, you must set the environment variable `SHLIB_PATH` to contain the directories of the VisualAge Generator program as well as the VisualAge Server runtime modules (that is, export `SHLIB_PATH=/opt/vgws22/lib:/home/hpuser1/genout`). A structure (working storage record) containing most of the VisualAge Generator data types

is passed to the VisualAge Generator program. The VisualAge Generator program initializes each data item, sets a return code, and returns to the C program. The C program formats and displays the value of each data item.

Example 6

In this example, a VisualAge Generator Main program is calling a C program. The C program is dynamically loaded by the VisualAge Generator Server runtime. The C program is located using the contents of the `SHLIB_PATH` environment variable. This example passes each data item to the C program individually instead of passing a structure. The VisualAge Generator program initializes each data item. The C program formats and displays the value of each data item.

The OS/2 Environment

VisualAge Generator Server for OS/2 Version 3.0 provides the runtime support necessary to execute VisualAge Generator programs on the OS/2 platform. It requires OS/2 Warp Version 3.0 or later. OS/2 runtime support was also provided in VisualAge Generator Version 2.2. It also required OS/2 Warp version 3.0 or later.

VisualAge Generator programs on OS/2 are subject to the following considerations:

- VisualAge Generator programs on OS/2 must be compiled and linked into dynamic link libraries (dlls). The command file that is generated for each VisualAge Generator program does this automatically.
- If VisualAge Generator programs are called by a 3GL program and parameters are being passed, the VisualAge Generator program must be a CALLED BATCH program. If no parameters are being passed, the VisualAge Generator program can be a MAIN

BATCH program.

- VisualAge Generator programs on OS/2 can call only programs (3GL or otherwise) that are compiled and linked into dynamic link libraries. They cannot call command files or executable (EXE) files directly. These types of programs must be called through a stub program that is compiled and linked as a dynamic link library. See example 9.
- When calling a command file via RUNCMD from a VisualAge Generator program (see example 9), only character parameters can be passed to the command file.

Examples on OS/2

The examples in this section were compiled and executed on an OS/2 Warp Version 3.0 system. The C examples used the IBM VisualAge C++ for OS/2 compiler Version 3. The COBOL examples used the IBM VisualAge COBOL compiler Version 2.0. The PL/I examples used the IBM PL/I for OS/2 compiler Version 1.1.

Example 7

In this example, a C program is calling a VisualAge Generator CALLED program. The VisualAge Generator program is dynamically loaded by the C program using the OS/2 routine `DosLoadModule`. Its entry point is then obtained by calling the OS/2 routine `DosQueryProcAddr`. To be consistent with the behavior of VisualAge Generator Server programs running on OS/2, the environment variable `LIBPATH` must be set to contain the directories that will be searched to locate the VisualAge Generator program. A structure (working storage record) containing most of the VisualAge Generator data types is passed to the VisualAge Generator program. The VisualAge Generator program initializes each

data item, sets a return code, and returns to the C program. The C program formats and displays the value of each data item.

Example 8

In this example, a VisualAge Generator Main program is calling a C program. The C program is dynamically loaded by the VisualAge Generator Server runtime. The C program is located using the contents of the LIBPATH environment variable. This example passes each data item to the C program individually instead of passing a structure. The VisualAge Generator program initializes each data item. The C program formats and displays the value of each data item.

Example 9

In this example, a VisualAge Generator Main program is calling a REXX command file. Since a VisualAge Generator program cannot directly call command files or executables, an intermediate C program (RUNCMD) must be called. The name of the script file and any parameters that the script file needs are passed to RUNCMD, which in turn invokes the script file. In this case, one parameter is passed. Only character parameters may be passed using this technique.

Example 10

In this example, a VisualAge Generator Main program is calling a COBOL program. The COBOL program is dynamically loaded by the VisualAge Generator Server runtime. The COBOL program is located using the contents of the LIBPATH environment variable. This example passes each data item to the COBOL program individually instead of passing a structure. The COBOL program formats and displays the value of each data item in a file called OVG2B1B.dat.

Example 11

In this example, a VisualAge Generator Main program is calling a PL/I program. The PL/I program is dynamically loaded by the VisualAge Generator Server runtime. The PL/I program is located using the contents of the LIBPATH environment variable. This example passes each data item to the PL/I program individually instead of passing a structure. The PL/I program formats and displays the value of each data item in a file called OVG2P1P.dat.

The Windows NT Environment

The VisualAge Generator Server for Windows NT Version 3.0 provides the runtime support necessary to run VisualAge Generator programs on the Windows NT platform. It requires Windows NT Version 3.51 or 4.0. Windows NT runtime support was also provided in VisualAge Generator Version 2.2. It also required Windows NT version 3.51 or 4.0.

VisualAge Generator programs on Windows NT are subject to the following considerations:

- VisualAge Generator programs on Windows NT must be compiled and linked into dynamic link libraries (dlls). The batch file that is generated for each VisualAge Generator program does this automatically.
- If VisualAge Generator programs are called by a 3GL program and parameters are being passed, the VisualAge Generator program must be a CALLED BATCH program. If no parameters are being passed, the VisualAge Generator program can be a MAIN BATCH program.

- VisualAge Generator programs on Windows NT can call only programs (3GL or otherwise) that are compiled and linked into dynamic link libraries. They cannot call batch files or executable (EXE) files directly. These types of programs must be called through a stub program that is compiled and linked as a dynamic link library. See example 14.
- When calling a script file via RUNBAT from a VisualAge Generator program (example 14), only character parameters can be passed to the batch file.

Examples on Windows NT

The examples in this section were compiled and executed on an Windows NT Version 4.0 system using the IBM VisualAge C++ for Windows NT compiler Version 3.5.

Example 12

In this example, a C program is calling a VisualAge Generator CALLED program. The VisualAge Generator program is dynamically loaded by the C program using the Windows NT routine LoadLibrary. Its entry point is then obtained by calling the Windows NT routine GetProcAddress. To be consistent with the behavior of VisualAge Generator Server programs running on Windows NT, the environment variable PATH must be set to contain the directories that will be searched to locate the VisualAge Generator program. A structure (working storage record) containing most of the VisualAge Generator data types is passed to the VisualAge Generator program. The VisualAge Generator program initializes each data item, sets a return code, and returns to the C program. The C program formats and displays the value of each data item.

