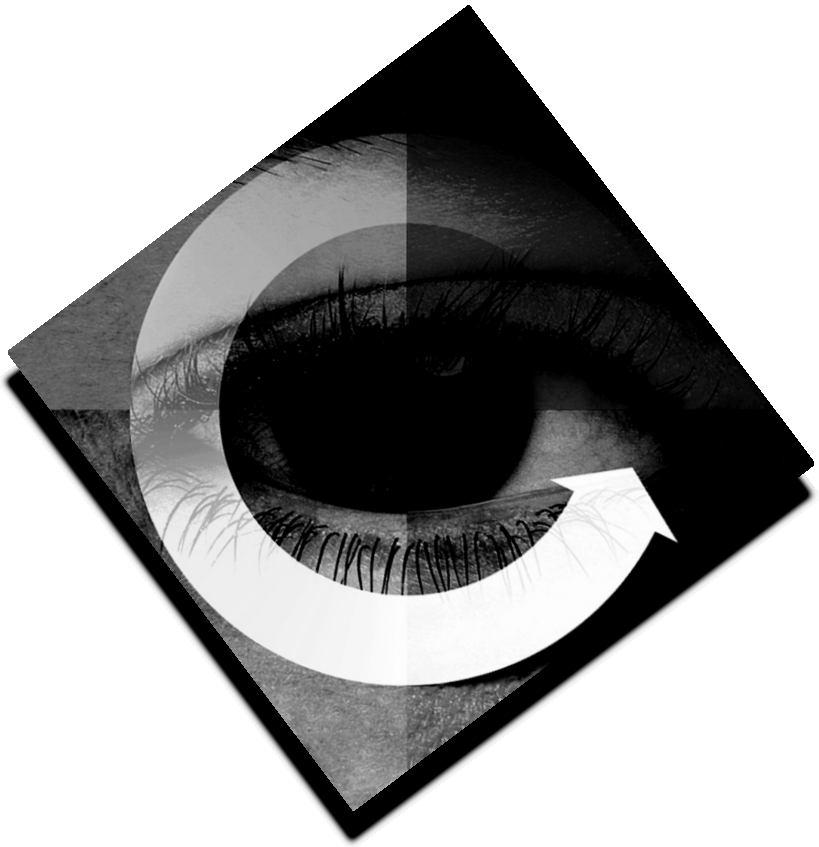
The IBM VisualAge Generator Newsletter

VisualAge® Generator

A Powerful New Vision of Programming™

Volume 2, Number 3
August 1997

Contents



Formerly the VisualGen Newsletter

VisualAge Generator Celebrates Another Birthday!

by Sandra Johnson, VisualAge Generator Product Manager

In June, the development and sales support teams for VisualAge Generator celebrated the product's 3rd birthday. It has been a great growth year for the VisualAge Generator customer base. Additionally, our first international user group meeting was held in Orlando, and the third EMEA user group meeting was held in London.

In this issue of the newsletter, we continue to bring you articles that provide hints and tips, insight into the future of the product, and descriptions of what other customers are doing to improve productivity and reduce costs in their companies. For example, this issue talks about the award-winning application developed by the State of California, some steps you can start taking to prepare for Version 3.0 of VisualAge Generator, generating JavaBeans wrapper classes for calling VisualAge Generator server applications, and additional techniques you can use for VisualAge Generator problem diagnosis.

Today, the development team is focused on delivering the Windows NT developer product. We have entered our final test stage, and we are excited about the new "look and feel" of the product. We are eager to hear what you think about it.

Let us hear from you soon!

Happy 3rd Birthday, VisualAge Generator!

As the VisualAge Generator team in Research Triangle Park (RTP) celebrates the third birthday of VisualAge Generator, we'd like to stop and thank each of you who uses the product for your part in that success. It is obvious that any success the product enjoys is directly attributable to its customers around the world!!! Thank you very much for your business now and in the future.

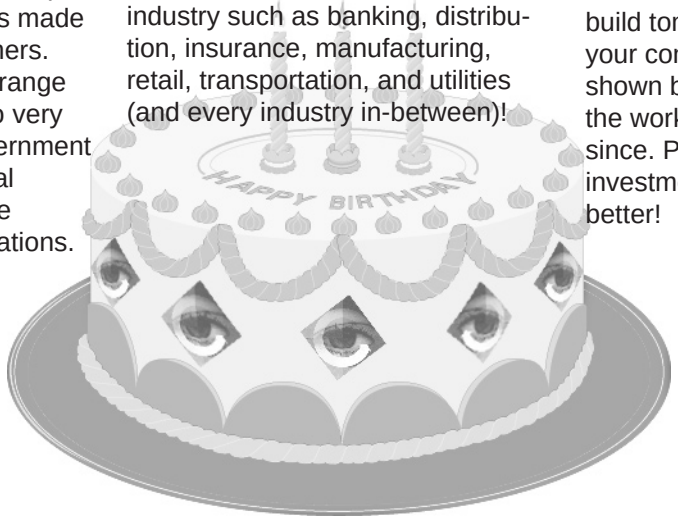
VisualAge Generator is installed in over 800 businesses, educational institutions, and government organizations all over the world. Those businesses range from very small companies with one application developer to very large companies with development shops made up of hundreds of programmers. The educational institutions range from local school systems to very large universities, while government users extend from small local governmental bodies to large federal and national organizations.

In the 800+ user organizations, there are thousands of developers using VisualAge Generator on a daily basis to create applications that range from traditional green-screen applications to the most sophisticated client/server applications in the market today. Many of these users are now investigating how to put those applications on the internet using the function shipped in Fixpack 5.

VisualAge Generator is a worldwide product with "homes" in 51 countries on every continent except Antarctica. In North America, it is being used in 38 of the 50 states, in 3 of the Canadian provinces, and in Mexico. It is used in virtually every industry such as banking, distribution, insurance, manufacturing, retail, transportation, and utilities (and every industry in-between)!

It is the belief of everyone on the VisualAge Generator team in RTP, NC, that the product is only getting started. With Version 3.0, which will become generally available soon, development on Windows NT will become a reality, bringing a whole new set of prospective users into the VisualAge Generator world. Add to that the power that interoperability with VisualAge Smalltalk brings to Version 3.0, and the future of the product appears to be unlimited.

However, success is built on success. The VisualAge Generator team recognizes daily how important our current customer population is to the success we hope to build tomorrow. Thank you again for your confidence in this product as shown by your initial acquisition and the work that you have done with it since. Please stay tuned—your investment is getting better and better!



VisualAge Generator Customer Wins Major Award!

The State of California, Department of Health Services won first place in an international competition co-sponsored by SOFTBANK COMDEX, the Object Management Group (OMG), and *Information Week* magazine. The State of California developed an Integrated State Information System (ISIS) application using VisualAge Generator and TeamConnection. The application won in the category, "Best Use of Object-Oriented Technology Within An Enterprise or Large Systems Environment." This object application award was announced at the Object World West conference in San Francisco on July 28.

Congratulations to the State of California!!!

You can find more information on this award-winning application in the story below.

A Customer Success Story

by Rusty Edmister, VisualAge Generator Sales and Technical Sales Support

Every month in the State of California, more than 1.2 million women and children from low-income families receive nutrition, education, and assistance through a federally-sponsored nutrition program. The Supplemental Nutrition Program for Women, Infants, and Children, also known as WIC, has proven effective under the direction of the State Department of Health Services. Every food dollar spent saves an estimated $3.50 in future medical costs.

Not so effective was the paper-based process in place until recently. Documents and other paperwork often took up to two-thirds of a client's scheduled appointment time. Now, however, thanks to a combination of process re-engineering, new technology, and application development techniques, administrative time has been slashed in half. Even more important, WIC clients benefit from a whole new level of service.

The key to this new level of personalized service is a new, object-based 4GL application called the Integrated State Information System (ISIS). The ISIS application was developed using a template-driven, object-based approach. Developers using OS/2-based PC workstations used tools from IBM's VisualAge family of products, particularly VisualAge Generator and TeamConnection. This approach enabled the developers to respond more quickly to changing requirements and to produce higher-quality code. Internal response times on the MVS host server have dropped from 2 to 5 seconds to subseconds, while the number of transactions during a normal 10-hour day has grown to over one million with VisualAge-generated code. In addition, developers have downloaded 90 percent of application development activity to the workstation, resulting in major productivity improvements and savings. Mike Virga, Staff Programmer and Analyst, and WIC ISIS Application Design Project Leader says, "We estimate that we're saving close to $150,000 (U.S.) per year by moving to our VisualAge development platform. And, where it would take three to four weeks to build, test and implement a standard data entry application in our older COBOL environment, with the VisualAge templates, we've reduced that time to 3 to 4 days, largely due to code reuse. That's a sevenfold productivity improvement!"

According to Virga, the goal is to continue to use the VisualAge toolkit to maximize efficiencies in ISIS and future development projects. Developers are now exploring how to move selected ISIS functions to a client/server platform, using VisualAge Generator. "The beauty of our new toolkit is that we've been able to create industrial-strength applications without being locked onto one platform environment. I don't believe any other programming languages around offer this capability."

(This article was excerpted from an application brief entitled, "Less paper, more services. New object-based application helps State of California improve services, cut costs." A copy of the full text may be ordered from IBM using publication order number GC09-2466.)

Get Ready for Version 3.0!

by Jeri Petersen, IBM VisualAge Generator Consulting Services

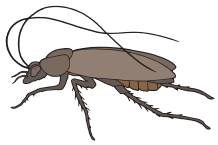
Are you ready for VisualAge Generator Version 3.0? This article lists some things you can do now to get ready for this new release.

- Avoid duplicate member names except for controlled situations such as:
  - A member is in the process of being changed so it exists in the production MSL and the changed version of the member exists in one of your test MSLs.
  - You are using templates, members are changed to include business logic, and your MSLs are structured so that there is a separate business logic MSL.
  - You have intentional duplicates between subsystems. For example, you might have a common MSL that contains a message handling program that uses a VisualAge Generator message table. Each subsystem has its own MSL and uses its own version of the message table with the contents set as required for the subsystem.
- Eliminate unintended duplicates that have occurred over time. The following might have occurred:
  - A member was accidentally copied into an incorrect MSL and never deleted.
  - Developers ignored naming conventions and named two unrelated members in different subsystems with the same name.
- Make sure that you have not lost members over time. This could happen if generation for your production system was done from a developer's read/write MSL and the members were never advanced to the production MSL. Running validation is one way to check that all required members still exist.
- Put procedures in place to prevent creating new, unintentioned duplicates and to avoid losing members.
- Consider assigning one developer as the owner of a group of functionally-related applications. This process will help determine how your current applications are grouped into ENVY applications. Make the groups small enough so that very few developers are working on the same group of applications at the same time. If you already have a database administrator responsible for maintaining SQL row record definitions and their associated global data items, this is a first step in assigning ownership.
- If you develop GUIs:
  - GUI application names must start with a letter (A–Z) and the remaining characters must be alphanumeric (A–Z, 0–9). National and special characters are not permitted. With VisualAge Generator Version 3.0, the restriction will be enforced. Therefore, ensure that your GUIs adhere to these restrictions.
  - Do not make connections between attributes that have different data types unless either the source or target is a VisualAge Generator data part or a tear-off from a data part. For example, do not connect a string to something that expects a Boolean data type. With Version 3.0, the only supported connections between objects of different data types require one of the objects to be a VisualAge Generator data part or a tear-off from a data part.
  - The Container Details part was introduced in VisualAge Generator Version 2.2. The Container Details part was intended to replace the visual table part. With Version 3.0, the visual table part will still work for existing GUIs. However, it will no longer appear on the parts palette. Therefore, you might want to do any new development using the Container Details part.

If you are just starting to use VisualAge Generator, you should also do the following to prepare for Version 3.0.

- Structure your MSLs for system test, acceptance test, and production so that the only members at the system test and acceptance test level are members that have changed from the next level of test and production. Do not have a complete (stand-alone) set of members at each level of test. See *Planning for VisualAge Generator* (SH23-0226-00) for a description of a good way to implement an MSL structure.
- Establish and enforce naming conventions to help you relate the various members of an application. See *Planning for VisualAge Generator* for a description of a naming convention that helps to tie together application-unique members.

Tracking Down Those Bugs



by Chuck Proffer and Jim Eberwein, VisualAge Generator Development

VisualAge Generator provides a robust test facility that enables programmers to diagnose coding errors in application logic prior to incorporating the applications into an enterprise production system. Using the test facility, developers are more productive in developing high-quality applications. Although the test facility does an excellent job of emulating the production system, problems not detected in the development environment might occur after the applications are ported to the execution platform. At this time, application developers need to rely on other diagnostic tools to determine the cause of the problem. VisualAge Generator provides additional tools to assist developers with problem diagnosis. This article provides insight to some of these diagnostic tools.

Tracing Connections

The connections defined in GUI clients can be traced while the client applications run. This enables the developers to determine what connections are being triggered during run time. You can start the connection trace in GUI runtime by issuing the EZE2RUN PROFILE ON command and issue EZE2RUN PROFILE OFF to end the trace. The output is written to the TSCRIPT.LOG file pointed to by the EZERTEMP environment variable. Note that you must start DEBUG before any output is written to TSCRIPT.LOG. To start DEBUG, enter the EZE2RUN DEBUG ON command or set the environment variable EZERRUN_DEBUG=1. Stop tracing prior to viewing the trace. You might not be able to read the contents of the TSCRIPT.LOG file while the GUI runtime is running, since the file is locked. Either stop the runtime by issuing the EZE2RUN KILL command or use the OS/2 EPM editor to view the file.

Once you have obtained the trace in TSCRIPT.LOG, you can compare it to a trace of the GUI events that you can obtain when using the TRACE function of the test facility. You should ensure that the GUI events are executing in the same order in both traces.

TSCRIPT.LOG Contents

The TSCRIPT.LOG file has the following three sections:

- Loading Applications
- Event History
- Event Profile

The *Loading Applications* section of the TSCRIPT.LOG file lists the GUI programs that are loaded, the date and time the programs were created, and the VisualAge runtime image in which the program is loaded.

The *Event History* section lists all of the events that have occurred in the system. This section is comparable to the trace that is produced via the test facility.

The *Event Profile* section gives a thorough breakdown of the time it takes to run each event in the system. Instead of listing the events in chronological order, the events are listed according to the amount of time it takes to complete the event. The events with the longer elapsed time are listed first. The elapsed time is the time it took to run the event in milliseconds. Also listed in the event entry is the number of times the event occurred, its correlating entry in the event history section, and the depth of connections from where the event was triggered.

The entry reading “Time to first” event should be ignored. This event measurement is not meaningful, as it usually involves the time between the user turning on profiling until the GUI started running. This means that there is user think time involved that could explain large variances in the times recorded.