Example 13

In this example, a VisualAge Generator Main program is calling a C program. The C program is dynamically loaded by the VisualAge Generator Server runtime. The C program is located using the contents of the PATH environment variable. This example passes each data item to the C program individually instead of passing a structure. The VisualAge Generator program initializes each data item. The C program formats and displays the value of each data item.

Example 14

In this example, a VisualAge Generator Main program is calling a BAT command file. Since a VisualAge Generator Main program cannot directly call command files or executables, an intermediate C program (RUNBAT) must be called. The name of the command file and any parameters that the command file needs are passed to RUNBAT. RUNBAT invokes the command file. In this case, one parameter is passed. Only character parameters can be passed using this technique.

Summary

Interfacing VisualAge Generator programs with programs written in one of the 3GL languages is often a necessity in today's complex system environments. In some cases, it involves interfacing with legacy applications and other times it might be necessary to implement an algorithm in a 3GL because of performance reasons. Using the information contained in this article, an application programmer can easily implement the changes necessary to interface their VisualAge Generator programs with new or existing 3GL programs.

VisualAge Generator Calendar of Events

<i>June 10–11</i>	Second VisualAge Generator International Symposium	<i>Stuttgart, Germany</i>
<i>July 19–21</i>	Australasian GUIDE	<i>Melbourne, Australia</i>
<i>July 23–24</i>	Singapore SHARE and GUIDE	<i>Singapore</i>
<i>July 27–28</i>	Malaysian SHARE and GUIDE	<i>Malaysia</i>

Introduction to Automated Testing of VisualAge Generator Clients with SilverMark's Smalltalk Test Mentor

By Michael Silverstein, SilverMark, Inc. and John Casey, VisualAge Generator Sales Support

Large enterprises are adopting VisualAge Generator on a grand scale as the development environment of choice for building business-critical, multi-tier client/server applications. As companies increase their reliance on these applications for their day-to-day operations, reliability, availability, and performance become increasingly critical.

Defects can creep in at any time during code development or maintenance. Seemingly trivial changes to complex applications can have far-reaching consequences. Failure to adequately ensure quality often shows up in headlines, causing embarrassment and financial hardships to companies that have not adequately tested their applications prior to deployment. The later a defect is discovered, the greater the economic penalty it exacts in the form of increased support cost and customer dissatisfaction.

Given this, it is clear that testing is an important piece of the software development puzzle. In order to adequately test an application, tests should be created that exercise paths within the context of many of the varied states that the application may be in. Writing test scripts to be executed manually is usually the starting point for formalized testing, but with frequent development cycles, repeated manual testing quickly becomes the dominant consumer of valuable human resources.

Automated Testing

The answer to this problem is to move effort expenditure from repetitious manual testing to an up-front investment in automation that can be amortized over the development and maintenance lifetime of an application. This encourages frequent testing throughout the application life cycle so defects are caught early when they are less costly to fix.

Automated testing implies adoption of a test automation tool. An ideal test automation tool should automate test case creation, strongly encourage the use of reusable test components, and be designed specifically with an application's operating and development environment in mind.

In Version 3.0 of VisualAge Generator, the underlying Smalltalk development environment was exposed, providing exciting opportunities for creating robust, GUI-rich intelligent clients. This has also opened up the

opportunity to use tools and add-ons previously available only to VisualAge for Smalltalk developers. The Smalltalk Test Mentor by SilverMark, Inc. (<http://www.silvermark.com>) is an add-on designed specifically for testing VisualAge for Smalltalk-based applications. As such, it is ideally suited for testing VisualAge Generator client applications.

This article introduces the Smalltalk Test Mentor and how it can be applied to testing VisualAge Generator GUI applications. In a later article we will discuss other aspects of testing VisualAge Generator client applications, including testing business objects and reusing test cases.

Testing the Smalltalk Client

One of the strengths of VisualAge for Smalltalk is that it provides a rich set of graphical user interface (GUI) components and that these GUI components may be subclassed and aggregated to form new, specialized GUI components. Unfortunately, this strength presents a serious challenge to traditional automated GUI testing tools. From outside of the Smalltalk image, many of the most useful components, such as notebooks and containers are only recognized as simple drawing areas to these testing tools. At best, they can record and play back mouse clicks over coordinates within the UI components.

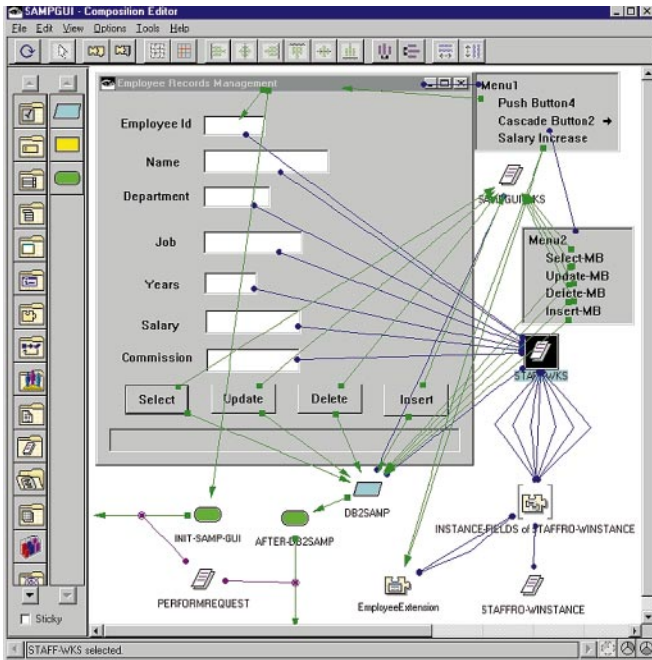
Because the Smalltalk Test Mentor records and plays back GUI interactions from within the Smalltalk image, it has direct knowledge and access to the state of all GUI components. Any GUI interactions can be recorded and played back, and GUI component state (for example, selected items in a container) can be verified.

The other advantage of testing within the Smalltalk image is that the testing tool has access to all objects, both visual and nonvisual. This means that client-side business logic and domain objects can be tested along with the user interface.

A Simple Example

The best way to illustrate is by example. In this example, we will create a simple automated test for the sample GUI view, **SAMPGUI**, shown below, that performs maintenance on rows in the *staff* table of the DB2 sample database.

For this example, we have chosen a use-case-driven approach to testing, where we will exercise the application under test with a real-world scenario. The scenario we have chosen retrieves a particular employee's record. As part of the test, we will verify that the expected employee information was retrieved.



Test Case Structure

Tests created using the Smalltalk Test Mentor are structured in terms of *suites*, *scenarios* and *steps*. A suite is a set of related scenarios. A scenario corresponds to a particular usage of the application under test, or use-case. Scenarios are composed of steps. Each step performs a discrete action on the application under test. There are several different types of steps to choose from. Step types include some that execute scripts, a type of step that prompts for operator intervention, one that iterates over test data in a file, and steps that reference reusable test components. In this example, we will concern ourselves mostly with those related to recording and playing back GUI interactions.

Recording a Test

Ideally a test tool should automate the task of test creation. The Smalltalk Test Mentor does this by recording your interactions with the application and generating test steps. To record a test, you simply add a UI recording step for some set of interactions to record, press the **record** button, and then start using your application. Each recorded step is generated as a short, Smalltalk script with a comment to make it easier to read the flow of interactions. The generated scripts follow a simple, consistent pattern in the form of visual part retrieval followed by an action to be performed on

it. For example:

(ActiveWindow widget: 'Push Button1') click clicks on 'Push Button1' in the window that currently has focus.

Verifying Application State

Exercising your application will certainly flush out hard exceptions, but you should also verify that your application's state is correct. You can verify the application's state through its GUI by comparing the contents of visual parts to a 'gold standard'. The Smalltalk Test Mentor enables you to do this by providing a visual part verification tool that enables you to click on a part in your running view and select from a list of attributes to be verified. Once you do this, the verification code is automatically generated as steps and added to the test scenario.

The Completed Test

The following shows the test editor with all of the steps for this scenario:

Step	Name	Description	Type	Iterati
	SampguiTestSuite	Sample VA Generator GUI-based tests	Suite	1
	Simple HR Query		Scenario	1
	Open view	Open the sample employee HR view.	Script	1
	Simple Query		UI recording	1
	(1)	Set focus to <Employee Records Manager	UI script	1
	(2)	Change the value of <Text>	UI script	1
	(3)	Click on <Select>	UI script	1
	Verify Message Line	Verify that Sampgui-Msg labelString = ['Er	Verification	1
	Verify Salary	Verify that Text10 value = ['\$16,502.83']	Verification	1
	Verify Commission	Verify that Text12 value = ['\$1,152.00']	Verification	1
	Close view		UI recording	1
	(1)	Set focus to <Employee Records Manager	UI script	1
	(2)	Close <Employee Records Management>	UI script	1

The bottom panel shows 'Description of SampguiTestSuite' with 'Sample VA Generator GUI-based tests' selected. The 'SampguiTestSuite classification' is 'Regression Tests', 'Runnable', and 'Demo'. The 'SampguiTestSuite's implementation details' panel shows 'Invariant Selectors' and 'Applications Covered' as 'HRMStaffSubsystem'.

In the **Type** column you can see the different types of elements. At the top level is the suite `SampguiTestSuite`, which contains the single scenario `Simple HR Query`. This scenario contains three steps: `Open View`, `Simple Query`, and `Close View`. `Open view` specifies the launch code to start the view under test. The Smalltalk code for this is:

```
SAMPGUI new openWidget.
```

The Simple Query and Close view steps are UI recording steps that contain generated steps for the recorded UI interactions and verifications. We chose to break up the query and close operations into two UI recording steps simply for the sake of clarity.

The Smalltalk Test Mentor automatically measures code (Smalltalk method) coverage during execution, if you specify the applications covered by the test. In this case, we specified the **HRMStaffSubsystem** application.

Executing a Test and Viewing Results

You run tests either by pressing the **Run** button or, if you need fine control over execution, by opening the advanced runner view (not shown). The advanced runner view provides the ability to set breakpoints in your test and walk through step execution. When tests are run, all exceptions are logged and metrics like individual step execution timings and method coverage are measured.

The results of executing a test are displayed in the Test Results Browser (below) using the same structure as the test itself, with traffic lights to show whether the steps passed or failed.

Name	Run	Pass	Fail	Time	Covered [%]	Testable [%]	Methods [#]
SampguiTestSuite	9	9	0	23,684	0.30	0.46	328

Had there been any exceptions, they would have been indicated by red lights, along with detailed information about the nature of their failures, including the execution stack trace. Summarized metrics and the result returned by each step are shown on the bottom of the view.

You can store test results persistently for comparison with later results. The Smalltalk Test Mentor provides tools for quickly finding anomalies between test runs, including out-of-tolerance execution times. This can be especially useful for highlighting subtle performance problems.

You can view method coverage details with the Coverage Browser shown below:

Item	Name	Methods	% Covered	Testable
	SampguiTestSuite	328	0.30	217
	HRMStaffSubsystem	328	0.30	217
	SAMPGUI	3	33.33	3
	STAFFGN class	28	0.00	28

Methods: SAMPGUI>>#cxBuild2: SAMPGUI>>#cxBuild

Method Source: inst := Array new: 170. "Set our instance key."

Coverage is separated by suite, application, and class.

As you can see, only a small part of the HRMStaffSubsystem application, which happens to have about 20 classes, was covered by this one simple scenario. Clearly, more testing needs to be done before this application is deployed!

Conclusion

Testing is as important a part of the software development process as design or coding. If you omit testing, you risk delivering defective products, with the accompanying economic penalties. Because manual testing is so labor intensive, the only practical way to effectively test enterprise applications is by automating the testing effort.