A more interesting and significant time measurement is how long it takes to load and open the GUI. To obtain this information in the TSCRIPT.LOG file, you must issue the EZE2RUN BENCH ON command or set the environment variable EZERBENCH=1. When you use the BENCH option, the “Loading Applications” section shows a breakdown of the loading time for the GUI and the time in milliseconds for each load. Also listed are the events that trigger calls to the 4GL VisualAge Generator components that are available to the GUI. The entries are indicated by either the event-to-action connection or the label associated with the push button that triggered the event.

Walkbacks

Walkbacks are stack-dumps that contain pertinent information when a Smalltalk error occurs and the EZERDEBUG and EZERRUN_DEBUG commands are issued, depending on whether the error occurs in the runtime or the development environments. To debug a runtime error, set the environment variable EZERRUN_DEBUG=1. Likewise, set the environment variable EZERDEBUG=1 to debug a VisualAge Generator Developer error. The walkback dump is written to the WALKBACK.LOG file. This file is appended to after each Smalltalk error, so it is important that you review the appropriate stack-dump when debugging a problem. When possible, delete the existing WALKBACK.LOG file prior to re-creating a problem. The stack-dumps are separated by two blank lines and a heading line that indicates the time and date the walkback occurred (for example,

Walkback at 11:17:16 AM on 01-08-97). Thus, if the file contains multiple dumps, scan the file from the bottom up until you find the beginning of the last dump contained in the file.

When reviewing a specific stack dump, search for the last indication that an error has occurred. This is noted either by the keyword “Error:” or by the last line in the dump that matches the message reported on the first line in the dump; for example, “(ExCLDTIndexOutOfRange) Index out of range.” The lines that follow explain the Smalltalk error and specific objects that were involved. At this point, you should review the dump to see what GUI connection triggered the Smalltalk error. Note that not all walkbacks are produced via an invalid connection. In these cases, the cause is most likely due to an error in the VisualAge Generator product. If this is the case, the WALKBACK.LOG file might provide insight to the VisualAge Generator development team.

For additional information regarding either the TSCRIPT.LOG or WALKBACK.LOG file, refer to the *Advanced GUI Development Guide, Learning to Walk, Run, Fly!!!* redbook, SG24-4238.

The Communications Middleware Trace Facility

VisualAge Generator supports a wide range of communication services that provide the middleware support in the client/server environment. Very often, errors are detected when attempting to implement a client/server paradigm using VisualAge Generator for the first time. Developers or system administrators can use the tracing facilities provided by the VisualAge Generator Communications Middleware to diagnose these problems they are experiencing.

Controlling Trace Output

The following environment variables control what information is traced and the name of the trace file.

CSOTROUT

This environment variable specifies the name of the trace output file. If the variable is not specified, the default name is CSOTRACE.OUT.

CSOTROPT

This environment variable specifies the trace options. If CSOTROPT is not specified or is set to 0, the trace is not produced; however, any messages produced as the result of an error during run time are still logged to the trace output file. The trace options are:

- 0–turn off trace
- 2–trace client/server events

The following sample trace output was produced with the option CSOTROPT=2:

```
<Jul 17 11:26:05>–>CMINIT
<Jul 17 11:26:05><–CMINIT
<Jul 17 11:26:05>–>CMCALL
<Jul 17 11:26:05>  ++CMCALL
<Jul 17 11:26:05>  Calling application ELACVP5
<Jul 17 11:26:05>  ++++readFromLinkTbl
<Jul 17 11:26:05>  ====0.054545 s
<Jul 17 11:26:05>  ++++loadAndInitDriver
<Jul 17 11:26:06>  ====0.160220 s
<Jul 17 11:26:06>  ++++CMDV_INIT
<Jul 17 11:26:06>  ====0.007162 s
<Jul 17 11:26:06>  –>DCE:CMDV_CALL
<Jul 17 11:26:06>    ++++DCE:CMDV_CALL
<Jul 17 11:26:06>    ++++++DCE:CreateParmBlock
<Jul 17 11:26:06>    ======0.005037 s
<Jul 17 11:26:06>    Using binding handle: 546ddf54-ce97-11cf-9269-
                                           08005afc355d@
<Jul 17 11:26:06>    Passing 7 bytes of data
<Jul 17 11:26:06>    ++++++DCE:RPCCall
<Jul 17 11:26:06>    ======0.120231 s
<Jul 17 11:26:06>    ====0.701094 s
<Jul 17 11:26:06>  <–DCE:CMDV_CALL
<Jul 17 11:26:06>  ==0.986634 s
<Jul 17 11:26:06><–CMCALL
<Jul 17 11:26:07>–>CMCLOSE
<Jul 17 11:26:07><–CMCLOSE
```

The sample trace was generated from an OS/2 VisualAge Generator client calling the VisualAge Generator sample application ELACVP5 on AIX, using DCE communications. The trace shows each of the major events that occurs in the communications middleware along with timing information. The contents of each trace varies depending on the communications protocol being used and whether the applications use DB2.

CSO_DUMP_DATA

When this environment variable is set to 1, the VisualAge Generator Communications Middleware produces trace entries that document the linkage table parameters used for the server call, entries that describe the parameters being passed, and the values of these parameters. The descriptor and value information is documented prior to and after the call to the server. This information is written to the file CSODUMP.OUT.

CSO_DUMP_CONV

When this environment variable is set to 1, the VisualAge Generator Communications Middleware produces trace entries that describe the parameters being passed and their values prior to and after the middleware support converts the data. The conversion table used is also documented in the trace. This information is written to the file CSODUMP.OUT.

Parameter Descriptors

The parameter descriptor is a variable length structure that defines the format of the parameter data. The descriptor is used when moving the parameter data to the transmission buffer and in converting the data format from the client format to server format or vice versa.

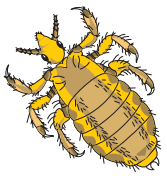
The structure of the descriptor is made up of various substructures. Four of these substructures are documented in the *VisualAge Generator Supplement* (SH23-0242) document. Other descriptor substructures can be included in the descriptor. When interpreting the descriptor information for a parameter and mapping it to the parameter data, the understanding of these four substructures should be sufficient. However, other substructures can be included in the parameter descriptor. You should understand the length of these substructures, so you can successfully interpret the values of the documented substructures. The following table documents the different control fields associated with each substructure and the number of bytes that follow the control field in each substructure. Control fields associated with the low-level data items in the passed parameter have been omitted from the table. They are documented in the Supplement document, under the CMITEM parameter descriptor.