Successful test automation rests largely on using a test automation tool that fits the development environment and automates test creation as well as execution. In this article, we presented the Smalltalk Test Mentor, a tool designed specifically for automated testing of Smalltalk-based applications. We then discussed how it can be applied to client-side GUI testing of applications created using VisualAge Generator V3.0. In a future article, we will discuss how to structure tests for reuse, how to test business objects, and how to use the metrics that the Smalltalk Test Mentor gathers to measure the health of your applications.

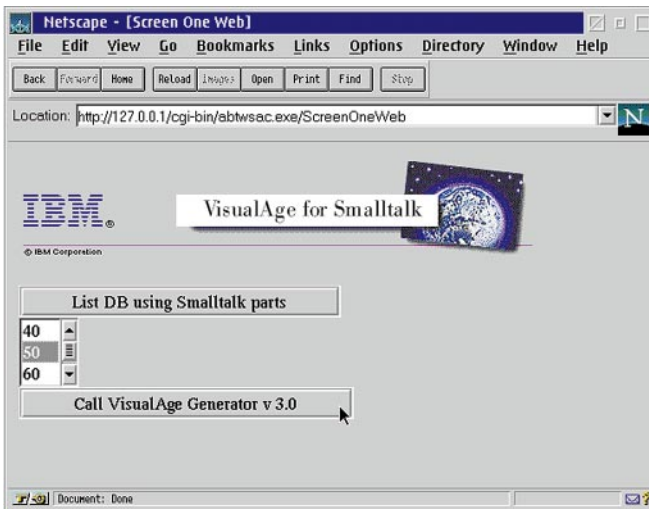
Moving your Application to the Internet

By Reginaldo W. Barosa—Certified AD Specialist, IBM Brasil

One easy way to move a VisualAge Generator application to the Internet is by using the VisualAge Smalltalk Web Connection Parts. The objective of this article is to show how to implement a sample application that accesses a DB2 database using the VisualAge database parts and then reads the details using a VisualAge Generator, *but does it on the Internet*.

The application we will build has two screens:

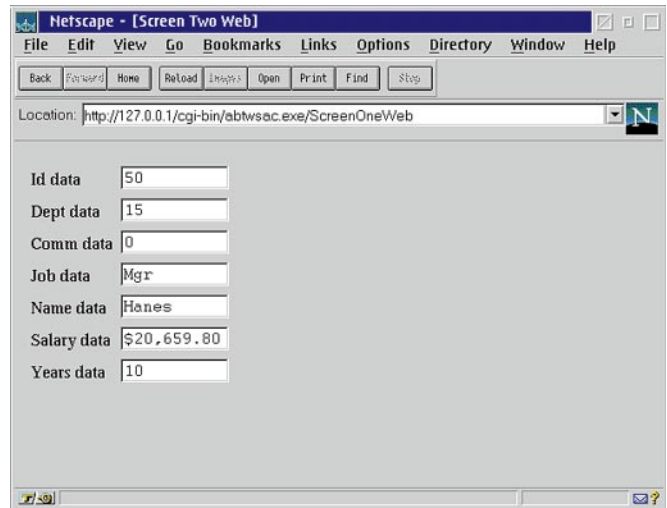
- The first screen shown in the figure below queries all the STAFF table when the **List DB using SMALLTALK parts** button is pressed. It is built using the VisualAge Smalltalk Database parts and Web Connection Parts.



Note that the query to DB2 is done on the client side (that runs in the Web Server).

- On the first screen, we selected the ID **50** and pressed the **Call VisualAge Generator v3.0** button.

The second screen, shown below, queries the details of the selected ID. This second screen was built using VisualAge Smalltalk Web Connection parts and VisualAge Generator. The VisualAge Generator application runs on the server, where the query is done (this could be on MVS/CICS).



The main activities to produce the screens shown above are:

1. Install the VisualAge Web Connection parts and load it in your Image
2. Create the first screen using VisualAge Database and Web Connection parts
3. Create a VisualAge Generator application and prepare to pass the selected ID to the second screen
4. Create the second screen using VisualAge Web Connection parts and a VisualAge Generator application.

1-Install VisualAge Web parts and load it at the image

This is not complex; just follow the installation procedure. After you have it loaded, the feature VisualAge Web Connection will appear as it is loaded in the image. See the figure below:



Also, the Web parts icons will be shown at the VisualAge Composition Editor:

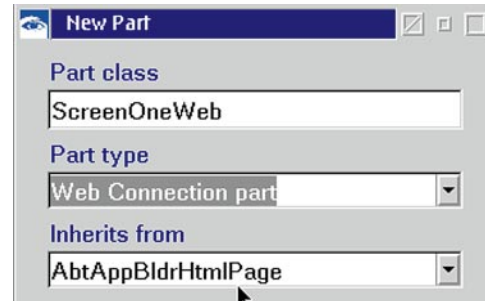


Now you're ready to start the coding.

2-Create the first screen using VisualAge Database and Web Connection parts

The activities here are:

1. Create an application and add a New Part named **ScreenOneWeb**. The Part type is **Web Connection part**. See below:



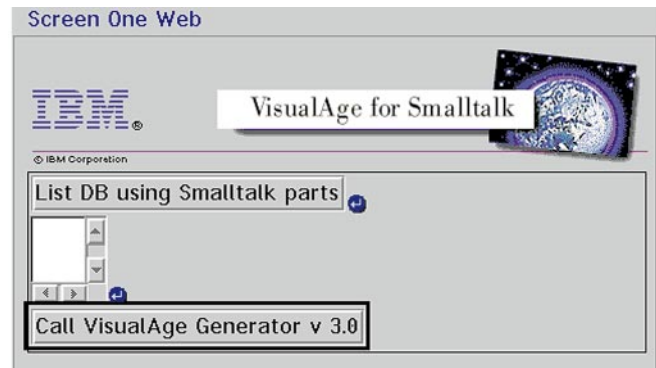
2. Using the categories **Web Connection** and **Web Form Parts**, add these visual elements: **Image**, **Form**, **2 Push Button**, **List** and **Line Breakers**.

The settings of the image should point to the gif to be shown. In our example, code

imageURL = /vamast.gif and
localFileName=vamast.gif.

Change the names and text of the buttons. Call them **pbListaDB** and **pbCallIVG**.

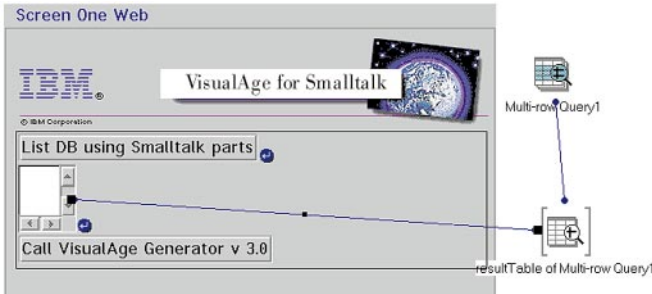
Change the name of the List to **efList**. See below:



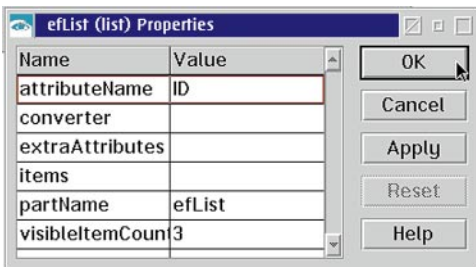
3. From the category **Database Functions**, add the **Multi-row query** part to the free form surface (white part). Alter its setting to perform a query to the STAFF table. In our example, the query will be:

```
SELECT STAFF.NAME, STAFF.ID, STAFF.JOB, STAFF.DEPT, STAFF.SALARY
FROM STAFF
ORDER BY STAFF.ID ASC
```

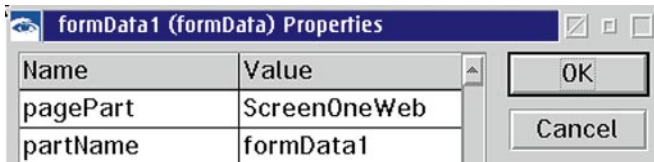
- Using the Multi-row, perform the Tear-Off attribute of **resultTable** and connect the **rows** attribute to the **items** attribute of efList (List):



- Change the **attributeName** of efList to get the column **ID** of the STAFF table. The settings will be:



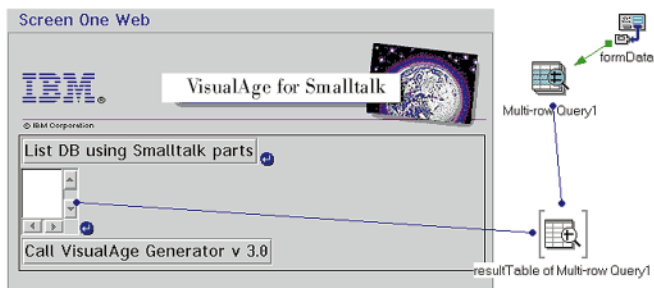
- To perform the database query, we must connect the List DB button to the Multi-row Query. From the category Web Connection add the part **FormData**. Open its settings and enter **ScreenOneWeb** in the field **pagePart**.



- Connect:

From	To
FormData:	Multi-rowQuery
event listADB clicked	action executeQueryAsTransaction

See below:



- Now we can test the application. But first be sure that the Web Connection Server is started, the Web Server Interface Monitor is on, and the Browser (Netscape) is active. If you are stand-alone, you must configure TCP/IP to use the loop-back; the local address is 127.0.0.1.

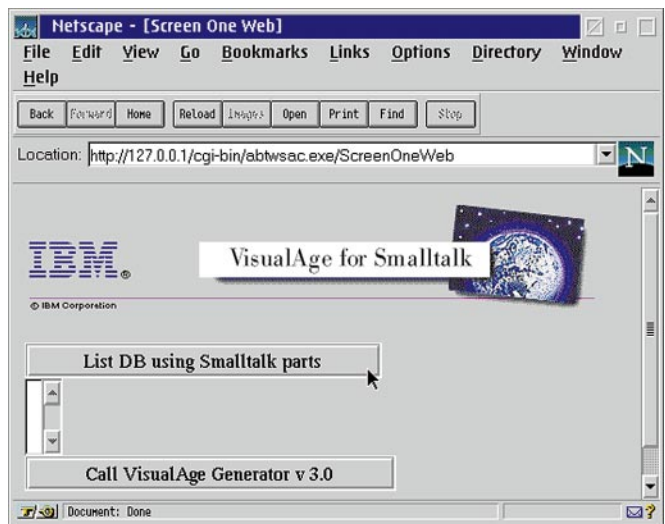
Also, all the setup necessary to have the Web parts working must be complete. See the Web Connection installation documentation (`\vast\webconn\server.txt`).

To start the Web Server Interface Monitor, use the Options/Open Web Server Interface Monitor.

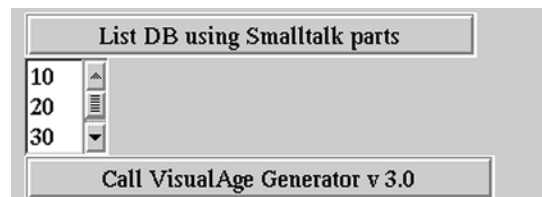
To start the application, enter the command:

`http://127.0.0.1/cgi-bin/abtwsac.exe/ScreenOneWeb`

See below the result of this input:



When the List DB button is pressed, the List field will have the IDs from the STAFF table:



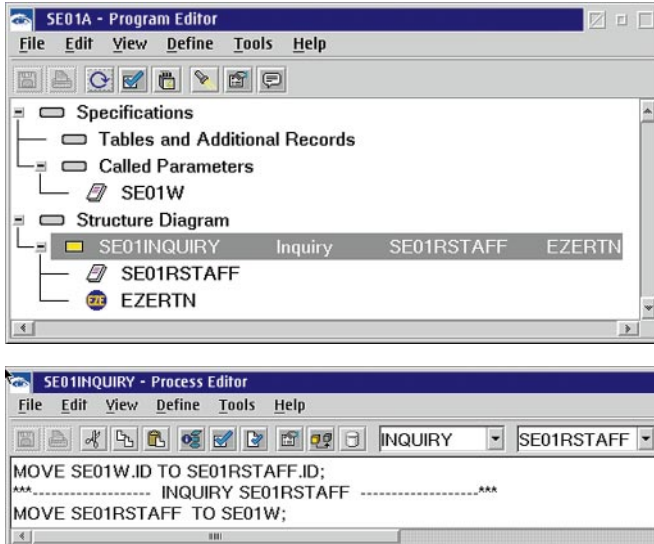
The next step is to select one ID (for example, 50) and press the second button (Call VisualAge Generator V3.0). Doing this will invoke a VisualAge Generator application that will be run on the server (this could be an MVS server) and send the details.