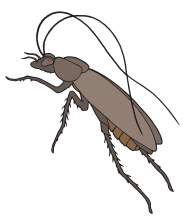
Value	Description	Followed by
'F3'	Max parameter size	4 byte binary length field
'F4'	Returned data length	2 byte binary length field
'F5'	DLI segment data length	2 byte offset into buffer of field containing length of the segment. The length of the segment is from this offset. It is followed by descriptor for the field.
'F6'	Map record indicator	Nothing
'F7'	SQL record indicator	Nothing
'F8'	Obsolete, ignored by the conversion routine.	2 byte Max field size
'F9'	Internal data length	2 byte offset into buffer followed by descriptor for field.
'FA'	External data length	2 byte length field
'FB'	Number of occurrences	2 byte offset of NUMOCCUR field indicator field in buffer and descriptor for field.
'FC'	Variably occurring structure	Number of fields and max number of occurs.
'FD'	Occurring structure	Number of fields and max number of occurs.
'FE'	Record name	18 character record name
'FF'	End of parameter list	Nothing

Descriptor Example

If the following record definition is used as a parameter:

NAME	LEVEL	OCCURS	TYPE	BYTES
*	5	1	Char	629
CHARITEM1	10	1	Char	8
BINITEM1	10	1	Bin	2
NUMITEM1	10	1	Num	4
ARRAY1	10	20	Char	30
STRUCTURE1	10	1	Char	10
CHARITEM2	15	1	Char	5
HEXITEM1	15	1	Hex	3
*	15	1	Char	2





You should expect to see the following descriptor in the trace:

```
Description #0

f3,75,2,0,0,fe,52,36,35,38,
32,8,0,0,0,0,0,0,0,0,0,0,0,
0,2,8,0,1,2,0,6,9,0,fd,1,0,
14, 0, 2, 1e, 0, 2, 5, 0,
2,3,0,2,2,0,ff,
```

The x'f3' indicates the beginning of the parameter descriptor, for example, the CMPARMLEN. The next four bytes indicate that the parameter has a length of 629 bytes. The next byte (x'fe') maps to the name of the record. This is shown in the next 18 bytes of the descriptor. After the name information, the following sequence needs to be interpreted:

```
2,8,0,1,2,0,6,9,0,fd,1,0,14,0,2,
1e,0,2,5,0,4,3,0,2,2,0,ff,
```

Here is where we will rely on the information in the Supplement document to analyze these hex values.

- The 2,8,0 indicates a Char data item with length 8 bytes.
- The 1,2,0 indicates a Bin data item with length 2 bytes.
- The 6,9,0 indicates a Num data item with length 9 bytes.
- The fd,1,0,14,0 indicates an array of 20 items that is not substructured.
- The 2,1e,0 indicates a Char data item with length 30 bytes.

  Since the previous descriptor data indicated an array of that contained one descriptor, we know that the array has 20 occurrences of 30-byte character data.
- The 2,5,0 indicates a Char data item with length 5 bytes. Note that the descriptor does not have information concerning the STRUCTURE1 data item.
- The 4,3,0 maps to a Hex data item with length 3 bytes.
- The 2,2,0 maps to a Char data item with 2 bytes.
- The ff indicates the end of the descriptor.

Debugging C++ Applications

VisualAge Generator C++ Workgroup Services provides a trace facility that logs error and trace information. Error information is automatically logged to the trace file. Trace information is controlled by the FCWTROPT environment variable. The capability to add customized trace statements to the application so that specific sections of the application logic can be traced and the contents of data items can be displayed in the trace file are also provided.

Controlling Trace Output

The following environment variables control what information is traced and the name of the trace file. Custom trace statements that are added to the application are not affected by the FCWTROPT environment variable. The trace statements must be removed from the application to stop the trace output. In the native environments (non-CICS), the environment variables can be set in the config.sys file (on OS/2), your profile (on AIX), the Control Panel (on Windows NT), or be specified on the command line prior to running FCWRUN. In the CICS environments (CICS for AIX, CICS for NT), the trace statements must be specified in the CICS environment file and CICS must be restarted.

FCWTROUT

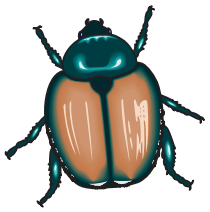
This environment variable specifies the name of the trace output file. If the variable is not specified, the default name is FCWTRACE.OUT.

FCWTROPT

This environment variable specifies the trace options. If FCWTROPT is not specified or is set to 0, the trace is not produced. However, any VisualAge Generator Workgroup Services messages produced as the result of an error during run time are still logged to the trace output file. The trace options are:

- 0–turn off trace
- 1–trace entry or exit from the application, process, or statement group
- 2–trace CALL, XFER, or DXFR statements
- 4–trace SQL I/O
- 8–trace file I/O
- 16–trace system events

Multiple trace options can be turned on by adding the appropriate values together. To turn on all options, specify FCWTROPT=31.



The following sample trace output was produced with FCWTROPT=31:

```
           (00957)<17:45:46>        -> CSO::CMINIT()    rc = 0
           (00957)<17:45:46>Starting RunUnit
           (00957)<17:45:46>Using RSC name, fcw.rsc
           (00957)<17:45:46>Found EZERNLS environment variable, name=ENU
EMPLOYE    (00957)<17:45:47> -> EMPLOYE::MAIN
EMPLOYE    (00957)<17:45:49>        -> ( SQL::DFTCONN ) Database = (SAMPLE) rc = 0
EMPLOYE    (00957)<17:45:49>             User = () Password = (XXXXXXXX) UOW = (D1E)
EMPLOYE    (00957)<17:45:49>     -> INIT
EMPLOYE    (00957)<17:45:49>     <- INIT
EMPLOYE    (00957)<17:45:49>     -> STEPVU
EMPLOYE    (00957)<17:45:49>         -> Converse EMPMAP.EMP001
EMPLOYE    (00957)<17:45:49>       -> ( SQL::COMMIT   )   rc = 0
EMPLOYE    (00957)<17:45:53>         <- Converse EMPMAP.EMP001
EMPLOYE    (00957)<17:45:53>     -> PROCSCN
EMPLOYE    (00957)<17:45:53>         -> ( SQL::SETINQ   )   Handle = 9
EMPLOYE    (00957)<17:45:53>              rc = 0
EMPLOYE    (00957)<17:45:53>     -> GETNEXT
EMPLOYE    (00957)<17:45:53>         -> ( SQL::SCAN     )   Handle = 9
EMPLOYE    (00957)<17:45:53>              rc = 0
EMPLOYE    (00957)<17:45:53>     <- GETNEXT
EMPLOYE    (00957)<17:45:53>     -> MSGSETU
EMPLOYE    (00957)<17:45:53>     <- MSGSETU
EMPLOYE    (00957)<17:45:53>     -> SHOWREC
EMPLOYE    (00957)<17:45:53>         -> Converse EMPMAP.EMP001
EMPLOYE    (00957)<17:45:54>         <- Converse EMPMAP.EMP001
EMPLOYE    (00957)<17:45:54>     <- SHOWREC
EMPLOYE    (00957)<17:45:54>     -> GETNEXT
EMPLOYE    (00957)<17:45:54>         -> ( SQL::SCAN     )   Handle = 9
EMPLOYE    (00957)<17:45:54>              rc = 0
EMPLOYE    (00957)<17:45:54>     <- GETNEXT
EMPLOYE    (00957)<17:45:54>     -> MSGSETU
EMPLOYE    (00957)<17:45:54>     <- MSGSETU
EMPLOYE    (00957)<17:45:54>     -> SHOWREC
EMPLOYE    (00957)<17:45:54>         -> Converse EMPMAP.EMP001
EMPLOYE    (00957)<17:45:56>         <- Converse EMPMAP.EMP001
EMPLOYE    (00957)<17:45:56>     <- SHOWREC
EMPLOYE    (00957)<17:45:56>     -> GETNEXT
EMPLOYE    (00957)<17:45:56>         -> ( SQL::SCAN     )   Handle = 9
EMPLOYE    (00957)<17:45:56>              rc = 0
EMPLOYE    (00957)<17:45:56>     <- GETNEXT
EMPLOYE    (00957)<17:45:56>     -> MSGSETU
EMPLOYE    (00957)<17:45:56>     <- MSGSETU
EMPLOYE    (00957)<17:45:56>     -> SHOWREC
EMPLOYE          (00957)<17:45:56>         -> Converse EMPMAP.EMP001
EMPLOYE (00957)<17:45:57>         <- Converse EMPMAP.EMP001
EMPLOYE (00957)<17:45:57>     <- SHOWREC
EMPLOYE (00957)<17:45:57>     -> GETNEXT
EMPLOYE (00957)<17:45:57>        -> ( SQL::SCAN     )   Handle = 9
EMPLOYE (00957)<17:45:57>             rc = 0
EMPLOYE (00957)<17:45:57>     <- GETNEXT
EMPLOYE (00957)<17:45:57>     -> MSGSETU
EMPLOYE (00957)<17:45:57>     <- MSGSETU
EMPLOYE (00957)<17:45:57>     -> SHOWREC
EMPLOYE (00957)<17:45:57>         -> Converse EMPMAP.EMP001
EMPLOYE (00957)<17:45:58>         <- Converse EMPMAP.EMP001
EMPLOYE (00957)<17:45:58>     <- SHOWREC
EMPLOYE (00957)<17:45:58>     -> CLOSSQL
EMPLOYE (00957)<17:45:58>        -> ( SQL::CLOSE    )   Handle = 9
EMPLOYE (00957)<17:45:58>             rc = 0
EMPLOYE (00957)<17:45:58>     <- CLOSSQL
EMPLOYE (00957)<17:45:58>     <- PROCSCN
EMPLOYE (00957)<17:45:58>     <- STEPVU
EMPLOYE (00957)<17:45:58> <-
EMPLOYE::MAIN
                (00957)<17:45:58>Ending RunUnit
                (00957)<17:45:58>        -> ( SQL::COMMIT   )   rc = 0
                (00957)<17:45:59>        -> CSO::CMCONCT( DISC )   rc = 0
                (00957)<17:45:59>        -> CSO::CMCLOSE()   rc = 0
```