3—Create a VisualAge Generator application and prepare to pass the selected ID to the second screen.

Now let's play with the generator.

1. Create a VisualAge Generator application that will access the details of the ID received from the first screen. This application will be named **SE01A**. It is a called application and the logic is straightforward.

See below the graphical representation:



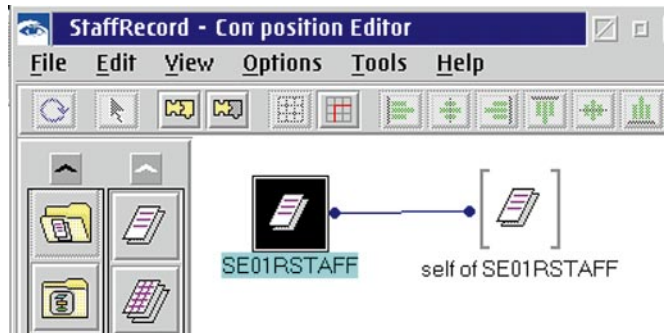
The SQL statements are:

```
SELECT
  ID, NAME, DEPT, JOB, YEARS, SALARY, COMM
  INTO
  :ID, :NAME, :DEPT, :JOB, :YEARS, :SALARY, :COMM
  FROM STAFF T1
  WHERE ID = :ID
```

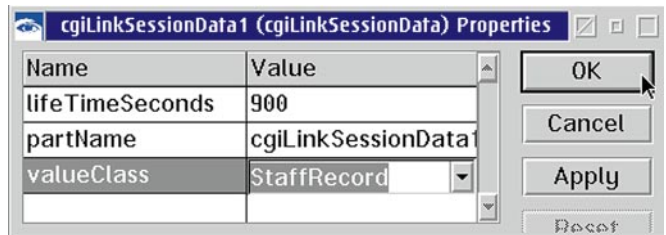
The VisualAge application is ready to be tested. Now let's go back to the Web stuff.

To keep the data entered in the Internet screens persistent in the session (lifetime default is 900 seconds), I recommend creating a nonvisual class that will hold the data across the Internet screens. This is optional but useful in applications with multiple screens. The steps to be performed are:

2. Create a nonvisual class that will hold the data. Create a new nonvisual part named **StaffRecord**. Add the part VAGen Record Part, created in the previous step, named **SE01RSTAFF**. Perform a tear-off attribute of self and promote the attribute **selfOfSE01RSTAFFID** data, since the **ID** must be used in future connection. This nonvisual part will be used as an object that will store the data across the screens on the Internet. See below:



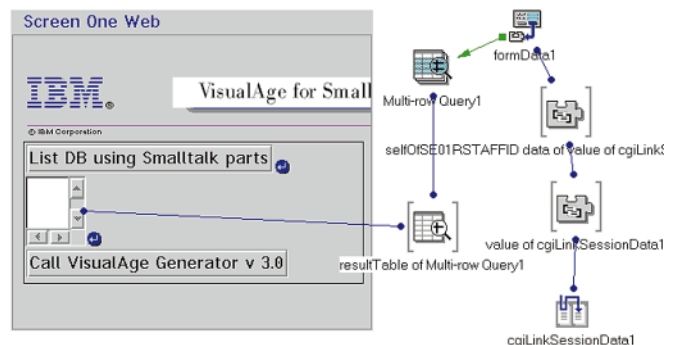
3. From Web Connection category add the part **cgiLinkSession Data**. Change its properties, adding **StaffRecord** (built in the previous step) in the field **valueClass**. See below:



The value 900 (seconds) is the time this data will be alive when the Internet session is initiated. If the Internet user will take longer than 900 seconds to interact with the Browser, it could be increased.

4. From the cgiLinkSessionData, perform a tear-off of **value (StaffRecord)**. Perform another tear-off of **self StaffRecord** and connect:

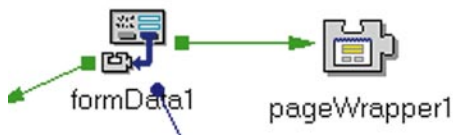
See the Composition Editor now:



At this point the selected ID is kept in the `cgiLinkSessionData`. It is on the instance of the class `StaffRecord`, attribute `ID` data.

When we need the data between screens, we just drop this part in the class that needs such information. This is one of the nice features of these Web Parts.

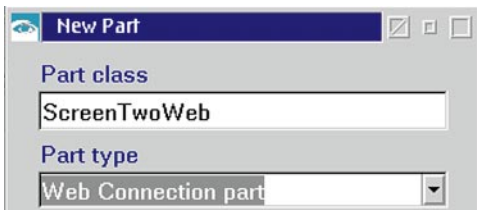
- Now we have to make an action for the button that Call `VisualAge`. We will add a part named **Page Wrapper** from Category `Web Connection`. Open its settings and add to the field `pagePart` the second web page named **ScreenTwoWeb**.
- Connect from `formData` the event **pbCallVg clicked** to action **transferRequest** of `pageWrapper`:



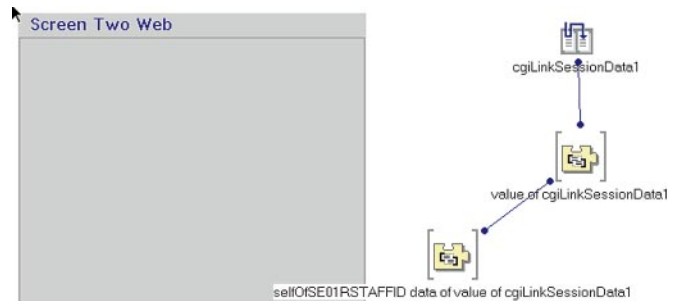
4—Create the second screen using *VisualAge Web Connection parts and a VisualAge Generator application.*

Now we have to create the second web page. We will name it **ScreenTwoWeb**.