This sample trace was generated by the VisualAge Generator sample application EMPLOYE running on OS/2. Each line in the trace will contain the name of the application, the process ID (obtained from the operating system), the current time, and a description of the occurring event.

Adding Trace Statements to the Application

If the trace information provided by Workgroup Services is not sufficient to diagnose the problem, you can add custom trace statements to the application. The process consists of editing the C++ source file, adding the trace statements, and repreparing the application. The trace statements use the following syntax:

```
Trace() << "string enclosed in quotes" << VAG_Data_Item << endT;
```

The output operator (<<) is used to direct a value to the trace file. Successive output operators can be concatenated. The manipulator endT acts as a new line character, causing any output that follows to be directed to the next line. The following trace statement would cause the string "Hello World!" to be written to the trace file.

```
Trace() << "Hello World!" << endT;
```

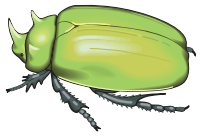
The value of a data item can also be written to the trace file. The C++ source file must be examined and the generated name of the data item must be specified in the trace statement. All of the VisualAge Generator data types can be traced. The following example shows a section of code extracted from a VisualAge Generator C++ application and the trace statements used to display the values of the data items:

```
DATAWS_.BIN_ITEM_ = 99;
DATAWS_.CHA_ITEM_ = "ABC";
DATAWS_.HEX_ITEM_ = "0D0A";
DATAWS_.NUM_ITEM_ = 99;
DATAWS_.NUMC_ITEM_ = 99;
DATAWS_.PACF_ITEM_ = 99;
DATAWS_.PACK_ITEM_ = 99;

Trace() << "BIN_ITEM: " << DATAWS_.BIN_ITEM_ << endT;
Trace() << "CHA_ITEM: " << DATAWS_.CHA_ITEM_ << endT;
Trace() << "HEX_ITEM: " << DATAWS_.HEX_ITEM_ << endT;
Trace() << "NUM_ITEM: " << DATAWS_.NUM_ITEM_ << endT;
Trace() << "NUMC_ITEM: " << DATAWS_.NUMC_ITEM_ << endT;
Trace() << "PACF_ITEM: " << DATAWS_.PACF_ITEM_ << endT;
Trace() << "PACK_ITEM: " << DATAWS_.PACK_ITEM_ << endT;
```

The following lines would be written to the trace file after executing the above trace statements:

```
TSTDATA (01675)<09:54:08>BIN_ITEM: 99
TSTDATA (01675)<09:54:08>CHA_ITEM: ABC
TSTDATA (01675)<09:54:08>HEX_ITEM: 0D0A
TSTDATA (01675)<09:54:08>NUM_ITEM: 99
TSTDATA (01675)<09:54:08>NUMC_ITEM: 99
TSTDATA (01675)<09:54:08>PACF_ITEM: 99
TSTDATA (01675)<09:54:08>PACK_ITEM: 99
```



Debugging COBOL Applications

VisualAge Generator COBOL applications can be generated to provide tracing of the 4GL statements and SQL calls (except SQLEXEC processes). To enable the source level tracing, the application must be generated with the /TRACE generation option. The /TRACE option can be specified with the following parameters:

- SQLERR—Trace only SQL error codes returned in the SQLCA.
- SQLIO—Trace both the data and SQL error codes in the SQLCA.
- STMT—Trace the VisualAge Generator 4GL statements coded in the application.

Note that multiple parameters can be included with the /TRACE option. The parameters need to be separated by commas.

To activate the trace at runtime, run the ELAZ transaction in the CICS and IMS/VS environments or specify the ELATRACE DD in the Batch, TSO, and IBM BMP environments. Note that the trace function is not supported in the OS/400 environment.

The output from the trace is stored in different locations based upon the runtime platform. Refer to the appropriate *Running Applications on OS/2, AIX, and Windows* document for information concerning printing the trace output.

Generating JavaBeans Wrappers

by Paul Hoffman, VisualAge Generator Development

VisualAge Generator generates JavaBeans wrapper classes for calling VisualAge Generator server (remote called batch) applications. The generated classes represent:

- Server applications
- Record parameters
- Record array rows

The wrappers help you (the Java developer) by performing the following functions on the server call:

- Extended unit of work control for calls to CICS and OS/400 servers
- Data marshalling and format conversion between:
  - Objects and record structures
  - Unicode and ASCII or EBCDIC code pages
  - Floating point and decimal or packed decimal numbers

You can use the classes to build Java applications and applets. When building Java applications, the server wrapper calls the VisualAge Generator POWERserver API from the system on which the application is running. When building Java applets, the wrapper runs on a Web client and uses Java Remote Method Invocation to request that the POWERserver API be called from the Web server by the VisualAge Generator UnitOfWorkServer object you started on the Web server.

This article describes how to get started and provides some examples of how you code the JavaBeans wrappers. For a complete description, see the *VisualAge Generator Developing Client/Server Applications* (SH23-0230) document. If you are using VisualAge Generator Version 3 or if you have installed VisualAge Generator Version 2.2 FixPak 5 or later, see the FixPak readme file for information on Java support.

Getting Started

To generate JavaBeans wrapper classes for calling a server (remote called batch application), VisualAge Generator needs, as input, the application member and the members defined in the called parameter list. All parameter types are supported, except for maps and SQL row records. As an option, you specify /SYSTEM=JAVAWRAPPER as the target runtime system. Other options you can specify include:

- /MSL=
- /LINKAGE=
- /GENOUT=
- /NLS=

An example of a batch generation command for generating Java wrappers for application STAFFMN follows:

```
eze2gen generate staffmn /msl=staff /system=javawrapper
 /genout=f:\vgout /nls=enu /linkage=f:\link\staff.lnk
```

Linkage table options: The linkage table entry for the called batch application defines how the application should be called using the PowerServer API. You can specify that the linkage table options are generated into the server wrapper by indicating REMOTEBIND=GENERATION. Or you can specify that only the application name and linkage table name are generated into the server application by specifying REMOTEBIND=RUNTIME.

Conversion Tables (CONTABLE Option): Java applications use Unicode 16-bit character encoding at runtime. Java specific conversion tables support conversion of the Unicode text to the appropriate ASCII or EBCDIC code page used on the server system. See the readme file provided with the product for a list of the conversion tables.

After generation, each generated class is in a file with the name:

```
classname.java
```

where classname is the class name derived from the VisualAge Generator name for the object. The generated classes are assigned to package:

```
serverP
```

where server is the class name derived from the server application name.

You can use the Javadoc tool to build a classname.html file from the Java file. The HTML file describes the public interfaces for the class.

JavaBeans For Servers

A JavaBeans class generated for calling a server application includes:

- *Private instance* variables for each parameter.
- Get and set methods for each parameter.
- An execute method for calling the server application using the *class instance* variables as parameters.

- A call method for calling the server with an argument defined corresponding to each parameter in the application called parameter list.
- The *addPropertyChangeListener* and *removePropertyChangeListener* methods that enable signaling of other beans when parameter values are changed by a set method or on return from a server call. These methods are inherited from the *Server* class.

You can use the server as a JavaBean by using the set methods to set parameter values prior to calling the server, the execute method to call the server, and the set methods to retrieve the returned parameter values after calling the server. The listener methods enable another bean to be notified when whenever the data values are changed.
You can also treat the server call as a function call, passing the parameters as arguments on the call method. If you use the call method, then you must use the get methods to retrieve the values returned for item parameters, since Java primitive parameters are always passed by value.

JavaBeans For Record Parameters

A JavaBeans class generated for a record includes the following:

- *Public instance* variables, such as the following:
  - Java primitives for each low-level item that is not within a substructured array.
  - Java primitive arrays for item arrays
  - An object array for each substructured array item
- The get and set methods for each *instance* variable, enabling the record class to be used as a JavaBean.
- The *addPropertyChangeListener* and *removePropertyChangeListener* methods enable signaling of other beans when parameter values are changed by a set method or on return from a server call. These methods are inherited from the *Record* class.
  Always use the set methods to change values of *Record* variables if other beans are listening.
- Non-public methods for marshalling record data on a server call occurring structure in the record.

JavaBeans For Record Array Rows

A JavaBeans class generated for a multiple occurring substructure in a parameter record includes the following:

- *Public instance* variables defined as Java primitives for each low-level item in the substructure.
- The get and set methods for each *instance* variable, allowing the class to be used as a JavaBean.
- The *addPropertyChangeListener* and *removePropertyChangeListener* methods enable signaling of other beans when parameter values are changed by a set method or on return from a server call. These methods are inherited from the *RecordArrayRow* class.

  Always use the set methods to change values of *RecordRow Array* variables if other beans are listening.
- Non-public methods for marshalling record data on a server call.

VisualAge Generator Java Package

A Java package, *ibm.cso*, is shipped with VisualAge Generator runtime for OS/2, Windows NT, and Windows 95. *cso* is the three-character identifier for VisualAge Generator POWERserver API middleware. The classes in the package are used with the generated JavaBeans to communicate with VisualAge Generator server applications.

To make the *ibm.cso* package accessible to Java, make sure the Java *classpath* environment variable includes the subdirectory *vgbase\java* in the directory list, where *vgbase* is the base directory for the VisualAge Generator product.

The *ibm.cso* package includes the following class definitions. For descriptions of these classes, see the csojava.html file provided with the product.

ibm.cso.UnitOfWork
ibm.cso.AppletUnitOfWork
ibm.cso.ApplicationUnitOfWork
ibm.cso.UnitOfWorkServerImpl
ibm.cso.CSOException
ibm.cso.CSOCallOptions
ibm.cso.Server
ibm.cso.Record
ibm.cso.RecordArrayRow
ibm.cso.Wrapper
ibm.cso.CSOConstants
ibm.cso.CSOConversionTable
ibm.cso.PowerServer
ibm.cso.CSOSession
ibm.cso.CSOSessionState
ibm.cso.Trace

Examples

How to Call Server Wrappers From Applications

An *ApplicationUnitOfWork* object establishes a communication session with the VisualAge Generator POWERserver API for the purpose of calling VisualAge Generator server applications via the POWERserver API.

To call a server application named STAFFMN from a Java application, generate a Java wrapper class for the server application using VisualAge Generator (the Java class name will be Staffmn). Then code the following in the Java application:

```
{
try
  {
  // Initialization
     ApplicationUnitOfWork auow = new ApplicationUnitOfWork();
     Staffmn staffmn = new Staffmn(auow);
  // Processing
     .
     .
     staffmn.call(.....
     .
     .
// Commit (use only if linkage table specifies client unit of work)
    auow.commit();
     .
     .
}
catch (ibm.cso.CSOException error)
{
String explanation = error.getMessage();
     .
     .
// Rollback (use only if linkage table specifies client unit of work)
    auow.rollback();
     .
     .
}
// Explicitly close the PowerServer unit of work when the application
// ends.  Do not rely on garbage collection, which may leave the
// session hanging for a long period of time.
//
if (auow != null)
{
try
  {
  auow.close();
  }
  catch (ibm.cso.CSOException error)
  {
  String explanation = error.getMessage();
     .
     .
  }
}
```

Multiple server applications can be called from the same ApplicationUnitOfWork.

How to Call Server Wrappers From Applets

An AppletUnitOfWork object establishes communication with a UnitOfWorkServer object on a Web server using the Java 1.1 remote method invocation (RMI) for the purpose of calling VisualAge Generator server applications via the UnitOfWorkServer object. To call a server application named STAFFMN from an applet, generate a Java wrapper class for the applet using VisualAge Generator (the Java class name will be Staffmn). Then code the following in the applet. Multiple server applications can be called from the same AppletUnitOfWork object.

```
AppletUnitOfWork auow ;
Staffmn staffmn ;

public void init()
        {
        try
                  {
                  // Initialization
                     auow = new AppletUnitOfWork(this) ;
                  // or
                  // auow = new AppletUnitOfWork(hostname) ;

                        staffmn = new Staffmn(auow);
        }
        catch (ibm.cso.CSOException error)
        {
        String explanation = error.getMessage();
                  .
                  .
                  }
        }
public boolean action(Event event, Object arg)
        {
                  .
                  .
        // Handle action requiring server call
        try
        {
         // Processing
                   .
                   .
                  staffmn.call(.....
                  .
                  .
        // Commit (use only if linkage table specifies client unit of work)
           auow.commit();
                  .
                  .
        }
        catch (ibm.cso.CSOException error)
        {
        String explanation = error.getMessage();
                  .
                  .
        // Rollback (use only if linkage table specifies client unit of work) auow.rollback();
                  .
                  .
        }
                  .
                  .
// Handling request to terminate applet
//
// Explicitly close the PowerServer unit of work when the application
// ends. Do not rely on garbage collection, which may leave the
// session hanging for a long period of time.
//
if (auow != null)
        {
        try
         {
         auow.close();
         }
        catch (ibm.cso.CSOException error)
         {
         String explanation = error.getMessage();
         .
         .
         }
        }
```

How to Run Applets From a Browser

To run an applet, you need to reference the applet in an HTML file you download from your Web browser. The text of a minimal HTML file (named Staffmn.html) for running an applet follows:

```
<APPLET code="StaffmnApplet.class" width=400 height=400>
</APPLET>
```

Your browser must be capable of running Java 1.1 applets to work with VisualAge Generator classes. To run your applet from a Java SDK applet view, enter the following command:

```
appletviewer http://hostname/hostalias/Staffmn.html
```

where *hostname* is the network identifier of your Web server system and *hostalias* is the alias by which the directory on which Staffmn.html resides is known to the server.