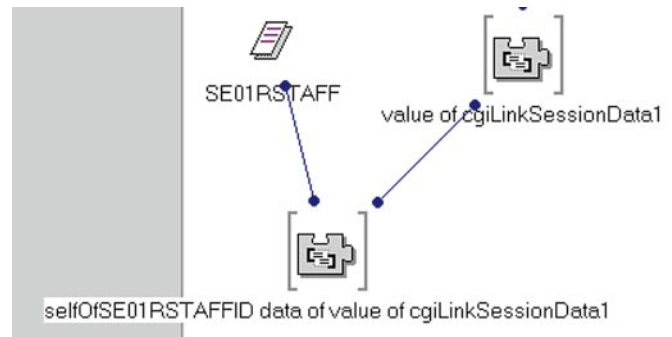
- Create a new Part named **ScreenTwoWeb**. It must be **Web Connection part** in *Part Type*. See below:



- Go to the `ScreenOneWeb` part you did before and make a **copy** of the **cgiSessionLinkData** and **paste** it on this new Web part. Also tear off the attributes as shown below. The ID is going to pass from `ScreenOneWeb` to this new screen, since it is kept in that instance.



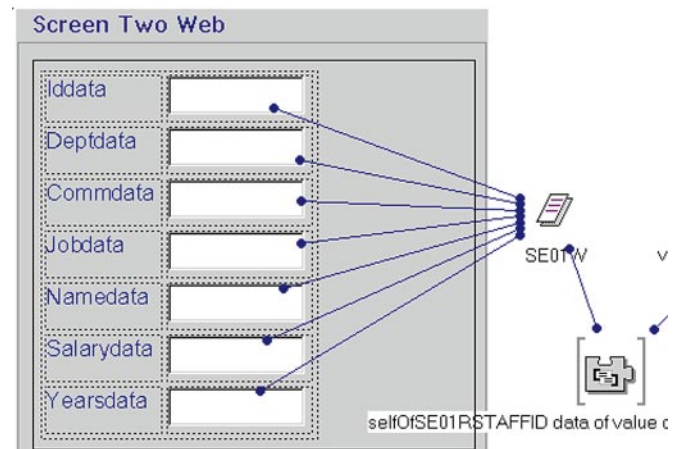
- Add the **VAGen Record Part** named **SE01RSTAFF** you did before. Connect from `SE01RStaff` attribute **ID data** to **self** of `selfOfSE01RSTAFFID data`:



- Add a **Form** part to the page:



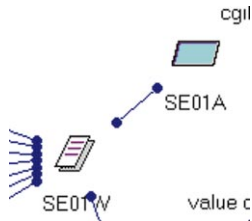
- Click mouse button 2 to the part `SE01W` and select **Quick HTML...** and **self**. Many entry fields will be generated. Do a cleanup and delete the ones without the word `data` in their names:



- Using the category VAGen Logic Parts, add the part **VAGen Callable Functions** and select the application **SE01A** you built before.

Click mouse button 2 on this part and select **Build parameters from definition** to generate the attributes to pass the data.

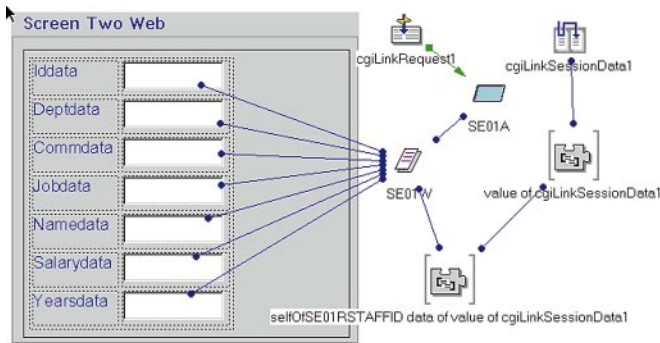
This application will receive the ID from the record SE01W and will return the other data. So you must connect from SE01A attribute **SE01W** to the attribute **self** of SE01W:



- Add the part **CGI Link Request** from category *Web Connection*. This part will send the event *requestReceived* when this page is shown on the Internet. We then have to connect from *cgiLinkRequest* the event **requestReceived** to the action **execute** of SE01A:

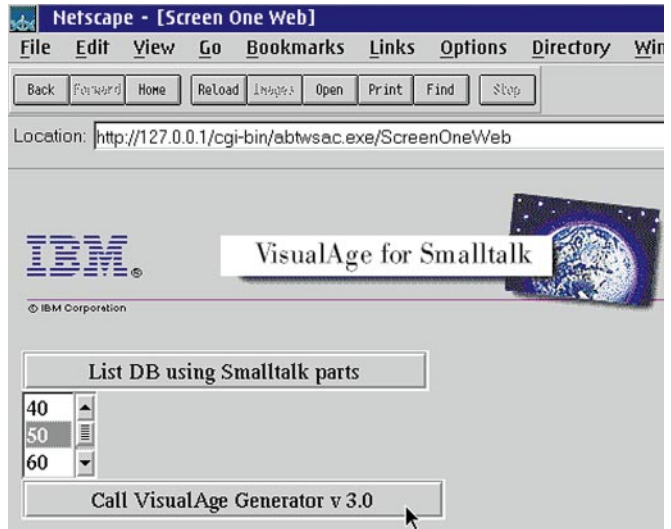
cgiLinkRequest1	SE01A
Event	Action
requestReceived	execute

The final connections will be:

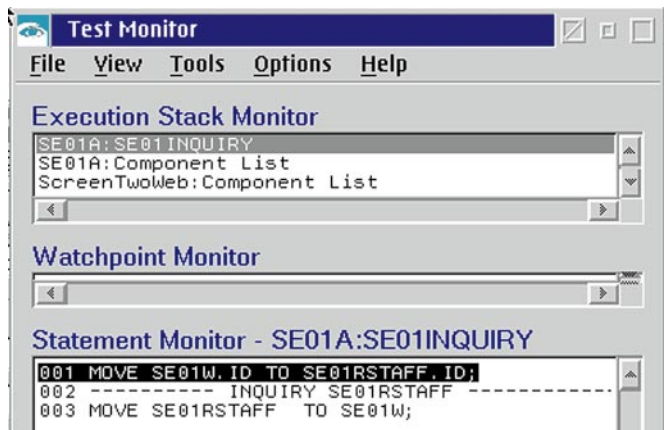


- Change the converter of the field salary from **Generic Converter** to **Decimal Amount**.