How to Start a UnitOfWorkServer Object on Your Web Server

To start the UnitOfWorkServer object, your Java Web server system must be at Java Version 1.1 or higher. The UnitOfWorkServer object provided with VisualAge Generator registers as a server with the Java Remote Method Invocation registry for the purpose of calling the POWERserver API on behalf of Java applets.

To start the JAVA Remote Method Invocation registry on the host system, enter the following:

```
start rmiregistry
```

To start a UnitOfWorkServer object for handling calls from applets to VisualAge Generator server applications, enter the following:

```
java -Djava.rmi.server.codebase=http://hostname/CSOServer/   ibm.cso.UnitOfWorkServerImpl event_level
timeout_int
```

Where:

- hostname = name of your internet host server system
- event_level = level of event report. Controls the reporting of events handled by the UnitOfWorkServer object. The events are written using the Java System.out.println method, and by default go to the window from which the server was started.
  - 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.
- report_interval = server session report interval in minutes. 0 = no report. Default is 5 minutes.

Java Names

The names of classes, methods, and objects in the generated class definitions are derived from VisualAge Generator member names according to the following algorithm:

- Convert VisualAge Generator name to lowercase.
- Delete any dashes (-) or underscores (_) and change the character that follows the dash or underscore back to uppercase.

- When the converted name is used as class name or within a method name, translate the first character back to uppercase.
- Package name for generated objects is the server class name with a P appended to it. DBCS names are not supported.

The following table shows an example of the derivation of Java wrapper names from VisualAge Generator member names.

VisualAge Generator Member Type	VisualAge Generator Name	Java Class Name	Java Object Name	Java Package Name	Java Get Method Name	Java Set Method Name
Server called batch) application name	STAFFMN	Staffmn	staffmn	StaffmnP	N/A	N/A
Record name for record parameter	STAFF-MAINT	StaffMaint	staffMaint	StaffmnP	getStaffMaint	setStaffMaint
Level-77 item parameter	BUTTON-PRESSED	N/A	buttonPressed	N/A	getButtonPressed	setButtonPressed
Name of substructured, multiply occurring item in record	STAFF-DATA in STAFF-MAINT	StaffMaint_StaffData	staffData	StaffmnP	getStaffData	setStaffData
Low level item in record or record array	ROWS-FETCHED	N/A	rowsFetched	N/A	getRowsFetched	setRowsFetched

Data Type Mapping

The following table shows the Java data types derived from VisualAge Generator data item definitions.

VisualAge Generator Data Type	Length in chars or digits	Length in bytes	Decimals	Java Data Type	Maximum precision in Java
CHA	1-32767	1-32767	N/A	String	N/A
MIX	1-32767	1-32767	N/A	String	N/A
DBCS	1-16383	1-32767	N/A	String	N/A
HEX	2-75534	-32767	N/A	Byte[]	N/A
BIN	1-4	2	0	Short	4
BIN	5-9	4	0	Int	9
BIN	10-18	8	0	Long	18
BIN	1-4	2	> 0	Float	4
BIN	5-9	4	> 0	Double	15
BIN	10-18	8	> 0	Double	5
NUM, NUMC	1-4	1-4	0	Short	4
NUM, NUMC	5-9	5-9	0	Int	9
NUM, NUMC	10-18	10-18	0	Long	18
NUM, NUMC	1-6	-6	> 0	Float	6
NUM, NUMC	7-18	7-18	> 0	Double	15
PACK, PACF	1-3	1-2	0	Short	4
PACK, PACF	4-9	3-5	0	Int	9
PACK, PACF	0-18	6-10	0	Long	18
PACK, PACF	1-5	1-3	0	Float	6
PACK, PACF	7-18	4-10	0	Double	15

Data Format Conversion Considerations

Numeric Conversion Considerations

VisualAge Generator to Java:

- 16-18 digit numbers with decimal places are precise to a maximum of 15 digits.

Java to VisualAge Generator:

- Java floating point numbers that have more precision than the corresponding VisualAge Generator item are rounded when converted to VisualAge Generator numbers.
- If a Java number is greater than the maximum for the corresponding VisualAge Generator data item, an exception is raised. The exception may be a java.lang.ArithmeticException or an ibm.cso.CSOException, message CSOE7953, depending on whether the error is detected during marshalling or numeric format conversion.

The calling method must ensure that numeric values are within the range that can be processed by the VisualAge Generator server application.

Character String Conversion Considerations

VisualAge Generator to Java:

- Trailing blanks are truncated when a VisualAge Generator character item is converted to a Java character string.

Java to VisualAge Generator:

- Java Unicode characters that do not have a corresponding character in the target code page are mapped to the SUB character for the code page. No exception is raised.
- Java strings are padded with blanks if shorter than the VisualAge Generator data item, and truncated to VisualAge Generator item length if longer than the VisualAge Generator data item. No exception is raised.

Tracing and Debugging Options

To trace server calls from Java applications, set CSO trace options from the window in which the application is started. To trace server calls from Java applets, set CSO trace options from the window in which the VisualAge Generator Java server is started.

Tracing Errors

To trace errors, use the following:

```
SET CSOTROPT=1 SET
CSOTROUT=trace_file_name
```

The default trace file name is CSOTRACE.OUT in the directory from which the server or application was started. For applets, trace is also written to the UnitOfWorkServerImpl window.

Tracing Service Calls

To trace all service calls to the PowerServer API, use the following:

```
SET CSORROPT=2 SET
CSOTROUT=trace_file_name
```

The default trace file name is CSOTRACE.OUT in the directory from which the server or application was started. For applets, trace is also written to the UnitOfWorkServerImpl window.

Tracing Parameter Contents

To trace contents of parameters before and after server calls, use one of the following:

```
SET CSO_DUMP_DATA=server_appl_name
```

to trace calls to a specific application, or:

```
SET CSO_DUMP_DATA=ALL
```

to trace calls to all applications. Parameter contents are written to file CSODUMP.OUT in the directory from which the server or application was started.

Exception Handling

The server wrapper raises a CSOException when an error is encountered during a call. The catching routine can retrieve the VisualAge Generator middleware CSO error message by calling the CSOException.getMessage() method.

Quarterly Teleconferences—We Want You!

By Rusty Edmister, VisualAge Generator Sales and Technical Sales and Support

Communication is critical to our business. We sometimes hear that we communicate better with our prospects than we do with our current customers. To correct this perception, we plan to host a quarterly communication session via teleconference for our worldwide customers. During this 60-minute call, we will discuss a subject of common interest followed by a question and answer session. As we developed this idea, we investigated the cost of doing such an international teleconference. As you might imagine, it is very expensive for participants who have no toll-free capability. (In the U.S. and Canada, we are able to provide a special number for which there is no charge.) With this in mind, we would like to proceed with the following plan.

Our first teleconference call will take place on October 15 at the time indicated in the following table. We will provide calling numbers for every geography, recognizing that participation outside the U.S. and Canada might be extremely low because the cost may be prohibitive. We will record the calls and make them available at no charge to anyone who might want a copy. You will find instructions later in this article on how to receive a copy of the calls on cassette tape.
The subject of the first teleconference is VisualAge Generator Version 3.0. The first 45 minutes will be devoted to a presentation on the content of Version 3.0 followed by 15 minutes of questions and answers.

If you are outside the United States and Canada, your telephone service provider will charge you for the call. In the U.S. and Canada, we will provide a toll-free number for your use. If the time of the call for your part of the world is not convenient for you, you are welcome to join one of the other calling times. To help us get an estimate of callers that will participate in each of the three calling times, please register in advance by calling 1-800-723-7880 (U.S./Canada) or 402-220-5249 (International). The recording will ask for your name, the name of your company, and the call in which you will participate. Remember, North America calls are toll-free and the PASSCODE for the call (for everyone) is VERSION 3.

Geographic region	Date	Call in Time	Number to Call
Asia	October 15	11:00 AM - 12:00 PM (Tokyo time)	212-547-0192 (International) 888-790-3159 (North America)
Europe/Middle East/Africa	October 15	1:00 PM (Greenwich time)	630-395-0449 (International) 888-455-9647 (North America)
North/South America	October 15	11:30 AM (New York time)	212-547-0176 (International) 800-369-2042 (North America)

If You Cannot Attend the Teleconference

The recorded calls will be available for you to listen to via the telephone through October 17. To hear the recording, call 1-888-839-1160 (U.S./Canada) or 402-220-2274 (International). The PASSCODE to hear the recording is 1527. This passcode is the same for everyone.

We will also record the calls on a cassette tape for those of you who cannot participate in person or via the telephone recording. You can get a copy of the audio tape and presentation materials by sending a request to Rusty Edmister at **edmister@us.ibm.com** or by sending Rusty a fax at 919-254-4820. Be sure to include your name and shipping address with your request.

If the response to this first teleconference is good, we will schedule future calls. We will continue to work to find a less expensive way for these calls. The topics for future teleconferences will include those of greatest interest to our participants. The topic for the call in January will be ENVY, the new library management system in VisualAge Generator V3.0. To help us plan for these calls, please fill in the following form and either fax it to Rusty Edmister at 919-254-4820 or mail it to him at:

Rusty Edmister
IBM Corporation - VisualAge Generator Sales Support
T9EA/002/EBC113
3039 Cornwallis Road
Research Triangle Park, NC 27709
USA

Teleconference Form

(Please fax this form to Rusty Edmister at 919-254-4820 if you plan to participate in some or all of our customer teleconferences)

My name is _____

I represent _____
(company or organization name)

Future teleconference topics that I would like to see scheduled (please list in order of importance or interest to you - most important listed first):

_____

_____

_____

_____

_____

My e-mail address is: _____

My telephone # is: _____

My fax # is : _____

VisualAge Generator Web Pages

Have you visited the Web page recently? We have redesigned the web page and will use it as a main source to communicate with current, future, and prospective customers. See the new design to the right. 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`

E-mail Anyone??

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

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

edmister@us.ibm.com.

Great Lakes User Group Meeting

The Great Lakes VisualAge Generator user group will meet on September 24 and 25, 1997 in Cincinnati, Ohio. For details, contact Rich Kelmer (RKELMER@us.ibm.com or 937-225-6513).

Acronyms

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
DDL	data definition language
DBMS	database management system
DCE	distributed computing environment
EMEA	Europe/Middle East/Africa
GUI	graphical user interface
IBM	International Business Machine
IMS	Information Management System
ITSO	International Technical Support Organization
LAN	Local Area Network
MSL	member specifications library
MVS	Multiple Virtual Storage
NT	Notes
OS/2	Operating System/2
OS/400	Operating System/400
RAD	rapid application development
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
VM	Virtual Machine
VSE	Virtual Storage Extended
WWW	World Wide Web

Comment Form

Please check any appropriate boxes:

- ☐ I'd like to receive future issues of this newsletter. (You need to check this item only if you have not already responded.)
- ☐ I'd like more information about Version 2.2.
- ☐ I'm interested in writing an article to include in *The VisualAge Generator Newsletter*. Subject:_____
- ☐ I'm interested in participating in an AD users' group meeting.
- ☐ I'm interested in participating in a VisualAge Generator users' group meeting.

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

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

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

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

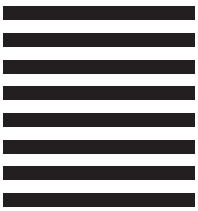
Fold, tape, and mail this page - no postage is required. Or FAX it to (919) 254-0206.

G242-0315-06

Cut or Fold Along Line

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

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

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

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

G242-0315-06

Cut or Fold Along Line

Questions & Answers

Question: Where can I find information on the World Wide Web about the impact of MSLs to ENVY library management system?

Answer: Our plan is to have information on the impact of MSLs to ENVY migration in the next issue of the newsletter, which will also go on the Web.

Question: With the closer relationship between VisualAge Generator and VisualAge, will it be possible to "migrate" components developed in VisualAge Generator to VisualAge and vice versa? Also, will there be restrictions on which VisualAge product can be used (for example, VisualAge for COBOL, C++, Java, and so on)?

Answer: The next release of VisualAge Generator will provide complete interoperability with VisualAge Smalltalk. What this means is that as a VisualAge Generator developer, you will have the full VisualAge Smalltalk environment available to you for developing GUI Client applications as part of client/server systems. Because VisualAge Generator's visual programming environment was derived from VisualAge Smalltalk's, VisualAge Generator developers will find themselves in a familiar environment as they use visual programming to create applications. As a result, VisualAge Generator's GUI capabilities are greatly extended. For example, client database access to IBM and Oracle databases is now available via VisualAge Smalltalk's database parts. VisualAge Smalltalk Web and Lotus Notes parts are just a few of the examples that enable the extension of the traditional client/server application into new territory. There are also over 1000 third-party parts that are now available to the VisualAge Generator developer with useful parts like Business Graphics. To complete this capability, VisualAge Generator parts (Programs, Statement Groups, Records, and so on) can be used within a VisualAge Smalltalk GUI, and can interoperate with VisualAge Smalltalk Parts. Now, GUI business logic can be developed in Smalltalk or VisualAge Generator's familiar 4GL. The choice is yours. Of course, VisualAge Generator's enterprise server capability remains a cornerstone of robust client/server systems.

VisualAge Generator - VisualAge Smalltalk interoperability is more than just interoperable parts. An integrated development environment brings it all together by providing seamless development, test, and code management of the entire client/server application system from one development tool. Components developed in earlier versions of VisualAge Generator will be fully migratable to the next release. As a clarification, VisualAge Generator 4GL code is not converted to Smalltalk; it remains in its familiar format. Also, GUI applications developed in VisualAge Smalltalk can be brought into the next version of VisualAge Generator and used in VisualAge Generator development.

One last note: in VisualAge Generator 2.2, we introduced the ability to create a Java bean that would, in essence, call a VisualAge Generator server by wrapping VisualAge Generator's middleware. We will bring this function forward in the next release. In this way, a Java client can interact with a VisualAge Generator server, adding yet another dimension to VisualAge Generator's network computing capabilities.

The VisualAge Generator Newsletter

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

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

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

G242-0315-06