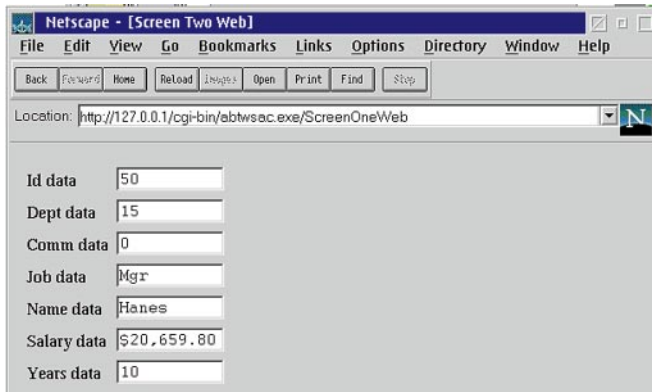
- It is time to test now. The first screen is shown. The user presses the **List DB** button and gets all the IDs in the List. The user selects the number **50** and presses the **Call VisualAge Generator v 3.0** button:



The second screen is showing and the *Test Monitor* is active, since the application SE01A is executed as soon the request is received. The VisualAge Test Monitor that is started:



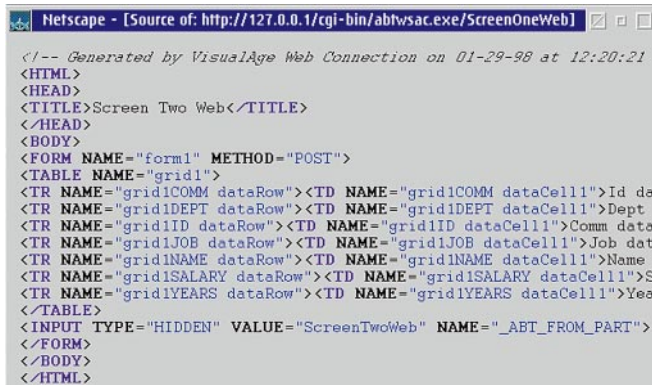
The second Web screen is below:



The screenshot shows a Netscape browser window titled "Netscape - [Screen Two Web]". The address bar contains "http://127.0.0.1/cgi-bin/abtwsac.exe/ScreenOneWeb". The main content area displays a form with the following fields:

Id data	<input type="text" value="50"/>
Dept data	<input type="text" value="15"/>
Comm data	<input type="text" value="0"/>
Job data	<input type="text" value="Mgr"/>
Name data	<input type="text" value="Henes"/>
Salary data	<input type="text" value="\$20,659.80"/>
Years data	<input type="text" value="10"/>

See below the HTML generated:



```
<!-- Generated by VisualAge Web Connection on 01-29-98 at 12:20:21
<HTML>
<HEAD>
<TITLE>Screen Two Web</TITLE>
</HEAD>
<BODY>
<FORM NAME="form1" METHOD="POST">
<TABLE NAME="grid1">
<TR NAME="grid1COMM dataRow"><TD NAME="grid1COMM dataCell1">Id da
<TR NAME="grid1DEPT dataRow"><TD NAME="grid1DEPT dataCell1">Dept
<TR NAME="grid1ID dataRow"><TD NAME="grid1ID dataCell1">Comm data
<TR NAME="grid1JOB dataRow"><TD NAME="grid1JOB dataCell1">Job dat
<TR NAME="grid1NAME dataRow"><TD NAME="grid1NAME dataCell1">Name
<TR NAME="grid1SALARY dataRow"><TD NAME="grid1SALARY dataCell1">S
<TR NAME="grid1YEARS dataRow"><TD NAME="grid1YEARS dataCell1">Yea
</TABLE>
<INPUT TYPE="HIDDEN" VALUE="ScreenTwoWeb" NAME="_ABT_FROM_PART">
</FORM>
</BODY>
</HTML>
```

Conclusion

It is very easy to connect existing VisualAge Generator Server applications to the Web Browser. That could be a nice option if users don't want to write Java clients or if the solution must be HTML. There are situations where Java is not acceptable since it requires JDK 1.1 and some browsers still don't support it.

Comment Form

Please check any appropriate boxes:

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

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

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

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

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

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



Cut or
Fold Along
Line

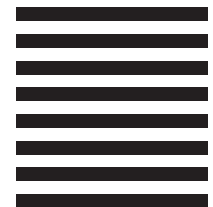
Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines
The VisualAge Generator Newsletter
Newsletter Editor
TF6B/062/J125
P.O. Box 12195
RTP, NC 27709-2195
USA



Fold and Tape

Please do not staple

Fold and Tape

Cut or
Fold Along
Line

A Question From Us To You

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

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. TF6B/062
P.O. Box 12195
3039 Cornwallis Road
RTP, NC 27709-2195
USA
FAX: (919) 254-0206

© Copyright International Business Machines Corporation 1997. 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, C Set ++, CICS, CICS OS2, Database 2, DB2, DB2/2, IBM, IMS, MQSeries, MVS, VSE, Operating System/2, OS/2, OS/400, VisualAge, and VisualGen.

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

Java and JavaBeans are trademarks of Sun Microsystems, Inc.

Lotus Notes is a trademark or registered trademark of Lotus Development 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, 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.

The VisualAge Generator Newsletter
IBM Corporation
Dept. TF6B/062
P.O. Box 12195
3039 Cornwallis Road
RTP, NC 27709-2195
USA

G242-0315-08

